UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
NEURO- UND BIOINFORMATIK

From the Institute for Neuro- and Bioinformatics
of the University of Lübeck
Director: Prof. Dr. rer. nat. Thomas Martinetz

E-nets as Novel Deep Networks

Dissertation
for Fulfillment of
Requirements
for the Doctoral Degree
of the University of Lübeck

from the Department of Computer Sciences and Technical Engineering

Submitted by
Philipp Grüning
from Lippstadt

Lübeck 2022

First referee: Prof. Dr.-Ing. Erhardt Barth
Second referee: Prof. Dr. Mattias Heinrich

Date of oral examination:

Approved for printing. Lübeck,

Zusammenfassung

Faltungsnetze sind ein prominentes Beispiel dafür, wie das Wissen über neuronale Verarbeitung im visuellen Cortex des Gehirns Bildverarbeitung verbessern kann. Die namensgebende Faltung kann das Verhalten sogenannter „Simple Cells“ im Gehirn nachbilden. Diese Zellen sprechen besonders auf Signale mit einer sogenannten intrinsischen Dimension eins (iD1) an. Faltungsnetze gelten somit als *von der Biologie inspiriert*.

Trotz riesiger Fortschritte erreichen Faltungsnetze noch nicht die Leistung des menschlichen Sehens. Zum Beispiel lassen sich die Netze relativ leicht mit für den Menschen fast unsichtbaren Rauschmustern zu falschen Aussagen hinreißen. Auch eine hundertprozentige Genauigkeit ist auf den meisten Benchmarks noch nicht erreicht.

Nur noch wenige der Forschungsansätze, die sich mit diesen Unzulänglichkeiten befassen, sind biologisch inspiriert. Unsere Meinung ist jedoch, dass insbesondere in diesem Feld spannende Ansätze existieren: vorrangig in der expliziten Nachbildung sogenannter „End-Stopped Cells“ des visuellen Cortex. Diese Zellen fokussieren sich auf Signale mit intrinsischer Dimension zwei (iD2). Derartige Signale treten nicht häufig in natürlichen Bildern auf, codieren aber einen Großteil ihrer Information.

Mit den sogenannten E-Nets präsentieren wir in dieser Arbeit neue Faltungsnetze, die an strategischen Positionen, mittels sogenannter E-Blöcke, direkt iD2 Signale detektieren können. Etablierte Faltungsnetze, wie das ResNet, DenseNet oder MobileNet-V2, können durch Austauschen bestimmter Standardfaltungsblöcke mit E-Blöcken in E-Nets transformiert werden. Eine Evaluation auf verschiedenen Datensätzen und Aufgabenstellungen zeigt, dass diese Netze in vielen Fällen parametereffizienter sowie leistungsstärker als die jeweiligen Ursprungsnetze sind.

Darüber hinaus können wir in dieser Arbeit zeigen, dass die aus der Biologie inspirierten Eigenschaften wie End-Stopping und Hyperselektivität E-Nets robuster gegenüber verrauschten Eingaben machen. Eine klare Regel wird präsentiert, um zu bestimmen, welche Blöcke ausgetauscht werden sollten. Somit lassen sich etablierte Faltungsnetze ohne viel Aufwand in E-Nets transformieren.

E-Nets könnten in Zukunft zu einem Standardverfahren werden, mit dessen Hilfe sich Faltungsnetzen weiter verbessern lassen. Zusätzlich zeigen unsere Ergebnisse, dass die optimale Positionierung der E-Blöcke darüberhinaus großes Potenzial birgt.

Abstract

Convolutional neural networks (CNNs) show how knowledge about the brain's visual cortex can enhance computer vision methods. A convolution can replicate so-called simple cells that strongly react to signals with an intrinsic dimensionality one (iD1). Thus, CNNs are bio-inspired.

Although research on CNNs has progressed, they still do not match human performance in vision. For example, the classification decision of a CNN can easily be altered by noise patterns almost invisible to the human eye. Furthermore, a perfect accuracy of 100% has not been reached yet on most benchmarks.

Nowadays, only a few works aiming to alleviate these shortcomings are bio-inspired. However, biological inspiration can yield exciting new approaches: for example, the explicit modeling of end-stopped cells. These cells only react to signals having an intrinsic dimensionality two (iD2). Such signals are sparse in natural images. Yet, iD2 signals contain a large portion of an image's information.

With E-nets, we present networks that can detect iD2 signals explicitly. Well-known CNNs, such as the ResNet, DenseNet, or MobileNet-V2, can be transformed easily into E-nets. To this end, certain standard convolution blocks are exchanged with so-called E-blocks. Our evaluation based on different datasets and computer vision tasks shows that often when transformed into an E-net, the CNN's performance increases while decreasing the number of parameters.

Moreover, we can show that behaviors found in biological vision, in this case, end-stopping and hyperselectivity, are present in E-nets. Hyperselectivity can increase an E-net's robustness against noisy inputs. We present a straightforward rule on which blocks to replace with E-blocks so that many state-of-the-art CNNs can be effortlessly transformed into E-nets.

Therefore, using E-nets may become a standard approach to enhance CNNs further, especially considering that there is even more potential improvement if the optimal positioning of E-blocks is known.

Contents

Abbreviations

ANN artificial neural network

BCE binary cross-entropy

BN batch normalization

CIGA clipped iterative gradient ascent

CNN convolutional neural network

DRIVE digital retinal images for vessel extraction

DW depthwise

FGSM fast gradient sign method

FB factorized bilinear

FCN fully convolutional network

FP false positive

FN false negative

iD intrinsic dimensionality

IN instance normalization

IoU intersection over union

IQA image quality assessment

ISBI international symposium on biomedical imaging

ISIC international skin imaging collaboration

LITW LIVE in the wild

LN linear non-linear

MAC multiply accumulate

MCC Matthews correlation coefficient

MLP multilayer perceptron

PLCC Pearson linear correlation coefficient

POCP percentage of changed predictions

SOTA state-of-the-art

SROCC Spearman's rank-order correlation coefficient

TN true negative

TP true positive

Chapter 1

Introduction

Very early in the 1940s, a general model of brain neurons [JL43] led to artificial neural networks (ANNs), including the well-known multilayer perceptron (MLP) [RHW85]. This network consists of many neurons arranged in layers communicating with each other. However, today's broad adoption of artificial intelligence has a more specific root in biological vision. The eye's retina converts incoming light into electrical signals that are carried via the optical nerve to the primary visual cortex (V1). There, a hierarchy of cell layers processes the input. Each cell passes an electrical signal to a cell from another layer if a particular stimulus (e.g., a particular object in an image) is observed [LBH15]. The complexity of a preferred stimulus increases with each layer, and visual cortex cells can be categorized depending on the complexity [HW68]. Most notably, so-called *simple cells* only react strongly to a line or edge with a particular orientation at a specific position in the visual field. In addition, so-called complex cells act similarly but react to a broader visual field area [HW65].

Mathematically, this behavior can be simulated by a convolution between an image and a filter kernel. More precisely, a simulated visual cortex layer (convolution layer) contains parallel convolutions with many different kernels that are further transformed by a non-linear function [Fuk80]. Stacking layers into a long sequence creates a hierarchy with increasing stimulus complexity [LBH15]. This subclass of the MLP is better known as a convolutional neural network (CNN), and its strong performance on intricate computer vision tasks stoked new interest in applying artificial intelligence in various tasks. For example, in 2012, the automatic classification of images was very much an unsolved problem. The ImageNet [Rus+15] dataset was especially complex, containing more than a million images depicting animals, insects, and objects of 1000 different categories, such as snakes, dogs, spiders, household items, and vehicles. In 2010, the best approach [Yua+10] did not produce the correct class in almost a third of all test images, even when allowed five guesses per image (28.2% top-5 error rate). In 2012, the now famous AlexNet [KSH12] almost halved the number of wrong classified images (15.3 % top-5 error), outperforming other approaches by a wide margin. Nowadays, state-of-the-art (SOTA) CNNs correctly classify more than 90% while needing only one guess per image (top-1 error) [Pha+21].

CNNs and MLPs are machine learning models. Hence, when for example, presented with images and their respective classes (training data), they gradually learn a general rule about which shapes, colors, and textures comprise an image that depicts a specific object. This process is called *induction*. Among many other factors, the structure of the model strongly affects which particular rule is induced from data. The entirety of all factors is called a *bias*. Depending on the task, a bias can be beneficial or detrimental. The CNN is a shining example of how an appropriate bias can help to solve a problem: by rudimentary modeling of how our brains perceive images, CNNs took automatic image classification to a new level. When looking at the SOTA in CNNs, biological inspiration plays a minor role, and today's research is more and more driven by engineering [He+16; How+17; TL19a].

However, we think that one valid path of further improvement of CNNs is looking at the visual cortex again. Apart from simple and complex cells, it contains hypercomplex cells [HW68]. We regard a subclass of those cells, so-called *end-stopped* cells, as crucial for vision. When first discovered, the behavior of end-stopped cells was described to be similar to simple cells. I.e., they responded to a line segment with a particular orientation. However, if that line segment exceeded a certain length, the response was inhibited, i.e., stopped by the ends of that line segment [Ros77]. In comparison, today's definition is more general: end-stopped cells respond to highly informative subregions in images, so-called *2D regions* [ZB90; ZBW93]. CNNs do not explicitly model end-stopped cells since a single convolution layer cannot directly recognize these regions. Nonetheless, one can assume that 2D regions are detected when using sequences of convolution layers [BZ98]. However, considering the importance of end-stopped cells in the visual cortex, we hypothesize that an *explicit implementation* of end-stopping should yield an improved bias for CNNs in computer vision which should at least improve their efficiency.

In the same spirit, CNNs display the massive potential of an explicit implementation of ideas from biological vision. From a biological and technical perspective, CNNs are an outstanding solution to many data-driven computer vision tasks [Li+21]. Thus, one might wonder why an MLP does not behave like a CNN. When given the right size, an MLP layer is more than capable of mimicking a convolution. Thus, one would expect that the performance of MLPs should be at least on par with CNNs. However, MLPs usually infer different solutions given image data. Hence, there is evidence that the inductive bias of an explicit implementation, i.e., using a CNN instead of an MLP, leads to better performance. Thus, one might wonder if the explicit implementation of other behaviors, e.g., end-stopping, may further improve the inductive bias of CNNs.

1.1 Why a proper inductive bias is important for machine learning

Induction is the process of creating a general rule, also called a hypothesis, based on evidence. Induction is used, for example, in software programs that learn to classify images based on a training dataset containing image-class pairs (evidence), where the classes were determined beforehand by a human annotator. Such a program can be considered the manifestation of a general rule in the form of a mathematical formula that produces a class probability given an image. Finding an appropriate hypothesis based on evidence is not a trivial task: there may be many hypotheses that explain the training data well – but only a few prove themselves useful when applied to new and unseen data.

Selecting a reasonable hypothesis while discarding the others needs extra-evidential reasoning. For example, considering the evidence, the rules 'all emeralds are green' and 'all emeralds are grue (green before tomorrow and blue afterward)' are both valid [Goo83]. Our intuition that the first hypothesis is more likely than the second cannot be based on today's data but only on certain beliefs. In practice, one needs to determine from a plethora of algorithms which one might be the most suitable – even most algorithms themselves need to choose from an infinite amount of hypotheses. What is needed is an inductive bias, i.e., a set of beliefs, heuristics, or even wisdom, to induce a hypothesis (for more information, see, for example, Watanabe [Wat85]).

In machine learning, a learner is an algorithm that, during a training phase, fits a hypothesis to a training dataset. Mathematically, the learner can be seen as a possibly infinite set of functions called a machine. During training, one of those functions is selected to fit the data well, which is usually quantified by some loss function that compares the function's output to the labels of the training data. The machine's particular set of functions is one example of an inductive bias. In Figure 1.1, three different machines try to solve a regression problem on parabolic data. Each machine contains polynomials of different degrees, one, two, and ten from left to right, respectively. Although already infinitely large, the first machine is a subset of the second machine, which is a subset of the third machine. Thus, the capacity increases with each machine. The line fit is not capable of fitting the training data. However, it is unlikely that a variance in the training set, e.g., more missing data or an increase in noise, would significantly alter the difference between the training and test error. The 10th-degree polynomial perfectly fits the noisy training data, which causes a very complex hypothesis that does not generalize well to the test data. Only the second-order polynomial is robust enough to disregard the noise in the data while fitting the training data well enough.

This simple example illustrates one of the critical challenges in machine learning: finding the proper bias for a specific problem. A learner needs to contain a function that

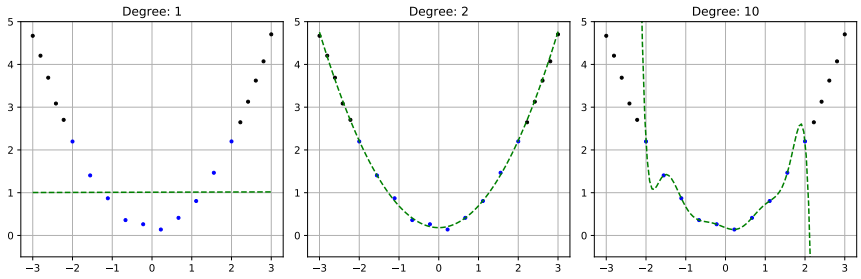


Figure 1.1: Regression example: blue dots show the training data, and black dots the test data unknown to the learner. The green curves show the hypotheses generated by different learners. Each panel shows a learner's result that fits a polynomial to the training data by reducing the squared error—the polynomial degree of each learner increases from left to right. The training data stem from a second-degree polynomial with a small addition of Gaussian noise. The second-degree polynomial (middle panel) has the perfect bias to fit the data well, disregarding noise and yielding a feasible extrapolation.

can fit the data, and it needs to be prone to select this function even when presented with only incomplete and possibly noisy training data. Note that the 10th-degree polynomial machine contained a suitable quadratic solution in our example, but the training algorithm failed to select it.

1.2 Why biology is a viable inspiration for an inductive bias

Regarding the visualization of the above-given one-dimensional example, Occam's razor (a typical bias in induction) [1] suggests a quadratic equation: it is a simple solution that still explains the training data well enough. Hence, a machine learning practitioner could find this reasonable solution by simply 'looking at the plot'.

However, today's most exciting tasks do not contain merely a single dimension that can be quickly plotted, and a reasonable approach is clearly visible. Instead, the input data are very high-dimensional, and thus, visualization approaches usually are not enough to acquire an intuition about a suitable machine for the task. This problem fully applies to computer vision tasks, where an input dimension [2] of 1000 is considered small, and typical image classification data have inputs of at least 150 thousand or more dimensions. For humans, vision is an intuitively easy task. Nonetheless, it is tough to formalize or describe and, thus, difficult to transform into an algorithm. Accordingly,

[1] Note that in machine learning, a similar inductive principle is structural risk minimization [Vap99].

[2] A gray image is, for example, a matrix of a certain height h and width w. In many applications, images are processed as vectors of dimension $h \cdot w$.

creating a solution from scratch is a daunting challenge. However, since the brain (from animals and humans alike) works well with visual tasks, it is sensible to draw inspiration from it. Although research has not progressed enough to understand the brain entirely, some basic paradigms can be established. For example, intelligent behavior stems from several simple interconnected units, and those units are quite adaptable to different inputs. I.e., they can learn a representation of the input that makes it easy for a learner to process. In contrast, many other machine learning approaches depend on carefully selecting handcrafted input features to convey the correct information to the learner.

The first mathematical model of a neural network was described by McCulloch and Pitts [MP43]. Later, Rosenblatt [Ros57] created the brain-inspired perceptron, which can be used as a binary classifier. The perceptron classifier is a weighted linear combination of the input with an additional offset: $\mathbf{w}^T\mathbf{x} + b$. If the value of this sum is greater than zero, a one is returned, otherwise a zero. This linear classifier is not very powerful. However, when combining several perceptrons and adding a non-linearity, such as the rectified linear unit (ReLU) [3], a very capable MLP [RHW86] can emerge. Note that a weighted sum with non-linearity is called a neuron because it is a simplified model of a brain neuron. A layer is a set of neurons that can be easily described via matrix multiplication, $\mathbf{h} = ReLU(\mathbf{W}\mathbf{x})$. Using several layers in a feedforward MLP is essentially a sequence of matrix multiplications with non-linearities: $\mathbf{y} = \mathbf{W}_n(...\mathbf{W}_3 ReLU(\mathbf{W}_2 ReLU(\mathbf{W}_1\mathbf{x}))...)$. The universal approximation theorem [HSW89] states that, given a sufficient width, any MLP with one hidden layer (i.e., a matrix, a non-linearity, and another matrix in sequence) can approximate any well-behaved function. Since a high capacity leads to MLPs being prone to overfit, they went out of fashion for quite some time. Numerous factors played into the revived widespread use of neural networks, [4] but for many, the starting point was the successful application of an MLP-subclass, the CNN, incorporating inductive biases inspired by biological vision.

In a very simplified model, light hits the eyes, the retina cells transform the incoming light into a slightly pre-processed electrical representation of the seen image. The optical nerve carries the signal to the brain, where it eventually arrives at the primary visual cortex (V1). This is the first area of the brain that processes the visual input significantly, and this area strongly inspired CNNs.

With their ground-breaking work, Hubel and Wiesel [HW65] examined neurons in the visual cortex of cats, measuring the neuron response to specific stimuli (see Figure 1.2). Interestingly, in V1, most neurons only reacted to lines and edges with a specific orientation. They dubbed cells that displayed this behavior *simple cells*. A

[3] $ReLU(x) = 0$ if $x < 0$ else x.

[4] E.g., using larger datasets, using the ReLU as non-linearity instead of saturating functions such as the sigmoid function, or using graphics processing units (GPUs) to train larger networks on more data feasibly.

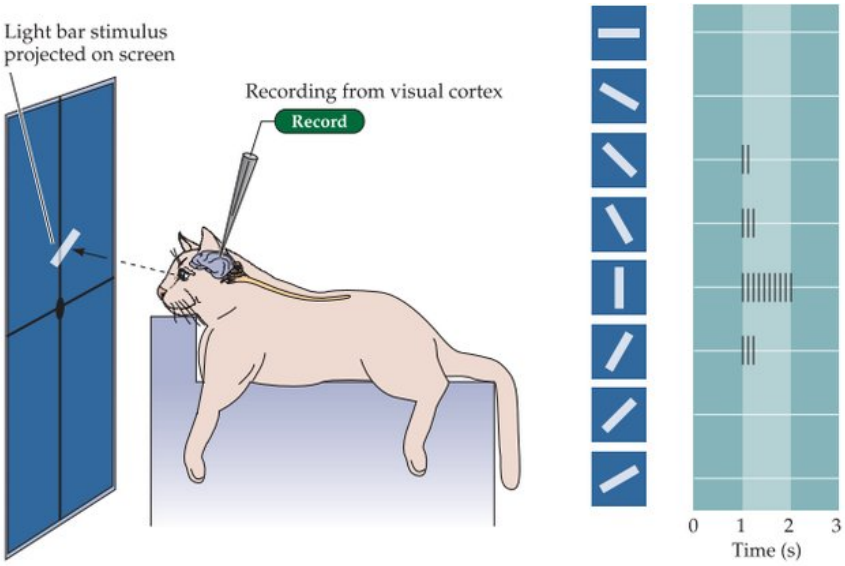


Figure 1.2: Experimental setup of Hubel and Wiesel [HW65] to examine neurons of a cat's visual cortex. Different cell types were discovered. For example, so-called simple cells strongly react to lines with a specific orientation (here, vertical lines) at a particular position in the visual field. Figure adopted from Berga [Ber19].

simple cell only reacts to the visual stimulus of a specific area of the visual field, which is also called the receptive field.

To model a simple cell behavior with a mathematical neuron, so-called oriented filters are used. The neuron's weights could, for example, resemble a Gabor filter [Mar80] being a sinusoidal function gated by an exponential function. These filters respond to a change in brightness with a certain frequency and orientation at a certain position.

Given the right weights, a convolution with a non-linearity can exhibit this behavior. Note that many vision models tend to learn oriented filters by themselves when presented with natural images, e.g., Olshausen and Field [OF96]. Similarly, CNNs learn edge detectors of different orientations in early layers: Krizhevsky, Sutskever, and Hinton [KSH12] show that their CNN's first convolutional layer contains 11×11 filters that are either frequency- or orientation-selective or strongly respond to certain color blobs, see Figure 1.3.

Apart from simple cells, the visual cortex contains so-called complex cells. Similar to simple cells, they react to specific orientations. In contrast, they are less restricted to a certain position in the visual field. In neural networks, this behavior is achieved by using max- or mean-pooling layers after convolutions.

Hubel and Wiesel's [HW68] experiments show that the visual cortex is hierarchical. Their model of the visual nervous system is divided into layers with mainly simple cells, complex cells, and even hypercomplex cells of different complexity orders. The

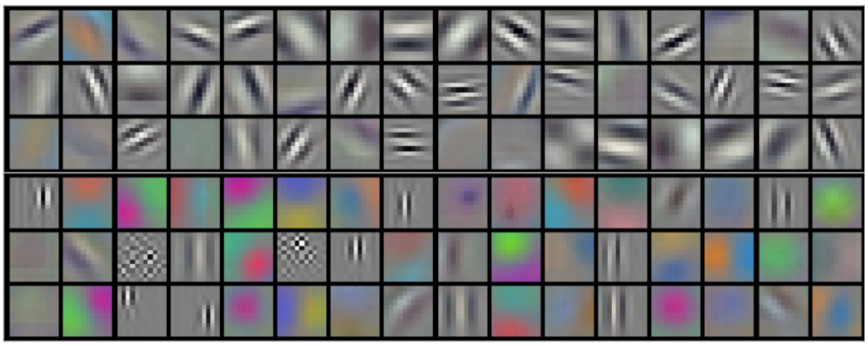


Figure 1.3: $11 \times 11 \times 3$ filter kernels of the Alexnet's first layer: many of the filters are edge detectors, matching the behavior of simple cells in the visual cortex. Figure adopted from Krizhevsky, Sutskever, and Hinton [KSH12].

stimuli that create a high response (also called *optimal stimuli*) gain complexity with each layer, from simple edges and lines (simple- and complex cells) to corners and more elaborated geometric shapes (hypercomplex cells); eventually, there may even exist so-called *grandmother cells* [Gro02]. Such neurons may only respond to a very particular stimulus, your grandmother, and are invariant to a broad spectrum of transformations. Among this plethora of different neurons, Hubel and Wiesel found end-stopped cells reacting to very specific stimuli that contain a high amount of information. Here, we mainly mention Hubel and Wiesel because their work is the most prominent research on the visual cortex (in 1982, they were awarded the Nobel prize). However, of course, other researchers made important contributions to this field as well, e.g., Movshon, Thompson, and Tolhurst [MTT78], Gilbert [Gil77], or Sillito [Sil75].

Inspired by Hubel and Wiesel's findings and other works [GRMB72], the Neocognitron [Fuk80] was created in 1980. Here, the concept of *weight sharing* was already introduced. In the brain, a simple cell is selective to a certain orientation and a certain position. However, it is likely that a simple cell with the same orientation selectivity exists that is only tuned to a different position in the visual field. The set of all simple cells with the same optimal stimulus but that focus on different positions can be summarized by a single convolution operation. A kernel, defining the optimal stimulus, is compared to any position of the input, and an output feature map is produced. Essentially, the feature hierarchy is achieved by creating a sequence of convolution layers with non-linearities, similar to an MLP. Back then, the training scheme, however, resembled more a clustering algorithm that differs from today's backpropagation [Lin70; RHW86]. One of the earliest CNNs trained with backpropagation was the so-called LeNet [LeC+89; LeC+98].

More than thirty years after the Neocognitron, the use of convolution sequences gained widespread recognition: with a combination of convolutional layers with ReLU, max-pooling layers, and a subsequent MLP, the AlexNet [KSH12] won the 2012 ImageNet

Figure 1.4: Feature hierarchy illustration: a CNN classifies an input image (left) by processing it sequentially with several convolution layers. The bottom circles on the right represent the input pixels of an image, i.e., the visible layer. Each circle above the visible layer symbolizes an artificial neuron of the so-called hidden layers. The small image patches within these circles visualize the features extracted by the neuron (patches based on Zeiler and Fergus [ZF14]). Hence, the neurons of the first hidden layer (right above the visible layer) process the input pixels, extracting features such as lines, edges, and color blobs. Subsequent layers extract increasingly complex features – from corners and edges to object parts. Finally, the output neurons (uppermost circles on the right) can recognize the objects important for the image's classification. Here, for example, a CNN may classify the input image as 'person', distinguishing it from other images showing cars, dogs, or cats. Figure adapted from Goodfellow, Bengio, and Courville [GBC16].

challenge [Rus+15] by a large margin. Compared to other neural networks, several more layers were used, coining the term *deep learning*. Training these layers efficiently was only possible by using GPUs. Using convolutions improved upon standard MLPs in reducing the number of weights drastically, thus, likely reducing the tendency to overfit. Still, the AlexNet maintained a feasible capacity because similar to the brain's visual system, interconnected simple cells could extract and represent a hierarchy of many different features that are important for vision tasks (see Figure 1.4 for an example) – i.e., a proper bias was imposed.

From the AlexNet, new architectures arose, helping with problems like the vanishing gradient [IS15], lesser effectiveness after a certain depth [He+16], and a too-large model size for deep learning on mobile devices [How+17]. From 2012 to 2020, CNNs

dominated popular benchmarks such as ImageNet [Den+09] or Cifar-10 [KNH10]. None of these works directly refer to a biological system that is recreated; the mentioned approaches come from practical challenges when working with CNNs that were solved by skilled engineers. However, it is worth mentioning that both the AlexNet and the Neocognitron were also not intended to exactly model the visual cortex. For example, complex cells mainly react to moving stimuli which is infeasible to model in a CNN since the network's input is just one stationary image.

More recent work showed that the self-attention-based transformer architecture, originating from natural language processing, can outperform CNNs by a small margin when trained with enough data. Dai et al. [Dai+21] argue that this is due to a larger capacity, but transformers lack the proper inductive bias leading to lesser generalization when facing too little data. Thus, it is likely that for practitioners, CNN architectures remain the most promising tool for machine learning in vision since the inductive bias leads to better results for datasets with less than several million training images. Accordingly, research on CNNs and proper inductive biases is a viable field, and in this work, we aim to refocus on the initial source of inspiration for CNNs: the visual cortex. As described above, simple- and complex cells inspired convolution- and pooling layers, although pooling layers become less and less prevalent in newer architectures. Yet, few works attempted to explicitly model end-stopped cells that usually occur in the intermediate stages of the visual cortex.

1.3 Why CNN architectures should model end-stopped cells

The term end-stopping was coined by Hubel and Wiesel when they observed neurons that responded to lines with a particular orientation and a certain length; once the line segment's length exceeded a particular threshold, an end-stopped neuron's activation was drastically reduced. This behavior inspired mathematicians and engineers to search for mathematical analogies to end-stopped cells. From their perspective, an artificial end-stopped neuron is a cell that is selective to a broader range of signals that are defined by their intrinsic dimensionality (iD). In this work, when we use the term *end-stopping*, we refer to the behavior of such artificial end-stopped cells.

When regarding an image, different parts of it (patches) contain different amounts of information. Each patch can be assigned to one of three classes of iD. Regarding an image as a function $f(x, y)$, 0D patches are small sub-regions, where $f(x, y)$ is constant. [5] For sub-regions corresponding to 1D patches, there exists a direction $ux + vy$ and a one-dimensional function g, such that $f(x, y) = g(ux + vy)$. I.e., $f(x, y)$ is constant when $(x, y) \cdot (u, v)^T = 0$ and only changes along one dimension. For *2D patches*, none of the above constraints can be formulated; the function changes in any

[5] I.e., for all $x \in [a, b]$ and $y \in [c, d]$, $f(x, y) = const.$ with $a, b, c, d \in \mathbb{N}$, $a < b$, and $c < d$.

direction. Statistically, 2D patches are the least common ones in an image [BW00; ZBW93]. Thus, considering information theory, the information content ($log_b(\frac{1}{p})$) of a low probability (p) event is very high. Given the vast amount of information, any efficient visual system must remove redundant information and focus on the relevant parts of the input. Arguably, millions of years of evolution have led to mammalian visual systems being efficient regarding complexity, energy consumption, and information. Thus, it is easy to see why end-stopping has emerged in our visual system.

Apart from image statistics, 2D patches contain interesting properties when investigating them using tools from differential geometry. The geometric perspective regards a grey-scale image as a surface $f(x, y)$, where pixels with high values (lighter gray or white) form peaks. If this surface, or a sub-area of it, is flat, the area's iD is zero. An iD of one corresponds to parabolic surfaces. Elliptic or hyperbolic surfaces have an iD of two. The Gaussian curvature K is adequate to determine 2D regions since it is zero for planar and parabolic surfaces, negative for hyperbolic, and positive for elliptic surfaces. One can show that to determine a compact surface, only the points with $K(x, y) \neq 0$ need to be known [MB00].

The two-dimensional Gaussian curvature is defined by the multiplication of the two main curvatures: two orthogonal directions, one corresponding to the maximum and one to the minimum curvature. Accordingly, curvature estimation, and thus, detection of 2D regions, needs to employ a logical AND combination. The K-value is only high if the first AND second main curvatures have a high value, see Figure 1.5.

Looking at the Gaussian curvature reveals how a 2D detector can be built: one can use a pair of oriented filters and combine their outputs with an AND. Only if both filters yield a high output for a pixel (i, j) the image function changes along both filter orientations; thus, the region around (i, j) is likely a 2D region. The Gaussian curvature analogy suggests using multiplication; however, we show results that indicate that other operations, such as the minimum operation, are also feasible.

Typical CNNs do not model AND combinations explicitly: convolutions can be described as matrix multiplications, and the weighted sums of matrix multiplications rather correspond to OR combinations. However, it can be shown that sequences of convolutions with ReLU can extract 2D regions [BZ98] and create AND combinations implicitly.

Nevertheless, it seems likely that, given the importance of 2D regions, employing AND combinations would lead to more efficient networks because they may extract crucial information more effectively.

This introduction concludes the primary motivation for this thesis. Regarding machine learning for vision tasks, biological inspiration drastically advanced the SOTA with CNNs. However, CNNs lack the explicit implementation of AND combinations. Nevertheless, AND combinations of oriented filters are helpful in extracting 2D regions of images that contain high amounts of information. Moreover, 2D region detectors have been observed in the brain in the form of end-stopped neurons. Hence, we will

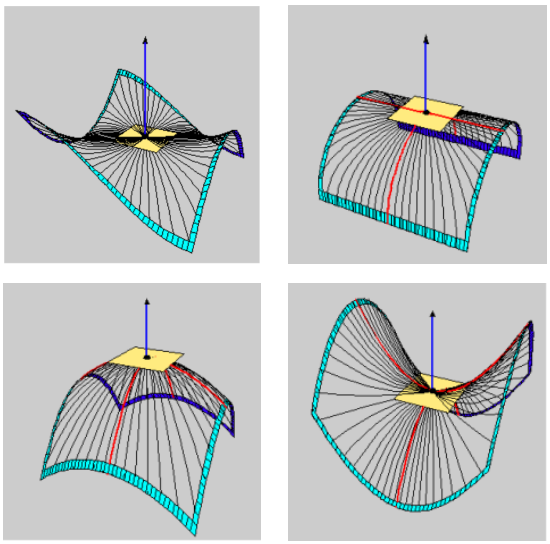


Figure 1.5: From a geometrical perspective, an image can be seen as a surface $f(x, y)$. The Gaussian curvature K is defined as the multiplication – more general, the logical AND combination – of the principal curvatures k_1 and k_2. K is only non-zero if both curvatures are non-zero. In these images, with patches, we refer to a small area around the intersection of the blue arrow and the surface. Redundant patches with iD 0 (see top left panel) have a zero Gaussian curvature, as well as 1D patches (top right). Elliptic surfaces have a positive K value (bottom left), and hyperbolic surfaces have a negative K value (bottom right). Both surfaces may correspond to highly informative 2D regions in images. Figure adapted from [Gor22]

explore how the explicit modeling of AND combinations of filter pairs (in principle, of two simple cells) can improve neural networks and present end-stopped (E)-nets.

1.4 Related work

Efficient architectures The desire to find more efficient networks that keep their outstanding performance while fitting on small, e.g., mobile devices, has led to an active research field in architecture and layer design. Well-known networks such as the MobileNet-V2 [San+18a] employ depthwise separable convolutions and mobile inverted bottlenecks. These types of layers are also used in the EfficientNet [TL19a], which refutes the notion that a higher number of parameters is the best way to increase generalization [Hua+19], by combining network depth, width, and resolution intelligently. The EfficientNet, and many similar architectures, are inspired by several search algorithms [Tan+19; XY17; VD18] that attempt to find a good trade-off between efficiency and accuracy. Another approach to gaining efficiency is the reduction of parameters by pruning the weights of the network [Li+17a; FC19]. Moreover, changing the structure of network connections and using grouped convolutions can yield more efficient CNNs [Zha+18b].

The structure of the E-net's main building block, the E-block (see Section 3.5), is similar to the MobileNet-V2's mobile inverted bottleneck. Since both the E-block and the bottleneck use efficient depthwise (DW) convolutions (see Section 2.2.2), both networks often use fewer parameters than alternative networks using standard convolutions. However, in addition, we will show that to obtain a certain test score, i.e., a certain level of generalization, E-nets need fewer layers than standard CNNs.

Multiplicative combinations and second-order interactions Apart from detecting 2D regions in images, there are other motivations for multiplicative combinations of hidden transforms: for example, gating [CSDS16], attention [Okt+18], and channel re-weighting [HSS18].

With the term *second-order interaction*, we here refer to a second-order Volterra series [Vol59] containing a term that linearly combines input pairs (x_i, x_j) that are multiplied: $f(\mathbf{x}) = \sum_i \sum_j m_{i,j} x_i x_j; \mathbf{x} \in \mathbb{R}^n$. Second-order interactions can be a useful tool to model and analyze complex cells of V1 [Rus+05; BW06], influencing works on bio-inspired image classification models. For example, Bergstra et al. [Ber+09] showed that second-order interactions improved simple ANNs on image classification. Similarly, Zoumpourlis et al. [Zou+17] improved the generalization of a Wide ResNet [ZK16] by exchanging the CNN's first standard convolution layer with a second-order convolution layer. In all these works, second-order interactions are used to increase the capacity of a unit with the intention of better modeling the behavior of more complex cells in V1.

Apart from approaches that were directly inspired by biological vision, using higher-order polynomials to increase model capacity has been known for years. For example, with sigma-pi networks [MK90; RHM+86]. More recently, Chrysos et al. [Chr+20] proposed a CNN based on polynomial non-linearities. Second-order interactions also play a role with Transformers [Vas+17; Dos+20] to compute attention weights.

Lin, RoyChowdhury, and Maji [LRM15] proposed so-called bilinear CNNs, where the outer product of two feature vectors (coming from two separate CNNs) is computed. The resulting combined feature vector is a pooled version of local pairwise interactions, and the approach can yield better performance in fine-grained recognition. Derivations of bilinear models are used in several different applications [Gao+16; UGL17; Cho+16], including image quality assessment (IQA) [Zha+18a]. Li et al. [Li+17b] presented a factorized version of the bilinear CNN, a so-called factorized bilinear (FB)-net. In Section 3.6.1, we show that the second-order interactions of E-nets can differ from second-order interactions modeled by FB-nets and point out the resulting architecture differences.

Compared to the works mentioned above that increase capacity to better model *some* complex behavior, our E-nets use second-order interactions with a more specific goal: to model end-stopping. Interestingly, with E-nets, there is not even an explicit need for using multiplications. Other functions, such as the minimum operation, can be used to model AND connections. However, it is worth mentioning that one can use the minimum to roughly approximate a multiplication: in MinConvNets [Yan+22], instead of multiplying weights and signals, the minimum is taken, reducing the computational burden introduced by the many multiply accumulate (MAC) operations that are performed when using a CNN.

1.5 Contributions and outline

Chapter 2 gives a quick overview of concepts, related architectures, and datasets used in this work. Chapter 3 presents E-nets and explains how they are created from baseline architectures. Moreover, analytical results concerning E-nets are presented.

Chapter 4 presents implementations and discusses several applications of E-nets. Most results are based on already published work in the following order: we present efficient E-nets for image classification [GMB20a] and IQA [GB21b]. Next, we outperform SOTA CNNs in image classification while enhancing the robustness against adversarial attacks and JPEG compression artifacts, furthermore proving that end-stopping occurs in E-nets [GMB22]. Subsequently, we show that E-nets that AND combine oriented filters via the minimum operation are a viable alternative to E-nets based on multiplication [GB21a]. Another publication proposes that logarithmic addition is feasible to train networks that compute large products efficiently [GMB20b]. Furthermore, we provide results where we optimize E-nets by placing end-stopped neurons at strategic positions

and propose a genetic algorithm to find those positions [GB23]. Lastly, we present results that investigate medical image segmentation with E-nets.

In Chapter 5, we discuss the results and provide an outlook for further research. Finally, the Appendix offers additional information on experimental details and additional results.

Chapter 2

Basics

This chapter aims to give additional background information on parameter reduction, specific convolutional neural network (CNN) architectures, operations, and datasets. It describes how convolutions and depthwise (DW) convolutions are computed and how these steps reduce the number of model parameters (i.e., unique weights that need to be stored in memory) compared to a traditional multilayer perceptron (MLP). We discuss these CNN operations and additional block structures because they heavily influenced our design choices. With blocks, we denote specific sequences of convolutions and non-linearities that are often the main elements for a particular CNN, such as the ResNet, PyrNet, or DenseNet. Furthermore, we give information about the two image classification datasets, Cifar-10 and ImageNet, since they are relevant throughout this work. First, we briefly explain the mathematical notation used to refer to images, activation tensors, and subsets (e.g., single pixels or channels) of these tensors.

2.1 Tensor notation

This work focuses on machine learning tasks that process 2D colored images. Thus, the standard input for each of the presented architectures is a tensor $\mathbf{T} \in \mathbb{R}^{h \times w \times d}$ with height h, width w, and $d = 3$ color channels red, green, and blue (RGB). Once processed by a convolution, $\mathbf{T}$ can have a channel dimension d far larger than three, and the width and height may be altered. In this work, we refer to a convolution layer's output channels, showing the neuron activations, as *feature maps*. The m-th single feature map of $\mathbf{T}$ is referred to as $\mathbf{T}^m$ or $\mathbf{T}[:, :, m]$. In the latter form, ':' denotes that we speak of all values of the respective dimension. I.e., in the case of $\mathbf{T}[:, :, m]$, we refer to all values along the width and height of $\mathbf{T}$, but we only regard the m-th entry along the feature map dimension. We point to a single pixel at position (i, j) of $\mathbf{T}^m$ with the expression $\mathbf{T}[i, j, m]$, while $\mathbf{T}[i, j, :] \in \mathbb{R}^d$ is the feature vector at (i, j). CNNs usually process stacks of tensors (including images), also called *batches*. A batch is denoted as a four-dimensional tensor $\mathbf{T} \in \mathbb{R}^{b \times h \times w \times d}$, being b 3D tensors 'stacked on top of each other'. These tensors are processed independently by the network. I.e., the activations of tensor $\mathbf{T}[i, :, :, :]$ do not affect the activations of $\mathbf{T}[j, :, :, :]$, $i, j = 1, ..., b$, $i \neq j$. If scalar values are combined with vectors or tensors in an equation, these scalars are

broadcasted. E.g., '$\mathbf{T} - x$' means that x is subtracted from each of the $h \cdot w \cdot d$ entries of $\mathbf{T}$.

2.2 Parameter reduction with convolution layers

2.2.1 Comparing an MLP to a CNN

Compared to MLPs that use dense layers, CNNs already drastically reduce the number of weights because their primary operation, the convolution, enforces (i) a locality constraint and (ii) weight sharing. Constraint (i) means that to transform a single feature vector $\mathbf{T}[i, j, :] \in \mathbb{R}^c$ of a tensor $\mathbf{T} \in \mathbb{R}^{h \times w \times c}$, only the position (i, j) and its neighbors are used as input. Furthermore, with (ii), the same weights are used to transform each feature vector.

For comparison, a dense layer, computing a matrix multiplication of the vectorized tensor $vect(\mathbf{T}) = \mathbf{x} \in \mathbb{R}^{hwc}$ and $\mathbf{W} \in \mathbb{R}^{d \times hwc}$ uses all $w \cdot h \cdot c \cdot d$ entries of the weight matrix $\mathbf{W}$ to compute a new output vector $\mathbf{z} = \mathbf{W}\mathbf{x} \in \mathbb{R}^d$. Note that if the image shape $h \times w$ is to be preserved and d_0 is the new number of channels, a very high output dimension $d = h \cdot w \cdot d_0$ is needed, possibly resulting in millions of parameters for $\mathbf{W}$. Furthermore, $\mathbf{W}$ can only process inputs with a fixed shape h and w – convolutions do not have this limitation.

When applying a locality constraint (i) with a window of $k \times k$ to a weight matrix $\mathbf{W}_{LC}$, only $k^2 \cdot c \cdot d$ of $\mathbf{W}_{LC}$'s entries per row are non-zero. When using weight sharing (ii), each row has the same $k^2 \cdot c \cdot d$ non-zero entries. The entries are only shifted to transform the pixel corresponding to the respective row; see the Matrix in Equation 2.10 for an example. The following subsection discusses the number of parameters used for convolutions.

2.2.2 Convolution vs. DW convolution

A convolution [1] transforms an input tensor $\mathbf{T} \in \mathbb{R}^{h \times w \times d}$ into an output tensor $\mathbf{T}_{out} \in \mathbb{R}^{h \times w \times d_{out}}$ with d_{out} being the desired number of feature maps. For an output feature map m, each pixel (i, j) of the output tensor is computed by multiplying $\mathbf{T}[i, j, n]$ and its neighbors with specific weights stored in the kernel [2] $\mathbf{K} \in \mathbb{R}^{k \times k \times d \times d_{out}}$ and summing over these products:

$$\mathbf{T}_{out}[i, j, m] = \sum_{a,b,n} \mathbf{T}[i + a, j + b, n]\mathbf{K}[a, b, n, m]; \tag{2.1}$$

with $a, b \in \{\lfloor -\frac{k}{2} \rfloor, ..., \lfloor \frac{k}{2} \rfloor\}$, $m \in \{1, ..., d_{out}\}$, and $n \in \{1, ..., d\}$. Accordingly, for a standard convolution, the entire depth of each pixel is incorporated into the sum.

[1] For simplicity, with stride= 1 and zero padding that preserves the height and width.

[2] $\mathbf{K}$'s width and height can differ, but throughout this work, only square kernel shapes are used.

Especially in deeper stages of a CNN, where d and d_{out} can be large values, the number of parameters for $\mathbf{K}$ can increase drastically because:

$$\#\text{Parameters}(\mathbf{K}) = d \cdot d_{out} \cdot k^2. \tag{2.2}$$

Research on more efficient CNNs [How+17; Cho17] showed that a standard convolution could be approximated by a factorized approach that uses a DW convolution and a subsequent standard convolution with $k = 1$ (i.e., a linear combination of the feature maps) – which is also called 1×1 *convolution*. The term 'depthwise' denotes that, instead of summing along the entire depth of an input tensor (along n in Equation 2.1), each input feature map is processed separately:

$$\mathbf{T}_{out}[i, j, m] = \sum_{a,b} \mathbf{T}[i + a, j + b, m]\mathbf{K}^{DW}[a, b, m]. \tag{2.3}$$

Thus, for each input feature map m, one two-dimensional kernel $\mathbf{K}^{DW}[:, :, m] \in \mathbb{R}^{k \times k}$ is learned, reducing the number of parameters for the entire kernel to $1 \cdot d \cdot k^2$. However, this type of convolution has two limitations: the number of feature maps cannot be changed [3] and there is no information exchange between channels. These disadvantages can be alleviated by employing a subsequent 1×1 convolution. Taken together, the number of used parameters is:

$$\#\text{Parameters}(\mathbf{K}^{DW}) + \#\text{Parameters}(\mathbf{K}^{1\times1}) = d \cdot k^2 + d \cdot d_{out} = d \cdot (d_{out} + k^2); \tag{2.4}$$

which can be far smaller than the number of parameters needed by a standard convolution – especially for large k values. Note that if no non-linearities are applied to the DW convolution output, both the DW convolution and 1×1 convolution could be reformulated as a single standard convolution. DW convolutions combined with 1×1 convolutions are the primary convolution types in several novel state-of-the-art (SOTA) CNN architectures, such as the MobileNet-V2 [San+18a] or the EfficientNet [TL19a]. Similarly, the main building block in this work – the E-block – is a combination of DW convolutions and 1×1 convolutions (see Chapter 3).

2.2.3 Matrix multiplication example

This subsection shows how weight matrices differ depending on which layer type is used: dense layer, convolution, and DW convolution. We start with a dense layer and consider a function f that linearly transforms an input Tensor $\mathbf{T}_{in} \in \mathbb{R}^{2\times2\times2}$ into $\mathbf{T}_{out} \in \mathbb{R}^{2\times1\times2}$

$$f(\mathbf{T}_{in}) = f\left(\begin{bmatrix} x_0^0 & x_1^0 \\ x_2^0 & x_3^0 \end{bmatrix}, \begin{bmatrix} x_0^1 & x_1^1 \\ x_2^1 & x_3^1 \end{bmatrix}\right) = \left(\begin{bmatrix} z_0^0 \\ z_1^0 \end{bmatrix}, \begin{bmatrix} z_0^1 \\ z_1^1 \end{bmatrix}\right) = \mathbf{T}_{out}. \tag{2.5}$$

[3] It is possible to create multiple outputs feature maps of each incoming feature map (i.e., $d_{out} = 2 \cdot d, 3 \cdot d, ...$). However, in this work, we only use one output feature map per input feature map.

MLPs and convolutions can be described as matrix multiplications. In this context, the input and output tensors need to be vectorized:

$$vect(\mathbf{T}_{in}) = (x_0^0, x_1^0, x_2^0, x_3^0, x_0^1, x_1^1, x_2^1, x_3^1)^T = (v_0, v_1, ..., v_7)^T = \mathbf{v} \tag{2.6}$$

$$vect(\mathbf{T}_{out}) = (z_0^0, z_1^0, z_0^1, z_1^1)^T = (y_0, y_1, y_2, y_3)^T = \mathbf{y}. \tag{2.7}$$

In this example, to vectorize an input, each feature map is first transformed into a vector by concatenating each row from top to bottom. Then, all transformed feature map vectors are concatenated. For a dense layer, the main building block of an MLP, all outputs are connected to all inputs, and the weight matrix $\mathbf{W}_0 \in \mathbb{R}^{4\times 8}$ can have up to 32 unique non-zero entries.

$$\begin{bmatrix} w_0^0 & ... & w_0^7 \\ w_1^0 & ... & w_1^7 \\ w_2^0 & ... & w_2^7 \\ w_3^0 & ... & w_3^7 \end{bmatrix} \mathbf{v} = \mathbf{W}_0\mathbf{v} = \mathbf{y}_0. \tag{2.8}$$

w_i^j denotes the weight value belonging to the j-th entry of $\mathbf{v}$ and the i-th entry of the output vector $\mathbf{y}_0$ (the subscript '0' refers to the unique output vector $\mathbf{y}_0$ that is created when using the weight matrix $\mathbf{W}_0$).

Consider a locality constraint with a local window of size $1 \times 2 \times 2$, i.e., only the rows of each feature map of $\mathbf{T}_{in}$ are used to compute a row of $\mathbf{T}_{out}$ (z_i^0 and z_i^1). The number of non-zero entries of $\mathbf{W}_1$ is reduced to at most $2 \cdot 1 \cdot 2 = 4$ entries per row (16 entries in total):

$$\begin{bmatrix} w_0^0 & w_0^1 & 0 & 0 & w_0^4 & w_0^5 & 0 & 0 \\ 0 & 0 & w_1^2 & w_1^3 & 0 & 0 & w_1^6 & w_1^7 \\ w_2^0 & w_2^1 & 0 & 0 & w_2^4 & w_2^5 & 0 & 0 \\ 0 & 0 & w_3^2 & w_3^3 & 0 & 0 & w_3^6 & w_3^7 \end{bmatrix} \mathbf{v} = \mathbf{W}_1\mathbf{v} = \mathbf{y}_1 \tag{2.9}$$

With weight sharing, the entries of each row that compute the values of one output feature map are equal. I.e., $w_i^j = w_{(i+1)}^{(j+2)}$ for $j \in \{0, 1, 4, 5\}$ and $i \in \{0, 2\}$. Thus, there are only 8 unique non-zero entries:

$$\begin{bmatrix} w_0^0 & w_0^1 & 0 & 0 & w_0^4 & w_0^5 & 0 & 0 \\ 0 & 0 & w_0^0 & w_0^1 & 0 & 0 & w_0^4 & w_0^5 \\ w_2^0 & w_2^1 & 0 & 0 & w_2^4 & w_2^5 & 0 & 0 \\ 0 & 0 & w_2^0 & w_2^1 & 0 & 0 & w_2^4 & w_2^5 \end{bmatrix} \mathbf{v} = \mathbf{W}_2\mathbf{v} = \mathbf{y}_2. \tag{2.10}$$

When using a DW convolution, we make an additional constraint that each feature map has its own 1×2 kernel and is processed independently from other feature maps:

$$\begin{bmatrix} w_0^0 & w_0^1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_0^0 & w_0^1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_2^4 & w_2^5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & w_2^4 & w_2^5 \end{bmatrix} \mathbf{v} = \mathbf{W}_3\mathbf{v} = \mathbf{y}^0. \tag{2.11}$$

With this configuration, there is no communication between feature maps, as each channel is processed independently. To ensure information flow, a 1×1 convolution with a weight matrix $\mathbf{M}$ is subsequently applied. This operation is, in its essence, a linear combination of the output feature maps of the intermediate result $\mathbf{y}^0$.

$$\begin{bmatrix} m_0 & 0 & m_1 & 0 \\ 0 & m_0 & 0 & m_1 \\ m_2 & 0 & m_3 & 0 \\ 0 & m_2 & 0 & m_3 \end{bmatrix} \mathbf{y}^0 = \mathbf{M}\mathbf{y}^0 = \mathbf{M}\mathbf{W}_3\mathbf{v} = \mathbf{y}_3. \tag{2.12}$$

In conclusion, for a dense layer, $4 \cdot 8 = 32$ unique non-zero weights are used. The locality constraint reduces the number of unique non-zero weights to $4 \cdot 4 = 16$. With weight sharing, there are only $2 \cdot 4 = 8$ unique non-zero weights. A DW convolution uses only $1 \cdot 4 = 4$ unique non-zero weights with an additional 4 weights for cross-feature map information flow. Although there is no parameter reduction when using a DW convolution with subsequent 1×1 convolution in this particular example, this combination can drastically reduce the number of parameters with larger kernel sizes and higher in- and output dimensions.

Of course, with a decreasing number of unique learnable parameters, the capacity of a single linear model decreases. However, deep CNNs usually do not lack the capacity to learn very complex tasks. For example, EfficientNets [TL19a], employing DW convolutions, are used very successfully on ImageNet [Pha+21]. Thus, large sequences of parameter efficient layers can be rather beneficial for generalization.

2.3 Additional operations and block structures

2.3.1 Batch normalization and instance normalization

Ioffe and Szegedy [IS15] proposed batch normalization (BN) to make the training of CNNs faster and more stable. BN normalizes and re-scales a CNN's feature maps $\mathbf{T}^m$ of an input batch $\mathbf{T} \in \mathbb{R}^{b \times h \times w \times d}$ with their mean and standard deviation μ_m and σ_m^2, $m = 1, ..., d$. These values are summed up along the batch and spatial dimensions as shown in the equations below:

$$\mu_{\mathrm{m}} = \frac{1}{b}\sum_{\mathrm{n=1}}^{\mathrm{b}}(\frac{1}{hw}\sum_{\mathrm{i=1,j=1}}^{\mathrm{h,w}}\mathbf{T}[n,i,j,m]), \tag{2.13}$$

$$\sigma_{\mathrm{m}}^2 = \frac{1}{b}\sum_{\mathrm{n=1}}^{\mathrm{b}}(\frac{1}{hw}\sum_{\mathrm{i=1,j=1}}^{\mathrm{h,w}}(\mathbf{T}[n,i,j,m]-\mu_{\mathrm{m}})^2). \tag{2.14}$$

The output of the BN layer is $\mathbf{T}_{out}$:

$$\hat{\mathbf{T}}^m = \frac{\mathbf{T}^m - \mu_{\mathrm{m}}}{\sqrt{\sigma_{\mathrm{m}}^2+\epsilon}}, \qquad \mathbf{T}_{out}^m = \rho_m \cdot \hat{\mathbf{T}}^m + \omega_m; \tag{2.15}$$

here, $\mathbf{T}_{out}^m$ are all the b feature maps with index m of the batch. ρ_m is a scaling factor, and ω_m is an offset learned for each dimension m. ϵ is some small value (we use $\epsilon = 10^{-5}$) to prevent a division by zero.

The behavior of a BN layer is different during training and testing. During training, for each batch in an update step [4], μ_{m} and σ_{m}^2 are calculated anew. In addition, for each μ_{m} and σ_{m}^2, moving average values, meant to approximate the mean and standard deviation over the entire training dataset, are computed and stored. When testing, these averaged mean and standard deviation values are used instead of computing the normalization for each incoming batch. This way, a model can be tested on arbitrary batch sizes, even single images. Otherwise, using different batch sizes could lead to different layer outputs. E.g., consider processing a particular image ($b = 1$) with a CNN opposed to processing said image belonging to a batch of N different images ($b = N$): μ_{m} could vary drastically between $b = 1$ and $b = N$.

BN is nowadays a standard layer in almost any SOTA CNN that facilitates the convergence of deeper networks and often even improves generalization. The reasons for these improvements are not entirely understood. The original paper proposed that BN reduces the internal covariate shift referring to a change in the input distribution caused by updating the parameters of previous layers. However, Santurkar et al. [San+18b] point out that there is an insubstantial connection between decreasing the internal covariate shift and facilitating network training and improving performance. Instead, employing BN leads to a smoother optimization landscape that speeds up training and increases robustness against poorly chosen hyperparameters.

Ulyanov, Vedaldi, and Lempitsky [UVL16] proposed instance normalization (IN) for style transfer tasks. Instead of normalizing each batch instance with a mean value and a standard deviation based on the entire batch, IN normalizes each feature map of each instance individually. No additional moving average values are computed, i.e.,

[4] A batch of b samples is processed by the network; the network output is compared with the intended (labeled) output. The network weights are updated, aiming to make the network output more similar to the labeled output.

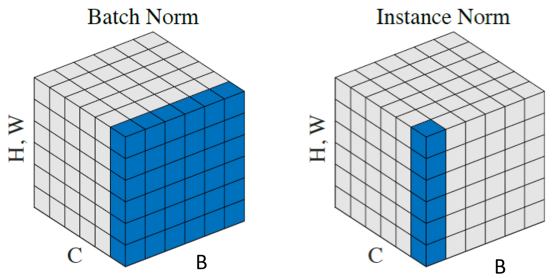


Figure 2.1: Comparing BN to IN: H and W refer to the spatial dimensions, C to the feature map dimension, and B to the batch dimension. BN calculates the mean and standard deviation for each feature map over the entire batch of three-dimensional input tensors (i.e., over B, H, and W). For later inference, validation, and testing, moving average values are computed over the entire training set. IN does not operate differently during training and testing. In both phases, IN calculates the mean and standard deviation for each feature map and for each instance over the spatial dimensions (over H and W). Figure adopted from Wu et al. [WH18].

there is no difference in behavior during training and testing, and IN does not employ trainable affine parameters ρ_m and ω_m. The two normalization strategies are compared in Figure 2.1.

2.3.2 Residual connections

A residual connection adds the input $\mathbf{x}$ of a layer f to its output $f(\mathbf{x})$:

$$\mathbf{y} = \mathbf{x} + f(\mathbf{x}). \tag{2.16}$$

These connections were first presented by He et al. [He+16] used in a new CNN architecture called residual connection networks – in short, ResNets.

At first glance, this approach appears to be another method to counter problems with vanishing gradients, which seems unnecessary since BN layers can already ensure layer outputs that are neither too small nor too large. However, before using ResNets, the network depth was a parameter that could have a detrimental effect on the training and test accuracy after exceeding a certain value. I.e., although BN ensures stable gradient values, at some depth d, a model with $d+1$ or more convolutions performed worse. This phenomenon could not be observed with ResNets: increasing the depth (up to a thousand layers) did not decrease training accuracy; however, very deep networks may generalize worse than medium-deep networks.

Residual connections are assumed to have several beneficial effects on CNNs. First, with a residual connection, a layer $f_{res}(\mathbf{x})$ rather learns a residual mapping $H(\mathbf{x})$ so that $f_{res}(\mathbf{x}) = H(\mathbf{x}) + \mathbf{x}$, $H(\mathbf{x})$ likely being easier to learn. Second, a residual connection ensures easy transport of low-level features to deeper levels of the CNN. Certain

low-level features could, in principle, help with computing more intricate high-level features.

Additionally, Veit, Wilber, and Belongie [VWB16] showed that ResNets function as ensembles of shallower networks. In general, ensembles of classifiers outperform single classifiers because, for example, blind spots of one particular classifier can be compensated by other classifiers (for more information about ensembles, see for example Dietterich [Die00]). Furthermore, new research shows [Din+21] that with the use of residual connections, older networks, such as the VGG-16 [SGS15], can be trained more efficiently to perform on par with newer models.

2.3.3 ResNet and PyrNet

Since its publication in 2016, the ResNet [He+16] has become one of the most widely adopted CNN architectures, with the paper having over 130 000 citations. Thus, ResNets make for a good baseline CNN that we aim to improve with E-blocks. In the paper, several models, most notably the ResNet-50, were proposed to approach datasets with larger images (244×244), such as ImageNet. These deeper networks, the ResNet-50, ResNet-101, and ResNet-152 (the number denotes the number of linear layers), mainly consist of so-called bottleneck blocks (see Section 2.3.6).

However, the authors also proposed shallower networks: in addition to the ResNet-18 and ResNet-34, they presented other ResNets specifically designed for Cifar-10 (e.g., the ResNet-20 or ResNet-56, which we will discuss in detail in Section 3.1). The shallower models mainly consist of so-called basic blocks. For an input $\mathbf{X} \in \mathbb{R}^{h_0 \times w_0 \times d_{in}}$ the output $\mathbf{Y} \in \mathbb{R}^{h_1 \times w_1 \times d_{out}}$ is:

$$\mathbf{Y}_{basic} = ReLU(BN(CONV(ReLU(BN(CONV(\mathbf{X})))))) + \mathbf{X}), \tag{2.17}$$

CONV is a convolution operation.

Over the years, this block structure was improved. Han, Kim, and Kim [HKK17] give an overview of different block implementations. Furthermore, they propose deep pyramidal residual networks, which employ an altered basic block:

$$\mathbf{Y}_{pyramid} = BN(CONV(ReLU(BN(CONV(BN(\mathbf{X})))))) + \mathbf{X}. \tag{2.18}$$

In this work, we denote this block *pyramid block*. In many of our experiments, we use the original Cifar-10 architecture proposed in the ResNet paper, but instead of the basic block, we use the pyramid block because these networks perform better. We call these networks *PyrNets*.

2.3.4 DenseNet

Inspired by the advantages of the ResNet, Huang et al. [Hua+17] proposed the DenseNet – focusing on feature reuse. A residual connection helps to propagate low-level features

to deeper layers of a network; however, these features are still altered by adding the value of a residual mapping. In contrast, in a DenseNet, low-level features are directly passed to deeper layers. The main building block of the architecture is called a DenseBlock containing N convolution layers with non-linearities. In this block, the input of each convolution layer $L_n, n = 1, ..., N$, is a concatenation of all previous layers $L_k, k = 1, ..., n-1$. For example, the third convolution layer input is the tensor created by concatenating all feature maps of the first layer output and the second layer output. Sequences consisting of a DenseBlock, followed by a convolution layer and a pooling layer, are connected in succession to form an entire DenseNet model.

2.3.5 Bottleneck block

Bottleneck blocks [He+16] were mainly created due to practical considerations to increase the computational efficiency of deep networks. A block consists of a 1×1 convolution that reduces the number of feature maps. The output of said convolution is processed by a 3×3 convolution. Finally, another 1×1 convolution again increases the number of feature maps. The number of parameters of a bottleneck block with input dimension d_{in}, intermediate dimension d_{hidden}, and output dimension d_{out} is: $d_{in} \cdot d_{hidden} + 9 \cdot d_{hidden}^2 + d_{hidden} \cdot d_{out}$. Given that $d_{hidden} \leq d_{in}$ and $d_{hidden} < d_{out}$, a bottleneck block can use fewer parameters than a standard convolution that uses $9 \cdot d_{in} \cdot d_{out}$ parameters. For example, the ResNet-50 (see Section 4.1.1.2), uses bottleneck blocks with $d_{in} = d_{hidden}$ and $d_{out} = 4 \cdot d_{in}$. The standard convolution has $36 \cdot d_{in}^2$ parameters, in contrast to $d_{in}^2 + 9 \cdot d_{in}^2 + 4 \cdot d_{in}^2 = 14 \cdot d_{in}^2$ parameters for a bottleneck block.

2.3.6 Mobile inverted bottleneck

The mobile inverted bottleneck was first introduced by Sandler et al. [San+18a]. These blocks are similar in structure to a standard bottleneck block: the inverted bottleneck's sequence starts with a 1×1 convolution with ReLU. Instead of a standard convolution, the next step of the sequence is a 3×3 DW convolution with ReLU, followed by a 1×1 convolution. Furthermore, instead of decreasing the input dimension d_{hidden} before the 3×3 convolution (see Subsection 2.3.5), d_{hidden} is increased. Since DW convolutions are used, inverted bottlenecks can still be more parameter efficient than standard convolutions. An expansion factor usually determines the size of d_{hidden}; e.g., for the MobileNet-V2, $d_{hidden} = 6 \cdot d_{in}$. For the E-blocks (see Chapter 3), we use $d_{hidden} = \lfloor q \cdot d_{out} \rfloor, q \in \{0.8, 1, 2\}$.

The main motivation for increasing d_{hidden} lies in reducing information loss. Given a set of real input images $\mathbb{T}$, the set of d_l-dimensional feature vectors of a layer L's output is $\{L(\mathbf{T})[i, j, :] | L(\mathbf{T}) \in \mathbb{R}^{h_l \times w_l \times d_l}; i = 1, ..., h_l; j = 1, ..., w_l; \mathbf{T} \in \mathbb{T} \subset \mathbb{R}^{h \times w \times d}\}$. This set forms a manifold that can, in principle, be embedded into a low-dimensional

subspace – justifying the use of a standard bottleneck. The manifold assumption implies that the width of a CNN can be reduced since a lower dimension can represent the manifold. However, with non-linearities like ReLUs, information can be destroyed; for example, when a ReLU collapses several values of a representation vector $\mathbf{T}[i, j, :]$ to zero. Nonetheless, if a representation vector contains enough dimensions, the information lost due to non-linearities can be compensated by other channels; i.e., the transformation containing a ReLU can be likely inverted. Consider the expression $\mathbf{y} = ReLU(\mathbf{W}\mathbf{x})$ with $\mathbf{W} \in \mathbb{R}^{m \times n}$ and $\mathbf{x} \in \mathbb{R}^n$. It can be inverted if at least n values of $\mathbf{y} \in \mathbb{R}^m$ are greater than zero. Increasing $m > n$ increases the likelihood of invertibility [San+18a].

2.4 Datasets

This work evaluates E-nets with a multitude of different datasets and tasks. However, Cifar-10 and ImageNet reoccur in almost all implementation sections as they are well-established benchmarks in image classification. Most of our ideas were first tested with smaller CNNs on Cifar-10 and – if the results were promising – subsequently with larger CNNs on ImageNet.

Our primary evaluation metric is the (test) error, ranging from 0 to 100, denoting the percentage of incorrectly classified images. Note that throughout this thesis, we do not explicitly state that the test error refers to a percentage.

2.4.1 Cifar-10

Cifar-10 [KNH10] contains 60.000 RGB images, each with a height and width of 32 pixels. The dataset was created in 2009 by Krizhevsky, Hinton, et al. [KH+09]. Each image is an instance of one of the ten classes: airplane, car, dog, frog, horse, bird, cat, deer, ship, and truck. The goal of a machine learning algorithm is to predict the correct class given an image. In the field of deep network architecture search, this dataset is quite popular. It is small enough to quickly train several networks while still conveying information about how well a deep network works on natural image recognition – see Ying et al. [Yin+19] for an example.

At the time of writing, the best approach on Cifar-10 has an error rate of 0.6% [Tou+21; ST] using an ImageNet (see Section 2.4.2) pre-trained image transformer. In our experiments, we do not employ pre-training and compare the results of baseline models trained from scratch to E-nets. Furthermore, we use the standard data augmentation approach of padding and randomly cropping the image during training and flipping the image along the vertical axis. This way, we obtain error rates of $5 - 10\%$, depending on the particular model.

2.4.2 ImageNet

The groundbreaking AlexNet paper [KSH12] is based on the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [Rus+15]. Since 2010, this challenge has been held annually, and the dataset is one of the most popular image classification benchmarks to this day. The large scale was the main novelty of the dataset consisting of over a million images of 1000 different classes. The classes span a wide array of very particular animal- and insect species (e.g., green snake, green mamba, Indian cobra, hornbill toucan, black swan, beagle, Irish wolfhound, barn spider, garden spider), objects (e.g., broom, cannon, canoe, vending machine), vehicles, and planes. There is no particular *person* class, but humans are visible in the images of different object classes (e.g., cowboy hat). Before the advent of CNNs, the ILSVRC was far from being solved; for example, the 2010 best approach had a top-5 error of 28.2%. However, in 2012, AlexNet drastically reduced the top-5 error rate to 15.3%, clearly showing that combining deep networks and many training images was a feasible approach to tackle this challenging image classification task.

At the time of writing, the top-1 [5] error of the validation dataset [6] is below 10%. A large EfficientNet [TL19a; Pha+21] (9.8%) is the best performing CNN, while vision transformers [Din+22] (9.6%) or combinations of vision transformers and CNNs [Dai+21] (9.12%) are currently among the best performing models. [7] However, it is important to note that the best results are now obtained by pre-training very deep and wide models on datasets that greatly exceed one million images – see Kolesnikov et al. [Kol+20] for an overview. For example, there is a larger ImageNet version (ImageNet-21k [Rid+21], 14 million images, 21841 classes). Furthermore, Google's internal JFT-300M dataset [Kol+20] contains 300 million noisily labeled [8] images.

Arguably, developing and researching models that scale well with abundant data is a feasible approach to improving image classification. However, in this work, we conduct our research on ImageNet without additional data and instead focus on researching inductive biases.

[5] Since 2012, the SOTA performance on ImageNet drastically improved. Thus, the main evaluation metric 'top-5 error', i.e., an image is correctly classified if the correct class is in the top five predictions, was changed to the more difficult 'top-1 error'.

[6] The validation dataset is normally used for benchmarking models since the test dataset is not publicly available.

[7] Results were taken from the Papers With Code leaderboard [ST20].

[8] The images were labeled by an algorithm, not by human annotators.

Chapter 3

What is an E-net?

E-nets are convolutional neural networks (CNNs) containing neurons that explicitly model end-stopped behavior. To create an E-net, a state-of-the-art (SOTA) CNN, such as the ResNet [He+16], MobileNet-V2 [San+18a], or DenseNet [Hua+17], is modified: for this so-called *baseline* CNN, we substitute particular sequences of convolution layers that we denote as blocks, with E-blocks. As described in the introduction, there is ample evidence that end-stopped neurons are essential for vision tasks, and standard CNNs model end-stopped behavior implicitly to some degree. However, we provide evidence that transforming a baseline CNN to an E-net, that models end-stopped neurons explicitly, can either (i) be more efficient while maintaining or improving the CNN's generalization or (ii) improve the generalization with only a slight increase in the number of model parameters.

In this chapter, we define terms to describe CNN architectures. Next, the general E-block structure is proposed with different implementations. In addition, we present theoretical results about E-nets.

3.1 Nomenclature describing CNNs

We will use the following terms to describe particular CNNs: A *layer* is the smallest building unit of a model, the most prominent example being the convolution layer. In addition, normalization layers such as batch normalization (BN) and instance normalization (IN) (for more information, see Section 2.3.1) or non-linearities (e.g., ReLUs) fall into this category. *Blocks* are distinct sequences of layers, e.g., the proposed E-block consists of several convolution layers and normalization layers and non-linearities arranged in a specific order (see Section 3.5). Analogously, *stacks* are sequences of blocks. Usually, they are separated by downsampling operations (e.g., strided convolutions or max-pooling layers), except for maybe the first stack being placed after an initial convolution layer; this convolution layer is also known as the *stem*.

Figure 3.1 shows an example of a ResNet architecture for Cifar-10 that has three stacks with five blocks each. Each first block of the second and third stacks contains a convolution layer with a stride of 2 that downsamples the input.

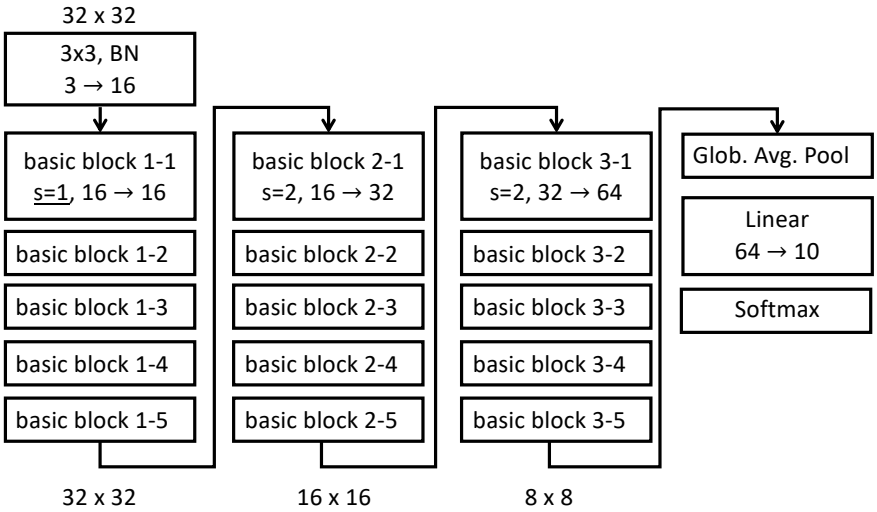


Figure 3.1: Schema of the ResNet-32 model used on the Cifar-10 dataset: the input size of the images is 32×32; $s = 2$ denotes a convolution stride of 2, $16 \rightarrow 32$ denotes a change in the number of feature maps from 16 to 32. The input is first processed by a 3×3 convolution layer with BN (stem). The subsequent operations are grouped in three stacks containing five basic blocks each. Global average pooling transforms the processed input into a 64-dimensional feature vector. Via matrix multiplication, one output score for each class is computed. These values are then transformed into a probability distribution by a softmax function.

With *architecture*, we denote a set of CNN models that are similar in particular aspects. Most notably, the E-net architecture consists of models containing at least one E-block. The ResNet, for example, contains a specific block: either a basic block for smaller ResNets or a bottleneck block for larger ResNets. Both block types contain residual connections, giving the architecture its name. Another example is the MobileNet-V2 architecture, which consists of mobile inverted bottlenecks. The ResNet and MobileNet-V2 blocks are depicted in Figure 3.2; additional information on these baseline architectures is also given in Chapter 2. Finally, note that several models can vary in size for one architecture. For example, the Cifar-10 baseline ResNets we employ in our experiments differ in the number of blocks for each stack ($N \in \{3, 5, 7, 9\}$).

In the context of CNNs, with *neuron*, we denote the part of a layer that produces a specific feature map. E.g., for a convolution layer with ReLU, a specific neuron i produces the i-th feature map. Such neurons are called linear non-linear (LN)-neurons. Similarly, an E-neuron generates a feature map by filtering the input feature map with two distinct filters and combining the filter outputs with a logical AND combination.

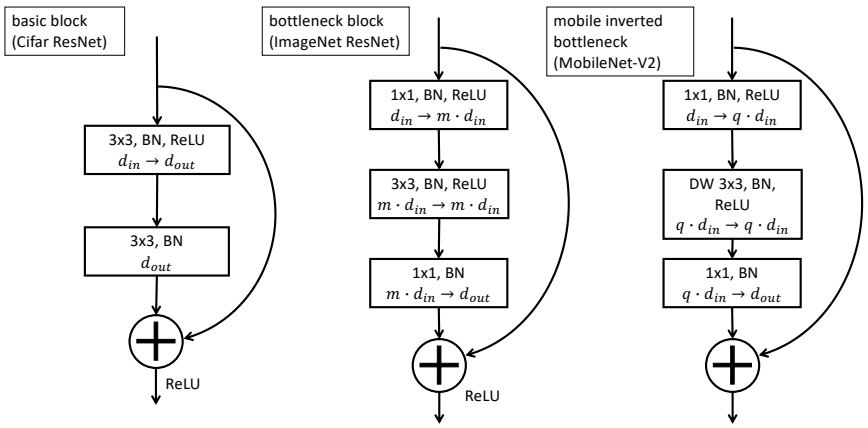


Figure 3.2: Comparison of different block architectures used in SOTA models: the left schema shows the basic block used in the ResNet-18 or ResNets operating on smaller input data (e.g., Cifar-10). The middle schema shows the bottleneck block of the larger ResNets trained on ImageNet (e.g., the ResNet-50). With m, for example, being $\frac{1}{2}$, this block reduces the number of parameters because the parameter-costly 3×3 convolution operates on fewer feature maps. The third 1×1 convolution again increases the number of feature maps to d_{out}, usually being $4 \cdot d_{in}$. The right schema shows the mobile inverted bottleneck used in the MobileNet-V2. This block strictly separates the interaction between feature maps (1×1 convolutions) and feature extraction (3×3 depthwise (DW) convolution). With usually $q = 6$, the first 1×1 convolution increases the number of feature maps. However, employing DW convolutions, this block still may use fewer parameters than the basic and bottleneck block.

3.2 E-net naming convention

Any CNN containing at least one E-block is an E-net. Since we create different E-net models from different SOTA CNNs, we use a combination of 'E-' and the name of the respective baseline model to convey that this model was used to create the E-net. For example, some sections of the following chapter present and evaluate an E-ResNet-50-I being an adaptation of the ResNet-50. This work investigates the performance of different E-blocks: e.g., the E-block-I and E-block-R, and others. Thus when referring to a particular model with a specific E-block, the name is E-[*name of baseline model*]-[*block specifier*].

3.3 Design rules: which blocks to substitute?

In biological vision, neurons are end-stopped to different degrees – thus, including neurons without end-stopped behavior. Accordingly, modeling an E-net consisting only of E-blocks is not in line with biological systems and produces E-nets with inferior

performance. Instead, one needs to predetermine which blocks of a baseline CNN to substitute with E-blocks – ideally at positions along a deep network's feature hierarchy where end-stopped cells are vital. In other parts, they may be unnecessary. For a CNN with B blocks, 2^B possible E-net models could be created. By *design rule*, we denote the procedure for choosing which blocks should be substituted in a baseline network. In most experiments, we use the design rule *substitute each first block of a stack* (see Section 4.3.1), a choice that is, to some extent, inspired by biological vision. Furthermore, we investigate exchanging entire stacks with stacks of E-blocks in Section 4.1.1.1. In Section 4.7, we systematically evaluate the effects of different design rules and propose a genetic algorithm to optimize E-nets.

3.4 Why AND combinations of filter pairs

In Chapter 1, we discussed that regions with an intrinsic dimensionality (iD) of two (2D regions) do not often occur in natural images and contain therefore a high amount of information about the image. Furthermore, one can show that knowing the 2D regions of an image is enough to reconstruct it. Differential geometry offers a way to detect 2D regions, viewing an image as a surface with pixel values indicating elevation levels. 2D regions have a non-zero Gaussian curvature. The determinant of the Hessian suffices to classify 2D regions using the Gaussian curvature:

$$DET = l_{xx}l_{yy} - l_{xy}; \tag{3.1}$$

with $l_{xy} = \frac{\partial l}{\partial x \partial y}$ being a partial derivative of the two-dimensional luminance function $l = l(x, y)$. From Equation 3.1, one can derive the essential components of a 2D detector: *a pair of measurements that is AND combined*, here via multiplication, being able to detect a change in signal in two directions. Note that there exist 1D signals that may cause non-zero outputs in both l_{xx} and l_{yy}; accordingly, l_{xy} is subtracted to remove this possible overlap (see Zetzsche, Barth, and Wegmann [ZBW93] and Zetzsche and Barth [ZB90] for more information).

The proposed E-blocks contain the essential components. We do not aim for E-nets to estimate the Gaussian curvature somewhere in the feature hierarchy explicitly; nor do we want to enforce that only 2D regions are detected and that only oriented filters are used – this goal would arguably necessitate an exact knowledge of where to place these perfect 2D detectors. Instead, with E-nets, we propose a CNN structure that can learn a 2D detector by learning the appropriate filter weights, but the structure is not forced to do so. Hence, the proposed E-block is only loosely based on Equation 3.1. However, as Figure 4.16 on page 77 shows, proper 2D detectors are indeed learned by E-nets.

3.5 General E-block structure

A block as a part of a CNN is a sequence of layers and operations that transforms an input tensor $\mathbf{T}_0 \in \mathbb{R}^{h \times w \times d_{in}}$ to an output tensor $\mathbf{T}_{out} \in \mathbb{R}^{\frac{h}{s} \times \frac{w}{s} \times d_{out}}$. A tensor consists of a number (e.g., d_{in}, d_{out}) of feature maps, each with spatial width w and height h that may be altered by a factor s. The typical input tensor for a CNN is an image, the three color channels being the feature maps. The E-block structure resembles the mobile inverted bottleneck (see Figure 3.2), with the important extension that it contains two DW convolution layers instead of one and that the results of those two layers are combined by an AND operation.

Figure 3.3 shows a schema of the general E-block; it shows that the block consists of (i) a convolution layer with kernel size one, BN, and ReLU, altering the number of feature maps, (ii) two DW convolutions, each learning one filter per feature map with optional non-linearities ϕ, the outputs of which are element-wise combined by a function ζ, (iii) a second 1×1 convolution with BN and ReLU, recombining the outputs of (ii) to the desired number of feature maps and, finally (iv) an optional residual connection. The following paragraphs offer a more detailed description.

First, $qd_{out} \in \mathbb{N}$ feature maps are computed as weighted sums of the d_{in} feature maps of the input tensor $\mathbf{T}_0 \in \mathbb{R}^{h \times w \times d_{in}}$. $q \in \mathbb{R}^+$ is an expansion factor, and d_{out} is the desired number of output feature maps. Thus, a convolution layer with kernel size one, and subsequent BN and ReLU, computes the (i, j)-th pixel of the m-th feature map as:

$$\mathbf{T}_1[i, j, m] = ReLU(BN(\sum_{n=1}^{d_{in}} w_m^n \mathbf{T}_0[i, j, n])); m = 1, ..., qd_{out}; \tag{3.2}$$

$w_m^n \in \mathbb{R}$ are learned weights.

In the following step, pairs of DW convolution filter outputs are AND combined. As described in Section 2.2.2, a DW convolution learns one $k \times k$ filter $\mathbf{W}^m \in \mathbb{R}^{k \times k}$ per feature map m (see Equation 2.3). We use DW convolutions because of two considerations: (i) compared to a standard convolution, the number of operations and parameters is reduced. (ii) a typical end-stopped cell is modeled with the combination of two oriented filters. Using a standard convolution would implicate filters that have additional OR combinations that may not be necessary. [1]

To better analyze the behavior of the AND combination, a good approach is to view the convolution operation as an ordered sequence of scalar products. Let $\mathbf{V}^m$ and $\mathbf{G}^m \in \mathbb{R}^{k \times k}$ be the DW filter weights for the m-th feature map. For each pixel $\mathbf{T}_1[i, j, m]$, a patch $\mathbf{X} \in \mathbb{R}^{k \times k}$ is extracted with (i, j) defining the center of the patch. The scalar product of the vectorized patch $\mathbf{x} = vect(\mathbf{X}) \in \mathbb{R}^{k^2}$ and the vectorized

[1] One can create a factorized version of a standard convolution by first applying a DW convolution layer and then a convolution layer with kernel size one, the second convolution being the OR combination.

filter $\mathbf{v} = vect(\mathbf{V}_m) \in \mathbb{R}^{k^2}$ is computed (see Figure 3.4), returning a correlation value between the filter and each patch. [2] In this work, several AND operations, denoted as ζ, and non-linearities and normalization layers (contained in the function ϕ) are investigated. ζ and ϕ are key components for computing the output of the Tensor $\mathbf{T}_2$:

$$\mathbf{T}_2[i, j, m] = \zeta(\phi(\mathbf{x}^T\mathbf{v}), \phi(\mathbf{x}^T\mathbf{g})). \tag{3.3}$$

$\mathbf{T}_2 \in \mathbb{R}^{\frac{w}{s} \times \frac{h}{s} \times qd_{out}}$ is the resulting tensor and s the stride of the filter operation. If s is greater than one, $\mathbf{T}_2$'s width and height are subsampled. Finally, a second linear recombination with ReLU and BN (see Equation 3.2) creates the tensor $\mathbf{T}_3 \in \mathbb{R}^{\frac{h}{s} \times \frac{w}{s} \times d_{out}}$. Some E-blocks use residual connections:

$$\mathbf{T}_{out} = \mathbf{T}_0 + \mathbf{T}_3. \tag{3.4}$$

Apart from the logical AND combination, other design decisions must be made to create an E-block. Thus, this work proposes different E-nets, using different E-blocks, denoted by a suffix X: E-block-X. Most implementations were developed having specific research questions in mind. Some of our questions were:

- Can an explicit implementation of end-stopping yield more efficient E-nets while maintaining the performance of the baseline CNNs (E-net-I)?
- Can we create networks that generalize better than their baselines (E-net-R)?
- The obvious choice to model end-stopping is using multiplication; are there alternatives (E-net-M)?
- Are end-stopped neurons more robust against noise (E-net-R and E-net-M)?
- Can we use the logarithm to model larger products efficiently (E-net-L)?

Having different goals yielded different choices of non-linearities (ζ, ϕ) – sometimes even different design rules. Table 3.1 gives an overview of the E-blocks. The remainder of this chapter briefly introduces each block and presents theoretical considerations. Further details are provided in the next chapter.

3.6 E-nets-I

The E-block-I is directly inspired by the Gaussian curvature. It contains two filters, $\mathbf{V}$ and $\mathbf{G}$, learned when training the E-net. The block's AND combination is a multiplication with subsequent BN (ζ). No additional non-linearities are applied to the filter outputs; i.e., ϕ is the identity function. The multiplication yields a non-zero

[2] From a strictly mathematical perspective, convolution layers in CNNs do not compute a convolution, but a cross-correlation.

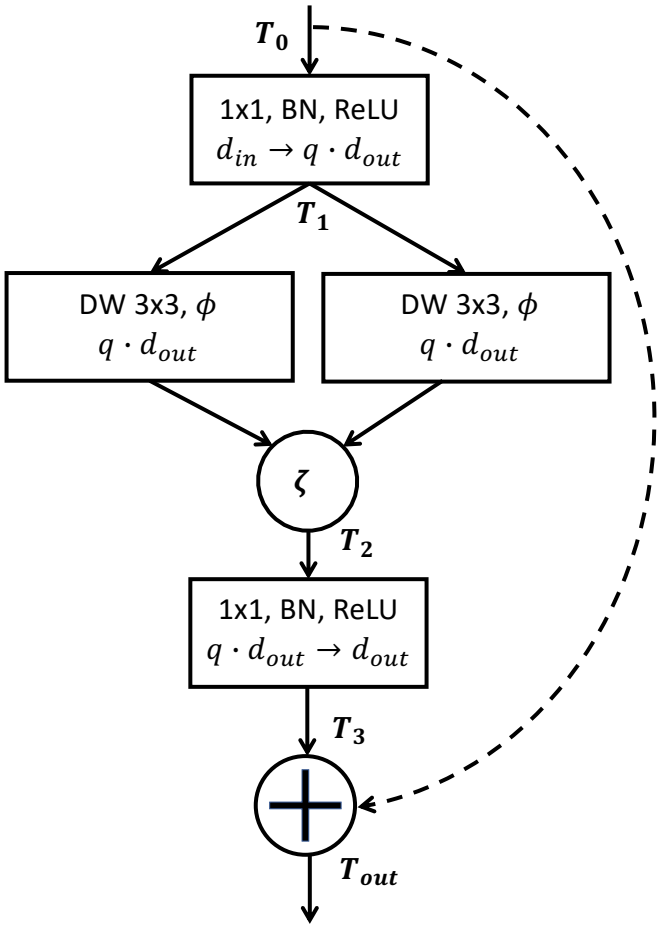


Figure 3.3: The structure of an E-block is illustrated with rectangles and circles for the various operations applied to the input tensor $\mathbf{T}_0$ gradually transforming it into $\mathbf{T}_{out}$. The first row within each rectangle denotes which operations are applied in sequence. In the second row, the number of feature maps is given and $d_{in} \rightarrow qd_{out}$ indicates that the input number of feature maps d_{in} changes to qd_{out}. The arrows in the figure indicate the inputs to the different operations and are labeled with the tensors defined in the equations (see text). Note that $\mathbf{T}_1$ is input to two different DW 3×3 convolutions (middle rectangles) that are learned. Convolutions are followed by the operation ϕ, which may include non-linearities and normalization, resulting in two different tensors. $\mathbf{T}_2$ is the result of the element-wise combination (ζ) of these two tensors (see Equation 3.3). A second linear combination, depicted by the bottom rectangle, yields $\mathbf{T}_3$. In some E-blocks, to compute the output $\mathbf{T}_{out}$ a residual connection adds the input tensor $\mathbf{T}_0$ to $\mathbf{T}_3$ (dashed arrow).

output if there is a positive or negative correlation between the input patch and each

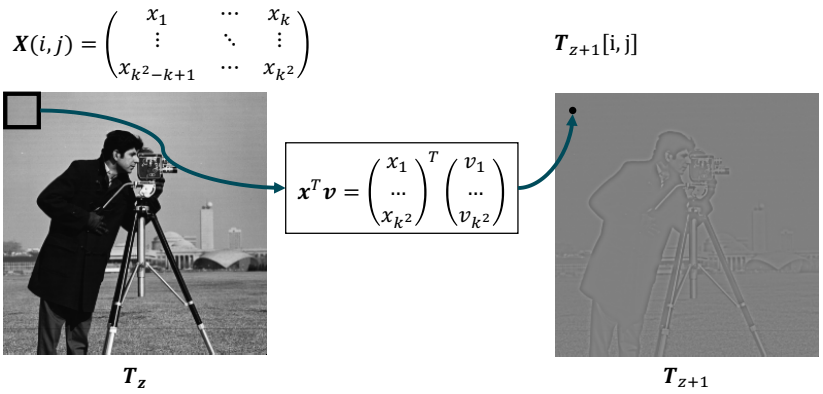


Figure 3.4: A DW convolution operation can be regarded as an ordered sequence of scalar products. For each pixel pair (i, j) of some input feature map $\mathbf{T}_z$ (left image), an image patch $\mathbf{X} \in \mathbb{R}^{k \times k}$ (black square in the left image) is extracted with (i, j) being the center pixel. To obtain the output of the convolution at (i, j) (the black dot in the right image), one computes the scalar product of the vectorized patch $\mathbf{x} = vect(\mathbf{X}) \in \mathbb{R}^{k^2}$ with the vectorized filter $\mathbf{v}$.

of the two filters. The output is a weighted sum of k^4 possible pixel pairs, similar to a Volterra kernel [Vol59]:

$$\mathbf{T}_2'[i, j, m] = \mathbf{x}^T \mathbf{v} \mathbf{g}^T \mathbf{x} = \mathbf{x}^T \mathbf{M} \mathbf{x} = \sum_{i,j}^{k^2} \mathbf{M}[i, j] x_i x_j. \tag{3.5}$$

$\mathbf{M}$ is a factorized matrix as it is an outer product of $\mathbf{v}$ and $\mathbf{g}$. Thus, the k^4 entries of $\mathbf{M}$ stem from only $2k^2$ filter values. Note that, for the E-block-I, each feature map $\mathbf{T}_2^m$ is subsequently normalized with a non-affine BN:

$$\mathbf{T}_2[i, j, m] = \frac{\mathbf{T}_2'[i, j, m] - \mu_m}{\sigma_m}. \tag{3.6}$$

I.e., the m-th feature map is subtracted with an average activation value μ_m and divided by the standard deviation σ_m; both values are estimated using the training set (see Section 2.3.1). The E-block-I does not use strided convolutions; we instead downsample the output of the E-block with a max-pooling layer with kernel-size 2 and stride 2. Furthermore, no residual connections are used.

In the literature, many works use pairwise interactions (in our case, AND combinations of filter pairs) with neural networks (see Section 1.4). Most notably, there are factorized bilinear (FB)-CNNs. However, as our inspiration comes from biology, essential differences can be observed that are discussed in the following.

Table 3.1: Overview of different E-blocks: the main operation of each E-block is the AND combination (ζ) of two DW convolution layer outputs that are first processed by ϕ. For more information, see Equation 3.3. Some of these blocks use residual connections. Note that the E-net-L models we investigate in Section 4.6 do not precisely fit the general E-block structure. However, creating an E-net-L using the general E-block structure is straightforward.

Block	ϕ	ζ	res. Connection?
E-block-I	Identity	Multiplication and BN	No
E-block-R	IN and ReLU	Multiplication	Yes
E-block-M	IN and ReLU	Minimum	Yes
E-block-L	BN and ReLU	E-layer-L (logarithm)	No

3.6.1 E-blocks-I vs. factorized bilinear CNN layers

Using multiplications with an artificial neural network (ANN) has been studied extensively with sigma-pi networks [MK90]. However, only a few works transferred AND combinations to CNNs. The presented E-block is similar to the class of bilinear CNNs [Gao+16]. Architectures of this type employ pairwise interactions of features to increase the capacity of a layer instead of simply increasing the number of layers. This idea can be viewed as a kernel approach: incoming data are implicitly mapped to a higher dimensional input space where a separation or transformation of the data is easier [Li+17b]. One particular architecture, FB-models [Li+17b], improve the parameter efficiency of bilinear CNNs, because they use FB-layers that constrain the number of parameters used to model the pairwise interactions. As there is a resemblance between the FB-layer and the E-block-I, it is sensible to take a closer look. Indeed, to some degree, the E-block can be seen as a specialized version of the FB-layer, with the E-block-I reducing the number of parameters spent on pairwise interactions even further (see Equation 3.5). However, configurations can be modeled with an E-block-I, which are impossible to replicate by an FB-layer.

In an FB-layer, the output of each neuron depends on a linear part (not shown here) and a weighted combination of pairwise terms across feature maps. Let $\mathbf{x} \in \mathbb{R}^n$ be the vectorization of the d-dimensional input patch $\mathbf{X} \in \mathbb{R}^{k \times k \times d}$, and $n = k^2 d$. The quadratic part of an output neuron is computed by a scalar product of a vector $\mathbf{w}_0 \in \mathbb{R}^{n^2}$ and the vectorized version of the outer product of $\mathbf{x}$ with itself:

$$y = \mathbf{w}_0^T vec(\mathbf{x}\mathbf{x}^T) = \mathbf{x}^T \mathbf{W} \mathbf{x}; \tag{3.7}$$

$\mathbf{W} \in \mathbb{R}^{n \times n}$ is a reshaped version of $\mathbf{w}_0$ that can be expressed by a factorized matrix $\mathbf{U} \in \mathbb{R}^{c \times n}$:

$$y = \mathbf{x}^T \mathbf{U}^T \mathbf{U} \mathbf{x} = \sum_{i=1}^{k^2} \sum_{j=1}^{k^2} (\mathbf{u}_i^T \mathbf{u}_j) x^i x^j, \tag{3.8}$$

$$y = \sum_{i=1}^{n} \sum_{j=1}^{n} w^{ij} x^i x^j; \tag{3.9}$$

with $\mathbf{u}_i$ being the i-th column of $\mathbf{U}$. On a first glance, Equation 3.7 appears to be a general case of Equation 3.5: FB-layers compute a weighted sum across **all** feature maps and sum over $k^4 d^2$ values. In comparison, E-blocks reduce the number of neuron interactions because they process multiplicative terms **for each** feature map separately. However, certain weight matrices $\mathbf{M}$ of E-blocks – combining two different filters – cannot be reproduced by factorized matrices $\mathbf{W}$. For example, $\mathbf{v} = (1,\ 0)^T$ and $\mathbf{g} = (0,\ 1)^T$ yield the symmetric 2×2 matrix $\mathbf{M} = \frac{1}{2}(\mathbf{v}\mathbf{g}^T + \mathbf{g}\mathbf{v}^T)$ that has zero entries on the main diagonal and $\frac{1}{2}$ on the top right and bottom left positions. No combination of $\mathbf{U}^T\mathbf{U}$ can create such a matrix. [3]

Apart from mathematical differences, E-blocks and FB-layers are also employed at different positions along a CNN's feature hierarchy: Li et al. [Li+17b] state that FB-layers in shallower positions are not beneficial – which is not the case with E-blocks that are often more beneficial in shallower positions (see Section 4.7). This indicates that the pairwise interactions that model AND connections of simple cells (E-blocks) differ significantly from the approach of introducing a bilinear kernel into convolution layers (FB-layers).

3.7 E-nets-R

Considering the E-block-R, the function ϕ (handling both filter outputs independently) uses IN and a ReLU. The AND combination ζ is a multiplication with no additional normalization. Thus, compared to the E-block-I, we move the normalization step before the AND combination and use IN instead of BN. Thus, Equation 3.3 is adapted as follows:

$$\mathbf{T}_2[i,j,m] = \frac{1}{\sigma_v \sigma_g} ReLU(\mathbf{x}^T\mathbf{v} - \mu_v) ReLU(\mathbf{g}^T\mathbf{x} - \mu_g). \tag{3.10}$$

μ and σ are the mean value and standard deviation of $\mathbf{T}_1^m$ after convolution with either $\mathbf{V}^m$ or $\mathbf{G}^m$:

$$\mu_v = \frac{s^2}{hw} \sum_{i,j} (\mathbf{T}_1^m * \mathbf{V}^m)[i,j], \tag{3.11}$$

$$\sigma_v = \frac{s^2}{hw} \sum_{i,j} (\mathbf{T}_1^m * \mathbf{V}^m - \mu_v)^2[i,j]; \tag{3.12}$$

with $(\mathbf{T}_1^m * \mathbf{V})[i,j]$ being the (i,j)-th pixel value of the filter result.

[3] Since the main diagonal of $\mathbf{M}$ is all-zero, $\mathbf{u}_1^T\mathbf{u}_1 = 0$ and $\mathbf{u}_2^T\mathbf{u}_2 = 0$ (see Equation 3.8). Thus, $\mathbf{u}_1 = \mathbf{u}_2 = \mathbf{0}$. Accordingly, $\mathbf{u}_1^T\mathbf{u}_2 = 0 \neq \frac{1}{2}$.

Mainly, both changes aim to reduce problems from the multiplication's exploding or vanishing values by reducing the variance of the input signal to the multiplication. In general, BN can be challenging to employ in CNNs if the BN inputs contain outliers.

During training, each feature map $\mathbf{T}[b, :, :, m]$ belonging to a sample b of a batch consisting of B Tensors is normalized by the mean μ_{batch} and divided by the standard deviation σ_{batch} computed over all $\mathbf{T}[i, :, :, m]$; $i = 1, ..., B$. In addition, the values μ_{BN} and σ_{BN} are calculated via a moving average operation. E.g., after each processed batch, μ_{BN} is updated like this: $\mu_{BN} \leftarrow \alpha\mu_{batch} + (1 - \alpha)\mu_{BN}$. μ_{BN} and σ_{BN} are used for normalization during inference so that the trained CNN can later be used on batches with sizes that differ from the training batch size.

However, the moving average is susceptible to outliers, and thus, the estimation of μ_{BN} and σ_{BN} is more difficult when the normalization layer is placed behind a multiplication. Accordingly, moving the normalization step before the multiplication would lead to smoother averaging. However, normalizing each instance removes the need for a moving average altogether: $\mathbf{T}_b^m$ is subtracted with the mean (divided by the standard deviation) of $\mathbf{T}[b, :, :, m]$ instead of the mean (standard deviation) over all $\mathbf{T}[i, :, :, m]$; $i = 1, ..., B$. In previous experiments, we observed a slight increase in performance when exchanging IN with BN. For more information about the two different normalization approaches, see Section 2.3.1.

Incorporating the ReLU aims to increase the selectivity of the AND combination. This approach is more similar to a logical AND operation as each filter with a ReLU only responds to positive correlations of the filter vector and the input patch. Thus, the AND combination only yields a high activation if both filters $\mathbf{v}$ AND $\mathbf{g}$ have a positive activation. Furthermore, applying a ReLU after IN, i.e., to a signal with zero mean, usually ensures that not many values are positive altogether, [4] increasing the sparseness of the output signal. Thus, an E-neuron-R may react to very specific stimuli.

3.7.1 Hyperselectivity of E-neurons

One way to investigate the behavior of a specific neuron is to find its *optimal stimulus*, i.e., the input pattern that yields the highest activation of a neuron. For example, simple cells yield a high output if presented with a visual stimulus of an edge or line with the right orientation. Additionally, it is interesting to investigate how specific a neuron is tuned to its optimal stimulus.

Vilankar and Field [VF17] used the term *hyperselectivity* to quantify this attribute, i.e., how quickly a neuron's response drops when the incoming stimulus is perturbed and differs more and more from the optimal stimulus. In deep learning, hyperselectivity

[4] Assuming a symmetric value distribution, with zero mean and a standard deviation of one, roughly half of the values of a feature map are equal or below zero; after applying ReLU, all these values are zero.

is relevant because it can increase robustness, e.g., against adversarial attacks [Pai+20]. In the following paragraphs, we show that E-blocks tend to be hyperselective.

One way to quantify hyperselectivity is to measure the *curvature* of *iso-response contours*. Given an n-dimensional input to a function f, an $(n-1)$-dimensional surface may exist such that for all points $\mathbf{s}$ on the surface, the output $f(\mathbf{s})$ is a constant. As n can be a high dimension, two-dimensional projections are used to analyze such iso-surfaces, which in two dimensions become iso-response contours $\mathbf{s} = \psi(t), t \in \mathbb{R}$.

The typical LN model neuron used in CNNs is a function $f_{LN}(\mathbf{x})$ that involves a linear projection on a weight vector $\mathbf{w} \in \mathbb{R}^n$ followed by a point-wise non-linearity $\rho(x)$. To analyze the iso-response contour of such a neuron, one first projects the input on $\mathbf{w}$, the axis corresponding to the optimal stimulus $\mathbf{x}_{opt}$. To find a second axis, one searches for a vector orthogonal to $\mathbf{x}_{opt}$, for example, by picking n random values and using the Gram-Schmidt process (see Equation 3.17) to transform the random vector to one that is orthogonal to $\mathbf{x_{opt}}$. When looking at the output of an LN-neuron for $\mathbf{x_{opt}}$ perturbed by any orthogonal vector $\mathbf{z}$ with $\mathbf{x_{opt}}^T\mathbf{z} = \mathbf{w}^T\mathbf{z} = 0$, the iso-response contour is always a straight line parallel to $\mathbf{z}$, because $f_{LN}(\mathbf{x}_{opt} + \mathbf{z}) = \rho(\mathbf{w}^T(\mathbf{x}_{opt} + \mathbf{z})) = \rho(\mathbf{w}^T\mathbf{x}_{opt}) = f_{LN}(\mathbf{x}_{opt})$. Thus, for LN-neurons, the iso-response contours have zero curvature.

For hyperselective neurons ($f_{HS}(\mathbf{x})$), there exist vectors $\mathbf{z}$ that are orthogonal to $\mathbf{x}_{opt}$ and decrease the neuron's optimal response such that $f_{HS}(\mathbf{x}_{opt}+\mathbf{z}) < f_{HS}(\mathbf{x}_{opt})$. In this case, the *exo-origin iso-response contour* bends away from the origin of the basis defined by $\mathbf{x}_{opt}$ and $\mathbf{z}$. A higher curvature of this bend indicates a more significant activation drop-off in regions different from the optimal stimulus, i.e., a greater hyperselectivity. One way to quantify the curvature is to use the coefficient of the quadratic term obtained by fitting a second-order polynomial to the iso-response contour.

We derive the analytical expression for the iso-response contours of E-neurons-R. We follow a geometric approach in order to show explicitly how the exo-origin curvature depends on the angle $\gamma = \measuredangle(\mathbf{v}, \mathbf{g})$. An alternative approach would be to work with the eigenvector of the symmetric matrix $\frac{1}{2}(\mathbf{v}\mathbf{g}^T + \mathbf{g}\mathbf{v}^T)$.

Note that all E-neurons presented in this chapter are hyperselective (if $\gamma \neq 0$) because all of them AND combine at least two filters – regardless of whether by multiplication or by minimum operation.

Prerequisites: In the two-dimensional subspace defined by $\mathbf{v}$ and $\mathbf{g}$, and for a specific constant $z \in \mathbb{R}^+$, we can derive the coordinates of the iso-response contours analytically by using a simplified version of Equation 3.10:

$$F(\mathbf{x}) = \mathbf{x}^T\mathbf{v}\mathbf{g}^T\mathbf{x}. \tag{3.13}$$

$F(\mathbf{x})$ is the output of the E-neuron-R, i.e., the product of the two linear filters $\mathbf{v}$ and $\mathbf{g} \in \mathbb{R}^n, n = k^2$. For simplicity, we disregard the instance normalization. Thus, we assume that the mean values are zero ($\mu_v = \mu_g = 0$) and the standard deviations

are one ($\sigma_v = \sigma_g = 1$), which are the two variables used for instance normalization. Furthermore, we constrain the input space of $\mathbf{x}$ to $\mathbb{S} = \{\mathbf{x} \in \mathbb{R}^{k^2} : \mathbf{x}^T\mathbf{v} \geq 0 \wedge \mathbf{x}^T\mathbf{g} \geq 0\}$ to account for the ReLU non-linearities. In addition, we restrict γ to $[0, \pi)$ since for $\gamma = \pi$, both vectors point in opposite directions, and for any point $\mathbf{x}$, one scalar product is always negative.

Optimal stimulus: The optimal stimulus of $F(\mathbf{x})$ is not parallel to one of the filters but points in the direction of the bisector of γ. This property becomes more obvious when rewriting $F(\mathbf{x})$ as a function depending on $\alpha = \measuredangle(\mathbf{v}, \mathbf{x})$ and $\beta = \measuredangle(\mathbf{g}, \mathbf{x})$:

$$F(\alpha, \beta, \mathbf{x}) = cos(\alpha)cos(\beta)\|\mathbf{v}\|\|\mathbf{g}\|\|\mathbf{x}\|^2. \tag{3.14}$$

To simplify this particular equation, we assume $\|\mathbf{x}\| = 1$ and disregard the vector lengths $\|\mathbf{v}\|$ and $\|\mathbf{g}\|$ since the arguments α and β for the maximum of F do not depend on vector length. With $\alpha + \beta = \gamma$ we obtain:

$$F(\alpha, \beta) = cos(\alpha)cos(\gamma - \alpha) = \frac{1}{2}(cos(2\alpha - \gamma) + cos(\gamma)). \tag{3.15}$$

Note that for $\alpha = \frac{1}{2}\gamma$, F reaches the maximum value $\frac{1+cos(\gamma)}{2}$.

Subspace with straight iso-response contours: The subspace of input vectors that do not alter the E-neuron's output is defined by:

$$F(\mathbf{x} + \mathbf{p}) = F(\mathbf{x}) \Leftrightarrow \mathbf{p}^T\mathbf{v} = \mathbf{p}^T\mathbf{g} = 0. \tag{3.16}$$

For any vector $\mathbf{p}$ orthogonal to $\mathbf{v}$ and $\mathbf{g}$, the iso-response contours are straight, as they are for LN-neurons. However, as we will show in the following, there exists an orthogonal direction $\mathbf{o}$ relative to which E-units-R exhibit curved iso-response contours and, thus, hyperselectivity.

Subspace with curved iso-response contours: It is important to note that any input vector $\mathbf{x}$ is projected to the plane defined by the vectors $\mathbf{v}$ and $\mathbf{g}$ - see Equation 3.13; any vector $\mathbf{p}$ from the subspace of Equation 3.16 is orthogonal to this plane. We can consider the function $f(\mathbf{a})$ that operates on only 2D input vectors $\mathbf{a} = (a, b)^T$, which are the projections of $\mathbf{x}$ onto the vectors $\frac{\mathbf{v}}{\|\mathbf{v}\|}$ and $\frac{\mathbf{o}}{\|\mathbf{o}\|}$, respectively. Unless $\mathbf{g}$ is parallel to $\mathbf{v}$, we can derive $\mathbf{o}$ as the direction orthogonal to $\mathbf{g}$ by using the Gram-Schmidt process:

$$\mathbf{o} = \mathbf{g} - \frac{\mathbf{v}^T\mathbf{g}}{\|\mathbf{v}\|^2}\mathbf{v}. \tag{3.17}$$

If $\mathbf{g} = \lambda\mathbf{v}, \lambda \in \mathbb{R}$, $\mathbf{o}$ is simply any vector orthogonal to $\mathbf{v}$. A point $(a, b)^T$ in the two-dimensional projection space can be injected into the original input space $\mathbb{S}$:

$$\mathbf{x}_{ab} = \frac{a}{\|\mathbf{v}\|}\mathbf{v} + \frac{b}{\|\mathbf{o}\|}\mathbf{o}. \tag{3.18}$$

$\mathbf{x}_{ab}$ denotes that the vector depends on only the position in the projection space $\mathbf{a} = (a, b)^T$. The relations between the scalar products in the input space and the scalar products in the projection space are given by:

$$\mathbf{x}_{ab}^T \mathbf{v} = \|\mathbf{v}\|(a, b)\mathbf{e}_1 = a\|\mathbf{v}\| \tag{3.19}$$

$$\mathbf{x}_{ab}^T \mathbf{o} = \|\mathbf{o}\|(a, b)\mathbf{e}_2 = b\|\mathbf{o}\| \tag{3.20}$$

$$\mathbf{x}_{ab}^T \mathbf{g} = \|\mathbf{g}\|(a, b)(cos(\gamma), sin(\gamma))^T = \|\mathbf{g}\|(acos(\gamma) + bsin(\gamma)). \tag{3.21}$$

with $\mathbf{e}_1 = (1, 0)^T$ and $\mathbf{e}_2 = (0, 1)^T$. Accordingly, the multiplication of $\mathbf{x}^T\mathbf{v}$ with $\mathbf{x}^T\mathbf{g}$ yields:

$$\mathbf{x}^T\mathbf{v}\mathbf{x}^T\mathbf{g} = (a\|\mathbf{v}\|)(acos(\gamma) + bsin(\gamma))\|\mathbf{g}\| = \begin{pmatrix} a^2 \\ ab \end{pmatrix}^T \begin{pmatrix} c_1 cos(\gamma) \\ c_1 sin(\gamma) \end{pmatrix} = f(\mathbf{a}); \tag{3.22}$$

with $c_1 = \|\mathbf{v}\|\|\mathbf{g}\|$. In the projection space, the direction vector of the optimal stimulus $\mathbf{a}_{opt}$ is given by $(cos(\frac{\gamma}{2}), sin(\frac{\gamma}{2}))^T$ (see Equation 3.15). $\mathbf{a}_{orth} = (-sin(\frac{\gamma}{2}), cos(\frac{\gamma}{2}))^T$ is orthogonal to it. We aim to find all points (x, y) such that:

$$f(x\mathbf{a}_{orth} + y\mathbf{a}_{opt}) = z; \tag{3.23}$$

with $z \in \mathbb{R}^+$. Substitution and simplification yields:

$$z = c1(y^2 cos^2(\frac{\gamma}{2}) - x^2 sin^2(\frac{\gamma}{2})). \tag{3.24}$$

For a given value x and $c = \frac{c_1}{z}$, the y position of the iso-response contour is given by:

$$y(x) = \sqrt{tan^2(\frac{\gamma}{2})x^2 + \frac{c}{cos^2(\frac{\gamma}{2})}}. \tag{3.25}$$

With this equation, we can estimate the curvature of the exo-origin bend by using the quadratic coefficient of the second-order Taylor approximation around $x = 0$ to obtain:

$$\frac{1}{2}[\frac{d^2}{dx^2}(y)](0) = \frac{tan^2(\frac{\gamma}{2})}{2\sqrt{\frac{c}{cos^2(\frac{\gamma}{2})}}}. \tag{3.26}$$

For $x = 0$, $y(0)$ is the position along the optimal stimulus, where $f(y(0)\mathbf{a}_{opt}) = z$. Keeping $y(0)$ fixed, the attenuation of f when moving in a direction orthogonal to the optimal stimulus is quadratic:

$$\Delta z = f(x\mathbf{a}_{orth} + y(0)\mathbf{a}_{opt}) - f(y(0)\mathbf{a}_{opt}) = -c1x^2 sin^2(\frac{\gamma}{2}). \tag{3.27}$$

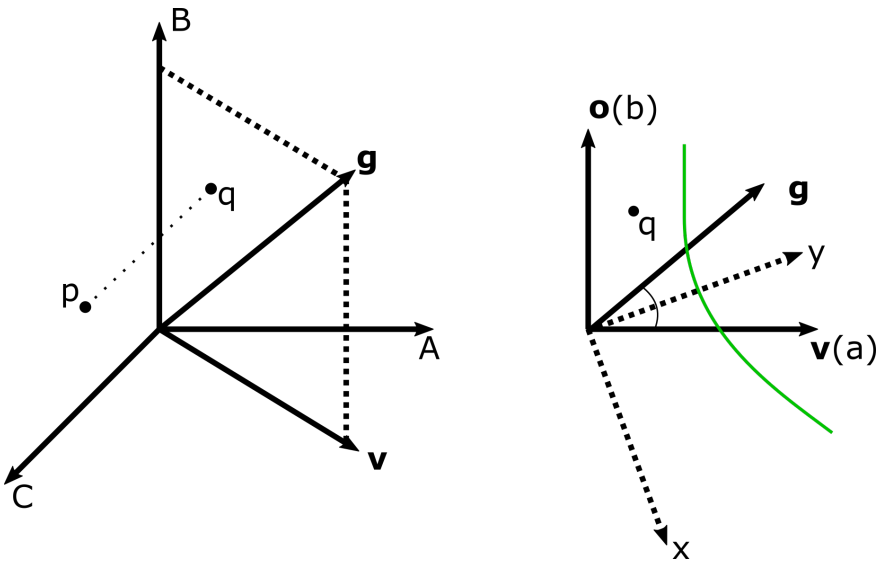


Figure 3.5: Finding the iso-response contour of an E-neuron for a two-dimensional subspace: a high-dimensional signal space is illustrated on the left side for just three dimensions, and the right side displays the corresponding two-dimensional projection. To obtain the E-neuron output for a vector $\mathbf{p} \in \mathbb{R}^3$, the scalar product of $\mathbf{p}$ and $\mathbf{v}$ is AND combined (here, multiplied) with the scalar product of $\mathbf{p}$ and $\mathbf{g}$ (see Equation 3.13). Note that the result does not change when $\mathbf{p}$ is projected to $\mathbf{q}$, i.e., projected on the plane spanned by $\mathbf{v}$ and $\mathbf{g}$. The coordinates of $\mathbf{q}$ are defined by the orthogonal axes $\mathbf{v}$ and $\mathbf{o}$. The green curve depicts the iso-response contour: any point on this curve results in the same activation z. The axes y and x span a basis with y pointing toward the optimal stimulus (see Equation 3.15). The green curve is an exo-origin iso-response contour since it bends away from the origin of x and y. Using Equation 3.25, the point $(0, y(0))^T$ is the intersection of the green curve and the optimal stimulus axis y. Equation 3.27 shows that when moving away from this point along the x-direction, the neuron's activation decreases quadratically.

Three-dimensional example: Figure 3.5 gives a three-dimensional example to illustrate how a three-dimensional point $\mathbf{p} \in \mathbb{R}^3$ can be mapped to the plane spanned by $\mathbf{v}$ and $\mathbf{g}$. The axes (a, b) of the projection space coincide with $\mathbf{v}$ and $\mathbf{o}$. Thus, there is a direct correspondence between $\mathbf{p}$ and the projected point $\mathbf{q} = (a, b)^T$ (see Equation 3.18). To estimate the curvature, we rotate the (a, b) coordinate frame clockwise by $\pi - \frac{\gamma}{2}$ to the frame (x, y). From this perspective, we can measure the change of y when moving along the x-axis and away from $x = 0$. Equation 3.25 shows that for $\gamma \in (0, \pi)$, $y(x)$ increases. Accordingly, the iso-response contour bends away from the origin of the rotated frame (x, y).

E-nets contain E-blocks that consist of E-units, or *E-neurons*, which yield the feature-product output for a pixel (i, j) in a feature map m as defined by Equation 3.10 on

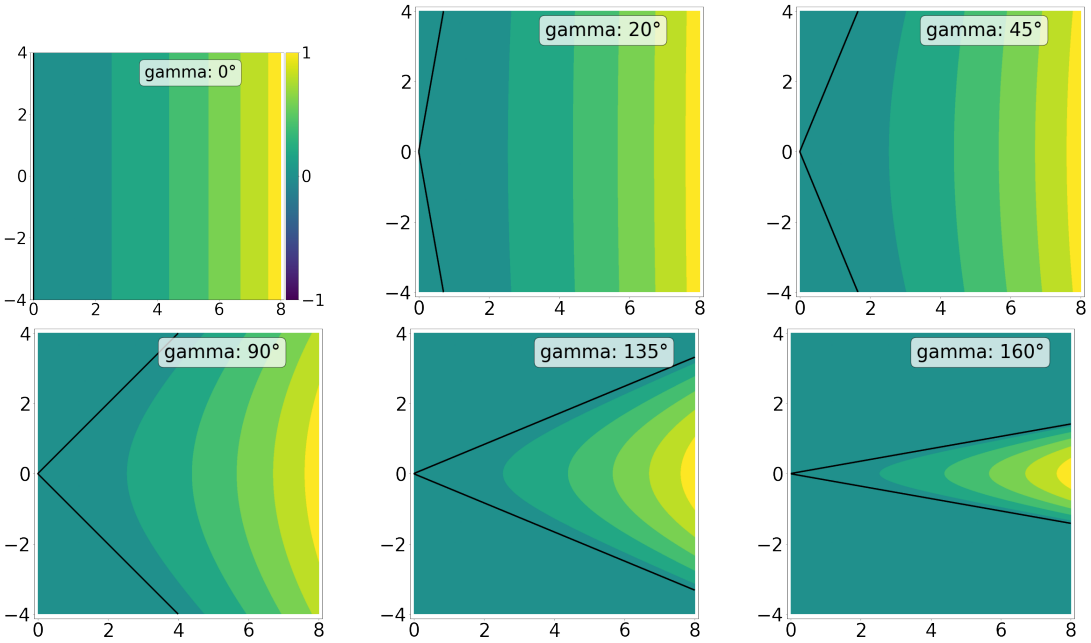


Figure 3.6: Iso-response contour plots for different values of the angle γ: each plot shows values determined by using Equation 3.23; furthermore, normalization and quantization to six bins were applied. The horizontal axis points to the direction of the optimal stimulus and is indexed by the y value in Equation 3.23. The vertical axis is orthogonal to the optimal stimulus and indexed by x. The black lines indicate the zero contour.

page 36. E-neurons exhibit curved exo-origin iso-response contours with a curvature that depends on the angle $\gamma = \measuredangle(\mathbf{v}, \mathbf{g})$. [5] Iso-response contours of E-neurons-R are shown in Figure 3.6 for different values of γ. Showing that the bend of the exo-origin iso-response contour increases with γ.

3.8 E-nets-M

The previous E-blocks used multiplication because it is the most apparent AND combination. However, as multiplications are employed in various applications in neural networks (see Section 1.4 for an overview), one may conclude that the specific multiplication is the main contributor to the observed improvements – instead of the general logical AND combination. To show that the AND combination matters – not its particular implementation via multiplication – we exchange the element-wise multiplication with the element-wise minimum operation. Regarding the AND

[5] Although our experiments only investigate E-neurons-R, this statement deliberately refers to E-neurons in general; i.e., all other proposed E-nets can have hyperselective neurons.

combination of two bits, both multiplication and minimum operation yield only a one if the two bits are non-zero; hence, both operations are valid logical AND combinations. When comparing an E-block-M to an E-block-R, only the AND combination ζ changes from multiplication to minimum (see Table 3.1 on page 35). Accordingly, the output of the AND combination $\mathbf{T}_2$ is now computed as:

$$\mathbf{T}_2[i,j,m] = \min\left[\frac{ReLU(\mathbf{x}^T\mathbf{v}-\mu_v)}{\sigma_v}, \frac{ReLU(\mathbf{g}^T\mathbf{x}-\mu_g)}{\sigma_g}\right]. \tag{3.28}$$

3.8.1 E-neurons-M vs. E-neurons-R

Similar an E-neuron-R, an E-neuron-M will have a significant output only if both filter outputs of $\mathbf{v}$ and $\mathbf{g}$ are activated. However, observing the behavior along the optimal stimulus shows that the simplified E-neuron-M can be seen as a more linear version of the simplified E-neuron-R. As discussed in Section 3.7.1, the angle of the optimal stimulus is $\alpha = \frac{1}{2}\gamma$. Thus, with $\|\mathbf{x}\|\|\mathbf{w}\|\cos(\alpha) = \mathbf{x}^T\mathbf{w}$:

$$\mathbf{x}^T\mathbf{v} \cdot \mathbf{x}^T\mathbf{g} = \|\mathbf{x}\|^2\cos(\alpha)^2\|\mathbf{v}\|\|\mathbf{g}\| \tag{3.29}$$

$$\min\left[\mathbf{x}^T\mathbf{v}, \mathbf{x}^T\mathbf{g}\right] = \|\mathbf{x}\|\cos(\alpha)\min\left[\|\mathbf{v}\|\|\mathbf{g}\|\right]. \tag{3.30}$$

I.e., moving along the optimal stimulus (increasing $\|\mathbf{x}\|$, while keeping α fixed) leads to a quadratic increase of the output of an E-neuron-R, but only to a linear increase when using an E-neuron-M. Analogously, deviating from α leads to a quadratic and linear decrease for the E-neuron-R and E-neuron-M, respectively. In general, using linear operations guarantees a more straightforward optimization, possibly reducing the risk of exploding or vanishing gradients. Another advantage is that the computation of the minimum is a bit faster than multiplication; thus, the E-block-M is a bit more time-efficient than the E-block-R (see Section B.2.1).

3.9 E-nets-L

We propose the E-block-L as an alternative to the E-block-I and E-block-R. The main idea is to substitute the explicit multiplication of filter pairs with a sum in log-space ($log(a) + log(b) = log(a \cdot b)$). This setup offers the means to efficiently compute large products of pixels of a feature map or even entire feature maps using convolution in log-space. Accordingly, more signals can be AND combined efficiently, leading, in principle, to a greater hyperselectivity. However, from a theoretical point of view, these new signals are not necessarily end-stopped anymore.

Chapter 4

Implementations and applications

In this chapter, we implement and evaluate E-nets for different applications: image classification, image quality assessment (IQA), and segmentation. This chapter's structure is based on E-nets that differ slightly in specific design decisions such as ζ, ϕ (see Section 3.5 on page 31), the design rule, and on our publications about E-nets. For an overview, see Table 4.1.

We start with E-nets-I: Section 4.1 shows the classification results of E-ResNets-I on Cifar-10 and ImageNet [GMB20a]. Subsequently, in Section 4.2, we investigate E-nets-I in the context of IQA, starting with a small IQA introduction, presenting related works, and showing the results that were first published in the Journal of Perceptual Imaging (JPI) [GB21b]. We show that with E-ResNets-I, we obtain efficient networks: for example, compared to the original ResNet-50, the E-ResNet-50-I has a similar performance on ImageNet, and on different IQA datasets while using 7 million fewer parameters and fewer layers.

Next, Section 4.3 proposes E-nets-R, which outperform their baseline architectures (ResNet, MobileNet-V2, PyrNet) in image classification (Cifar-10 and ImageNet). Furthermore, we investigate the biological properties of E-neurons-R and show that they are end-stopped. In Section 3.7.1, we already showed analytically that E-neurons are hyperselective; i.e., these neurons are tuned very specifically to an optimal stimulus. Previous works showed that hyperselective neurons could yield models that are more robust against adversarial attacks [Pai+20]. Indeed, we show in Section 4.4.3 that E-nets-R are more robust against adversarial attacks and JPEG compression artifacts. The above-mentioned results were published in the 'Deep Neural Networks and Biological Vision Special Issue' of the Journal of Vision [GMB22].

Subsequently, the following two sections cover alternative AND combinations. (i) Section 4.5.1 evaluates the E-nets-M that are a more linear version of E-nets-R because they use the minimum operation as AND combination instead of multiplication (see Section 3.8.1). Our results show that the minimum operation is a valid AND combination to model end-stopped cells, as they perform slightly better than the previously presented E-nets-R. In addition to improving the PyrNet, we show that our approach can also improve DenseNets in generalization and robustness against JPEG compression artifacts. The results were published in the 2021 Neurips Workshop 'Shared Visual Representations

Table 4.1: Overview of the topics covered in this chapter with the corresponding sections and publications presenting different implementations and applications of E-nets.

Subject	Model	Section	Publication
Classification	E-net-I	4.1	[GMB20a]
IQA	E-net-I	4.2	[GB21b]
Classification	E-net-R	4.3.1	[GMB22]
End-stopped neurons	E-net-R	4.4	[GMB22]
Robustness	E-net-R	4.4.3	[GMB22]
Classification	E-net-M	4.5.1.1	[GB21a]
Robustness	E-net-M	4.5.1.2	[GB21a]
Classification	E-net-L	4.6.2	[GMB20b]
Classification, model optimization	E-net-M	4.7	[GB23]
Segmentation	E-net-I, E-net-R, various	4.8	–

in Human & Machine Intelligence' (SVHRM) [GB21a]. (ii) In Section 4.6, we successfully train and employ E-nets-L, which are based on the logarithm's product rule. These networks can efficiently model large products of several filters, offering further exciting research opportunities with hyperselective neurons (AND combining more oriented filters could yield more selective neurons). We published the results at the 2020 International Conference on Artificial Neural Networks (ICANN) [GMB20b].

Section 4.7 gives an extensive evaluation of how to optimize E-nets-M using different design rules. For a small E-PyrNet-20-M, we evaluate all possible E-block-M positions, showing that there exist E-PyrNets-M that can outperform the baseline PyrNet-20 and even our previous models from Section 4.5.1. For larger models (e.g., the E-PyrNet-56-M), an exhaustive search is infeasible; however, we present a genetic algorithm to efficiently find well-performing E-block-M positions.

Lastly, in Section 4.8, by using E-nets, we improve two state-of-the-art (SOTA) approaches on medical image segmentation datasets: we compare E-ResNets-R and E-ResNets-I for skin cancer segmentation, and we show that E-nets yield better results when segmenting vessels in retina images.

4.1 E-nets-I for image classification

With our first experiments using the E-net-I, we mainly aimed to improve the computational efficiency of convolutional neural networks (CNNs). In many cases, 'deeper is better' is a standard strategy to enhance an architecture. However, this implies that the best computer vision models will mostly run on computers with the necessary hardware. The only way to use a high-performing CNN on a small device, such as a smartphone, would be to send a query image to a high-end server that quickly responds

with a prediction, making the client-server connection a possible bottleneck. To use CNNs directly on small devices, a whole branch of CNN research focuses on creating efficient models (e.g., [Ian+16; How+17; LZY21; TL19a; Li+17a; FC19]). A model is *efficient* if it can provide the outstanding performance of SOTA CNNs while needing fewer milliseconds, computations, or less memory.

If we assume that CNNs work similarly to the visual cortex, end-stopped neurons that are selective for 2D signals should emerge at some stage of the feature hierarchy. A CNN may implicitly model such an end-stopped neuron via a sequence of convolution layers and ReLUs (i.e., a sequence of so-called linear non-linear (LN)-neurons). However, it lacks the explicit AND combination to model an end-stopped neuron with only a few steps. If AND combinations are needed, an E-net with fewer E-blocks, and thus, fewer overall layers should create networks having a performance that matches larger conventional CNNs. Indeed, on ImageNet, we successfully trained an E-ResNet-I, which uses fewer parameters and layers than the original ResNet-50 while matching the validation accuracy. [1] Similarly, on Cifar-10, E-nets-I with fewer parameters outperform different ResNet models.

4.1.1 Experiments

4.1.1.1 E-ResNets-I on Cifar-10

Having concluded that substituting a certain number of conventional convolution blocks with a smaller number of E-blocks should make a CNN more efficient, there are still open questions concerning where to place the E-blocks or how many are needed. Using the end-stopped cell analogy, one would assume that E-blocks are more beneficial in earlier stages of the visual processing chain, i.e., after one or a few more convolution layers. However, how many E-blocks are needed and where to place them is difficult to estimate beforehand.

Accordingly, we ran a systematic evaluation by replacing entire stacks containing conventional blocks with entire E-block-I stacks. As a baseline, we used the Cifar-10 ResNet [He+16], the ResNet-32 being depicted in Figure 3.1 on page 28, because of its symmetric structure: the overall architecture starts with a stem consisting of a single convolution layer with batch normalization (BN) and ReLU. Next, there are three stacks containing $N \in \{3, 5, 7\}$ so-called *basic blocks* (see Section 2.3.3 on page 22) each. Because of the symmetric stack structure, we deemed these particular models to be a good fit for our experiments. In other models, the number of blocks per stack usually varies, making a systematic evaluation more challenging. For each of these three models, the ResNet-20 ($N = 3$), ResNet-32 ($N = 5$), and the ResNet-44 ($N = 7$), [2] we replaced one or two stacks with M E-blocks-I. Since we assumed that fewer blocks

[1] On ImageNet, only the validation labels are freely accessible to evaluate new architectures.
[2] The numbers refer to the number of convolution layers and dense layers in a ResNet.

suffice, we chose M as the number of blocks of the previous smaller network. I.e., if $N = 7$ then $M = 5$, if $N = 5$ then $M = 3$, if $N = 3$ then $M = 1$. For example, we substituted the seven blocks of the third stack of a ResNet-44 with five E-blocks-I. On Cifar-10, each E-block-I had an expansion factor $q = 2$.

In the following sections, when referring to a specific replacement scheme, we use binary strings; e.g., "011" denotes a model where the second and third stacks are E-stacks-I, consisting only of E-blocks-I. All other blocks are basic blocks that contain two sequences of 3×3 convolution, BN, and ReLU. Additionally, basic blocks employ the residual connection. Replacing one or more original layers yielded $2^3 - 1 = 7$ possible configurations. We disregarded the configuration "111" because it did not produce feasible results using only E-blocks-I. Hence, we trained six configurations. We report the minimal test error on Cifar-10 averaged over four or eight runs with different seeds (different weight initialization and batch order during training). The training details and hyperparameters are given in Section A.2.

4.1.1.2 E-ResNet-I on ImageNet

To test our design choices on a larger dataset, we trained an E-ResNet-I on the ImageNet benchmark. Our network was based on the ResNet-50 architecture containing four stacks consisting of bottleneck blocks. Initial tests showed that replacing the second and fourth stacks was a viable choice. For each, we used one E-block-I with an expansion factor $q = 1$. Note that we used no residual connections. A comparison of the baseline model and the E-ResNet-50-I is given in Table 4.2. See Section A.3 for training details and hyperparameters.

4.1.2 Results

Our main result is that by using E-blocks-I in ResNets, we obtained several parameter efficient E-ResNets-I. Figure 4.1 shows the results for six different E-ResNets-32-I, and the baselines ResNet-32 and ResNet-20. The choice of which layer to replace greatly impacted the number of parameters because the feature map dimensions increased with depth. Accordingly, substituting the middle and last layer decreased the number of parameters more than substituting the first layer. The configuration "100" yielded the best performance, which aligns with our assumption that E-blocks should be placed in front for optimal performance. However, E-nets of type "001" showed a good trade-off between parameter efficiency and generalization. Thus, we could create E-nets, with the ResNet-32 as the base network, that had much fewer parameters than the ResNet-20 and yet performed better on the test set.

Therefore, we present further results for configuration "001" with different base networks in Figure 4.2. Here, an even better network emerged: based on the ResNet-44,

Table 4.2: Comparison of the ResNet-50 and E-ResNet-50-I architectures. For rows 2 to 4, the first convolution down-samples the input with a stride of 2. Instead, if an E-block-I is involved, the output of the E-block-I is down-sampled by using max-pooling with a kernel size of 2 and a stride of 2.

Output size	ResNet-50	E-ResNet-50-I
112	7×7, 64, stride 2	
56	7×7 max pool, stride 2	
28	$\begin{bmatrix} 1 \times 1 & 128 \\ 3 \times 3 & 128 \\ 1 \times 1 & 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1 & 64 \\ 3 \times 3 & 64 \\ 1 \times 1 & 256 \end{bmatrix} \times 3$ $1 \times$ E-block-I $(256 \rightarrow 512), q = 1$ 2×2 max pool, stride 2
14	$\begin{bmatrix} 1 \times 1 & 256 \\ 3 \times 3 & 256 \\ 1 \times 1 & 1024 \end{bmatrix} \times 6$	
7	$\begin{bmatrix} 1 \times 1 & 512 \\ 3 \times 3 & 512 \\ 1 \times 1 & 2048 \end{bmatrix} \times 3$	$1 \times$ E-block-I $(1024 \rightarrow 2048), q = 1$ 2×2 max pool, stride 2
1	Global average pooling, linear layer	

we created an E-ResNet-44-I with performance comparable to the ResNet-32. However, we still managed to have fewer parameters than the ResNet-20.

Our data show a pattern: deeper networks with E-blocks-I (E-nets-I) outperform shallower baseline networks, although the E-nets-I have fewer parameters.

Remarkably, we could replicate this pattern on ImageNet. We present the results in Table 4.4: our E-ResNet-50-I achieves a validation error comparable to the original ResNet-50. Moreover, it outperforms the ResNet-34 and ResNet-18 and has 5 million parameters less than the ResNet-34. Compared to the Cifar-10 results, the ImageNet results are even better because the error of the E-ResNet-50-I – although being shallower than the ResNet-50 – does not increase.

Finally, for the ResNet-32 "001", we conducted an ablation study to quantify the effect of the filter pairs apart from other design decisions. For each new block, rather than learning two depthwise (DW) convolutions, only one was used with a ReLU non-linearity instead of a multiplication ('ResNet-32 Single Filter ReLU'). Thus, this reduced block consisted of a 1×1 convolution (with BN and ReLU), a DW convolution (with ReLU), and another 1×1 convolution with BN and ReLU. The results are shown in Table 4.3: using an E-layer with AND combination decreased the average test error

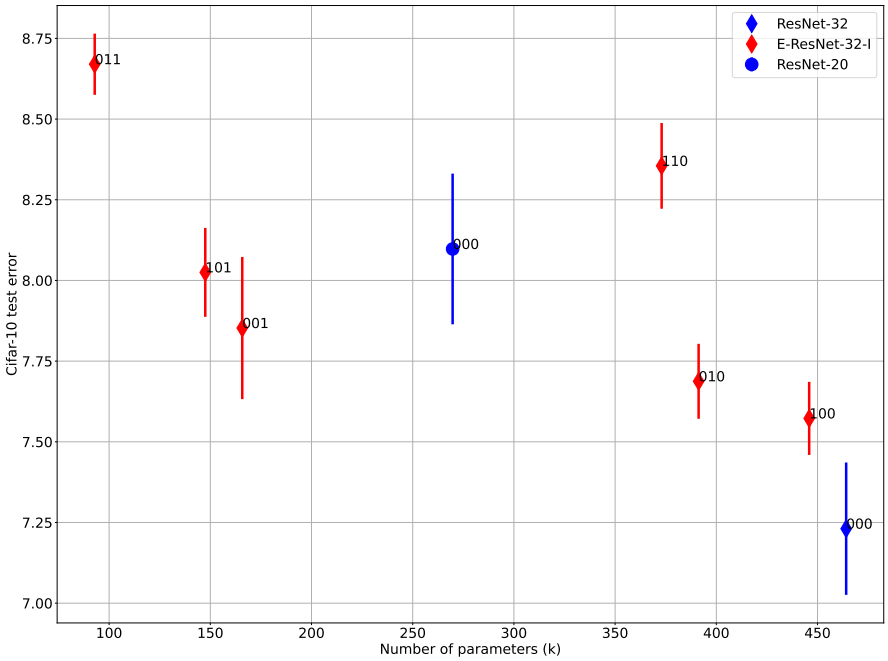


Figure 4.1: Results averaged over at least 4 runs with standard deviation for layer substitutions of the ResNet-32 on Cifar-10: the original ResNet-32 is made of three stacks containing five basic blocks. The binary string next to the point indicates which stack was substituted. E.g. for "101", we substituted the first and third original stacks with E-stacks-I, each consisting of three E-blocks-I.

by 0.4%, while only 3456 [3] parameters were added. Accordingly, when using E-blocks-I, we improved generalization over simple DW convolutions with only relatively few more parameters.

4.1.3 Discussion

Our results show that by adding E-blocks-I to established architectures, one can create E-nets-I as hybrid CNNs that are more efficient, i.e., they generalize well with fewer parameters. Moreover, transforming any standard CNN to an E-net-I is straightforward: we did not change the structures of the base networks and did not perform any hyperparameter tuning. In conclusion, we can recommend using E-blocks-I as building blocks for more efficient deep networks, the E-nets-I.

The ablation study suggests that AND combination, an idea derived from biological vision, is a vital design choice for alternative CNN architectures.

[3] $3456 = 9$ [entries per kernel] $\times$ 64 [feature maps] $\times 2[q] \times 3$ [blocks].

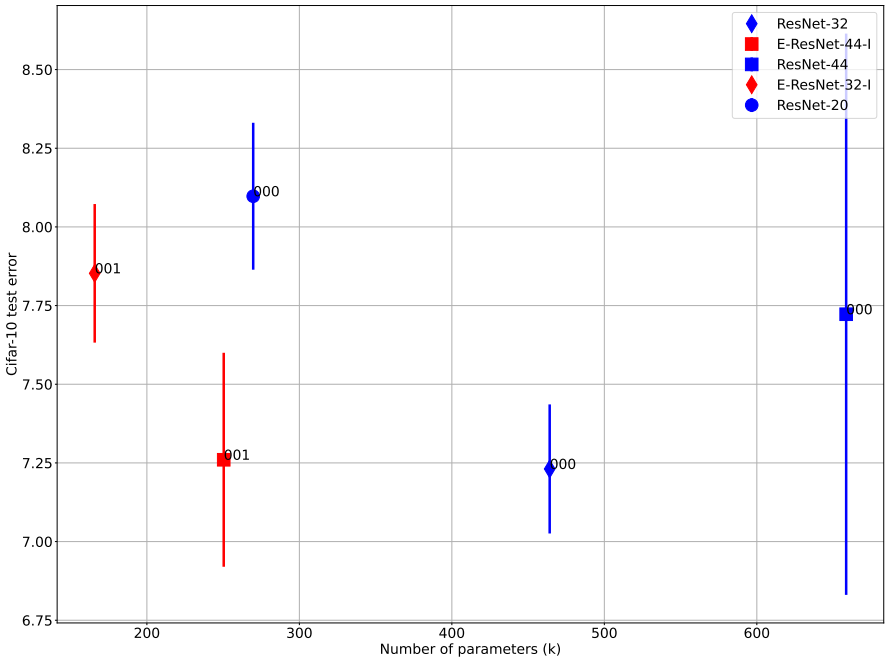


Figure 4.2: Results averaged over at least 4 runs with standard deviation for substituting the last stack on different ResNet architectures: the blue color shows the ResNet models (point: ResNet-20, diamond: ResNet-32, square: ResNet-44). The red color shows the E-nets-I; where we substituted the original last stack that contains N basic blocks (7 for the ResNet-44, 5 for the ResNet-32, 3 for the ResNet-20) with fewer E-blocks-I (5 for the E-ResNet-44-I, 3 for the E-ResNet-32-I).

However, our initial assumption – E-Nets-I need fewer operations – was only clearly met with the ImageNet experiment. One possible explanation may be that E-nets-I need more data to achieve the same performance as their corresponding baselines. Another explanation is that we did not choose an optimal design rule by replacing entire stacks. Results for the E-net-R and E-net-M will show that smaller E-ResNets on Cifar-10 can outperform their baselines. Though, those models always mix convolution blocks with E-blocks within a stack.

In the next section, we test E-nets-I on a more fine-grained task strongly connected to human vision: the subjective assessment of image quality (IQA). We show that AND combinations help find regions of interest and present another hybrid model with even fewer operations than the E-ResNet-32-I.

Table 4.3: Cifar-10 ablation study: to show that multiplying DW filter pairs has an actual beneficial effect, we compared our results to a single DW filter output with an additional ReLU. The number of parameters is shown in thousands (k).

Model	N. param. (k)	Test error
ResNet-32 001 Single Filter ReLU	162	8.25 ± 0.17
E-ResNet-32-I 001	166	7.85 ± 0.22

Table 4.4: ImageNet validation results: we compared our architecture to the PyTorch [Pas+19] pre-trained ResNets. The number of parameters is shown in millions (M)

Model	N. param. (M)	Val. error
ResNet-18	11	30.34
ResNet-34	21	26.78
ResNet-50	23	23.88
E-ResNet-50-I	16	23.87

4.2 E-nets-I for image quality assessment

In recent years, we witnessed a staggering increase in the consumption and distribution of visual content, primarily on social media, streaming platforms, and digital television. Apart from the extension of digital infrastructure, the technology of digital cameras also evolved. Nevertheless, distortions due to all kinds of sources can still deteriorate image quality. Thus, the need for efficient and effective IQA is growing. The desired application could be, for example, a quality score computed with every taken image or a quality ranking of a set of images. Approaches with high efficiency, low computational cost, and memory consumption would be beneficial for running IQA on mobile devices and embedded systems. Several datasets with artificial [Pon+15; LC10; SSB06] or natural distortions [Hos+20; GB15], and human image quality ratings are available. These datasets usually pose IQA as a regression problem. Given an image, one has to assess the level of distortion either compared to an undistorted reference image or with no reference (NR-IQA, blind IQA). We focus on the more challenging task of IQA without a reference image. Many of today's SOTA IQA approaches successfully use CNNs. However, we hypothesize that E-nets could be particularly beneficial for the subjective task of IQA since principles of human vision inspire them.

Related work Several algorithms for IQA and several benchmarks have already been proposed (for an overview, we refer to Zhai and Min [ZM20], and Kim et al. [Kim+17]). We evaluated our algorithms for blind IQA on the LIVE legacy dataset (Legacy) with artificial distortions and two datasets with natural distortions: LIVE in the wild (LITW) and Kon-IQ (see Figure 4.3 for LITW examples).

Apart from CNN-based approaches, handcrafted or unsupervised features in combination with regression models are used on these datasets. BRISQUE [MMB12], for example, is based on the statistics of locally normalized luminance coefficients and CORNIA [Ye+12] on a dictionary learned from image patches. Tu et al. [Tu+21] created an NR-video quality assessment model by carefully selecting a subset of statistical features used in other SOTA algorithms. Pei and Chen [PC15] presented a full-reference IQA model based on difference of Gaussian features paired with a random-forest regressor.

For the Legacy dataset (see Figure 4.4 for an example), Kang et al. [Kan+14] first presented promising results using CNNs with only two convolution layers. Several small 32×32 patches are extracted from an image, and a quality value is estimated for each patch. These quality values are then combined to obtain a single quality value for the entire image. This process is called patch-based fusing. Although it was assumed that high-level features do not contribute much to the visibility of image distortions, Bosse et al. [Bos+16] showed that deeper networks can yield better results. Even deeper models [VSS18; Kim+17] and the use of pre-training [Bia+18] were explored further. Liu, Weijer, and Bagdanov [LWB17] used unlabelled data to pre-train a VGG-16 [SZ14] network on a large IQA-related dataset. Cheng, Takeuchi, and Katto [CTK17] showed that the prediction errors of a model are higher in patches of homogeneous areas. Therefore, they used saliency to weigh each patch. Similarly, we use the more salient patches with a simple saliency measure based on the Structure Tensor [Jäh93].

For datasets with authentic distortions, such as LITW and Kon-IQ, successful CNN approaches use deeper and wider networks with larger input patches and CNNs pre-trained on the ImageNet dataset. Kim et al. [Kim+17] used pre-trained networks such as the ResNet-50 [He+16] and fused scores of several random patches of typical ImageNet inputs sizes (224×224 or 227×227). They argue that the features learned on ImageNet to represent natural images are also viable in characterizing natural distortions, not artificial ones. Bianco et al. [Bia+18] increased their model's performance by first training it on ImageNet and the Places [Zho+17] dataset. This pre-trained network was then fine-tuned on a different version of the LITW dataset using a classification task and support vector regression on the resulting feature vector. Varga, Saupe, and Szirányi [VSS18] used a similar fine-tuning step with a spatial pyramid pooling layer and a multilayer perceptron (MLP). Zhang et al. [Zha+18a] introduced the use of bilinear CNNs for IQA. Ma et al. [Ma+17] trained a multi-task CNN to compute a quality score by assigning each input image to a specific distortion class.

4.2.1 Methods

This section first explains the fusing of scores from different image patches. Next, we show how the structure tensor can extract more salient patches. Finally, we present the E-nets-I used in our experiments.

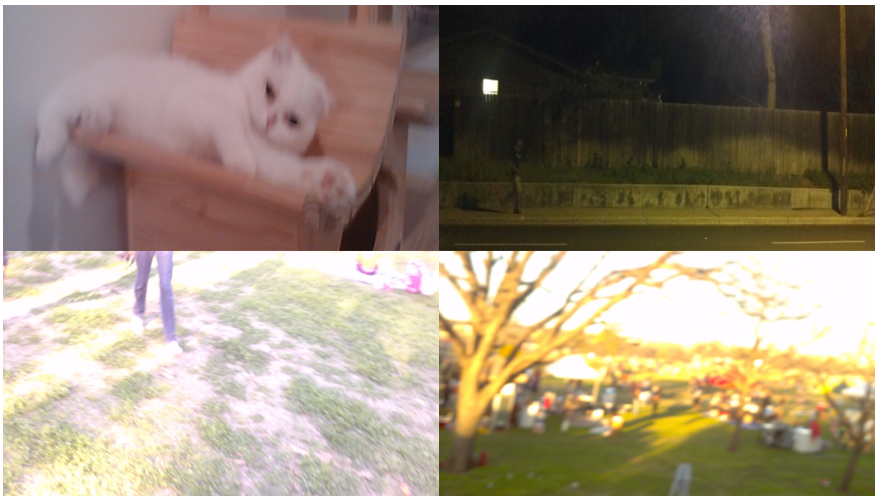


Figure 4.3: Examples for the LITW dataset: the dataset consists of authentic images captured mainly using mobile devices. The examples show motion blur (top-left), under-exposure and noise (top-right), over-exposure (bottom-left), and blurring (bottom-right).

4.2.1.1 Patch-based fusing

Patch-based fusing is a typical approach when employing CNNs in IQA. A network used for regression is presented with several patches of an image to determine its quality. For each patch, a score is computed. These values are combined into one global score. In blind IQA, the desired output for an input image is a score for subjective quality. We train a CNN that is presented with a subset of image patches, a batch, to predict the scores directly. We use the absolute difference between the CNN's prediction and the ground truth as the loss function. Only patches with constant height and width are processed to ensure all batch images have the same dimensions.

In the testing phase, we follow the typical approach of fusing patch-based predictions as illustrated in Figure 4.5: to determine an image's quality, the trained network processes several image patches, yielding one score for each patch. These scores are then averaged to create the final output. When using this patch-based approach, one important design choice is the actual patch size: how much context is needed to predict the overall quality? Research indicates that smaller patches are needed for artificial distortions, while natural distortions require larger patches [Kim+17]. Moreover, for natural distortions, good results could only be achieved with sufficient pre-training on large-scale datasets such as ImageNet. Hence, the quality assessment of natural distortions is a more demanding task requiring a considerable amount of data. So far, no IQA dataset exists that comes close to a million or more samples, which is not

Figure 4.4: Example from the Legacy dataset: a reference image is distorted by different operations. Image quality is encoded by a number between 1 (high) and 100 (very poor), and the ground truth is the mean score of several annotators. Top left: reference; top right: white noise; bottom left: JPEG compression; bottom right: Gaussian blur.

surprising since label acquisition for IQA is a more intricate task compared to, e.g., label acquisition for image recognition. [4]

The fusing of patch-based predictions relies on the assumption that the scores of the individual patches capture the score of the whole image. This assumption is convenient since one can use several hundred patches per image to train a model, but it is not necessarily valid for all patches. Accordingly, some patches are better suited for prediction. We enhanced our results by selecting patches using an attention model based on the Structure Tensor.

4.2.1.2 Attention model

For artificial distortions, we can assume that small patches already contain sufficient information about perceptual quality. However, the predictive quality of a particular patch heavily depends on the image structure in that patch. Figure 4.6 gives an example of how the predictive quality of patches can vary for JPEG 2000 compression. The right side of Figure 4.6 shows the patch-wise absolute distance between the ground truth and the local prediction of a CNN. One problem that can be observed is that blurring and compression effects do not alter homogeneous areas; therefore, no information can be gained there. Hence, one should focus on patches that contain more structure. To

[4] As the quality score of an image is subjective, an annotator may need more time for scoring (IQA) than for simply classifying which object can be seen in an image (classification). Furthermore, to not rely on a single person's opinion, more than one annotator should score an image.

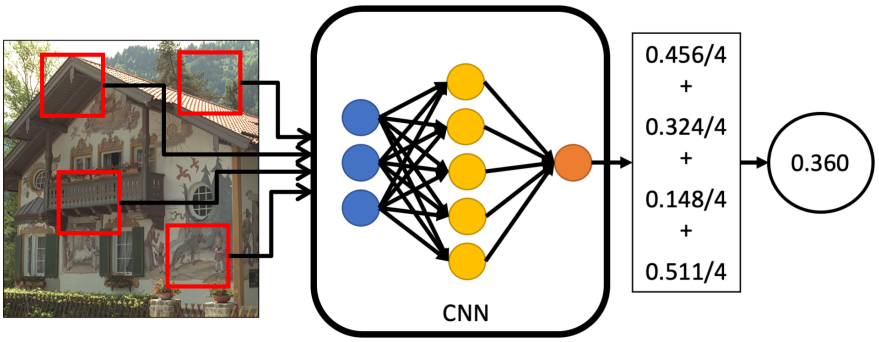


Figure 4.5: Patch-based processing and fusing for IQA: to infer the score for one image, the CNN is presented with several image patches. For each patch, a score is predicted. The final score for the whole image is the mean value of all individual scores. For the datasets with authentic distortions, we selected patches randomly. We compared an attention-based patch selection to a random selection for the Legacy datasets.

this end, we present an effective strategy to sample such patches by using the Structure Tensor [Jäh93]:

$$\mathbf{J} = \int_{\Omega} \begin{bmatrix} I_x I_x & I_x I_y \\ I_y I_x & I_y I_y \end{bmatrix} d\Omega; \tag{4.1}$$

I_x and I_y are the image derivatives in horizontal- and vertical directions; $d\Omega$ is a local region. High values of the determinant of $\mathbf{J}$ indicate areas with 2D structure – i.e., 2D patches. Note that the determinant of $\mathbf{J}$ bears a strong resemblance to the determinant of the Hessian in Equation 3.1.

In our implementation, we transform the RGB input image to a grayscale image and convolve it with a Gaussian filter with $\sigma_{pre} = 3$. We computed the derivatives using Sobel filters. After computing the product terms $I_x I_x$, $I_x I_y$, and $I_y I_y$, we filter the product terms with a Gaussian filter with $\sigma_{post} = 5$ (integration over a local region). Finally, we compute the determinant and apply non-maximum suppression in a window of 15 pixels to obtain a feature map where each pixel encodes the amount of structure (examples for $I_x, I_y, I_x I_y$, and the determinant of $\mathbf{J}$ are shown in Figure 4.7). To select the best n patches, we use the top n saliency values and crop patches around the respective center pixel. The pseudo-code for the method is given in Table 4.5. The invariants of the Structure Tensor have already been used successfully to model human attention and saliency [Vig+11], i.e., areas where humans tend to look in an image.

4.2.1.3 E-net models

We tested different approaches using E-nets for three blind IQA datasets. The Legacy dataset contains artificial distortions of high-quality reference images. The LITW and

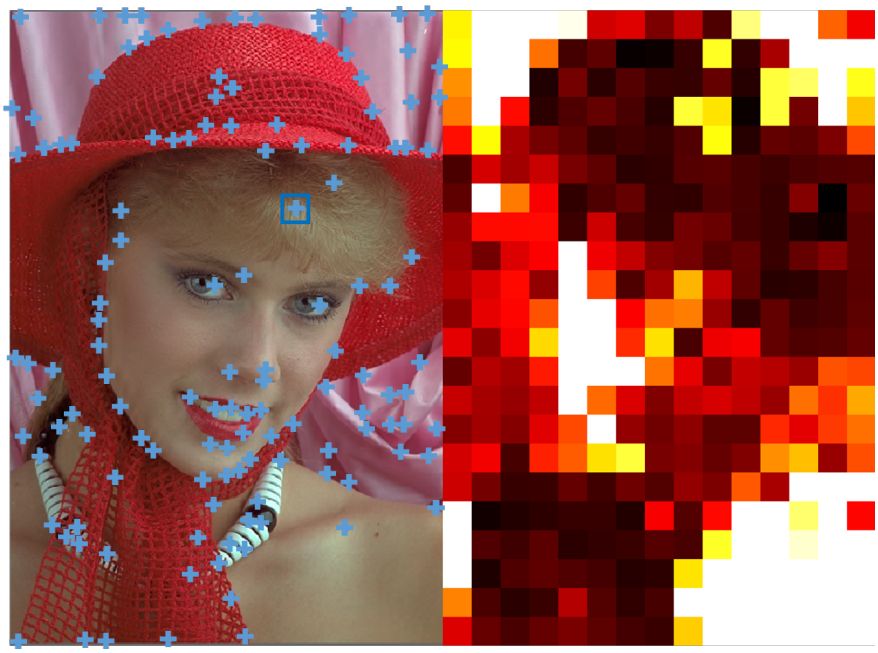


Figure 4.6: For the image on the left, the right panel shows the differences between the prediction and the ground truth for local patches (for white pixels, the absolute difference is > 25). Note that predicting the image quality in a homogeneous area is more complicated. Therefore, we used the Structure Tensor to sample patches (see the blue box in the left image) with high structural information (blue crosses show patch centers).

Kon-IQ datasets contain images with natural distortions. For all three benchmarks, the E-nets-I were trained on image patches. To infer the overall quality of an image, the values obtained for different patches were averaged (patch-based fusing). The approaches used for the artificial and authentic datasets differ in the size of a patch, the way patches are selected, the use of pre-training, and the network size.

Since the Legacy dataset has smaller input patches, we used the ResNet-32 employed on Cifar-10 (see Section 4.1.1.1) as a baseline. We evaluate an E-ResNet-32-I "001", where we substituted the last stack, i.e., five basic blocks, with three E-blocks-I. In the following, we will refer to this network as E-ResNet-32-I.

Replacing an entire stack with a set of E-blocks is just one variant of enriching a typical CNN architecture. Accordingly, this approach may not always be optimal, and combinations of E-blocks within a stack may yield better results. [5] In order to find a suitable architecture, we ran several experiments with different E-nets-I on Cifar-10 and evaluated them. We widened the search space to different convolutional blocks

[5] Section 4.7, further elaborates on E-block positioning.

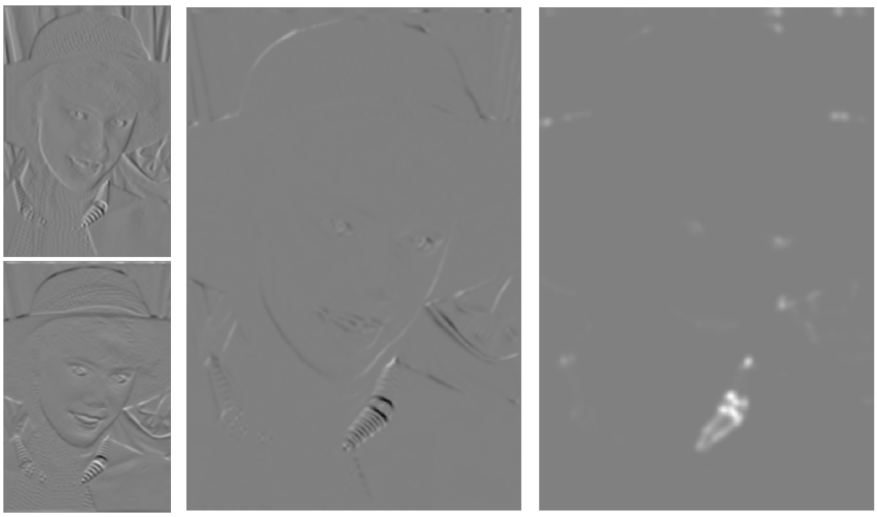


Figure 4.7: Left top and bottom panels show the horizontal and vertical derivatives of an image, and the middle panel shows the pixel-wise multiplication of the two derivatives. The determinant of the Structure Tensor (right) encodes the amount of structure in a local area.

Table 4.5: Pseudo-code of the attention-based inference method. For random cropping, steps 1 to 4 are substituted by 'crop n patches randomly'.

(1)	For the input image $\mathbf{I}$, compute the determinant of $\mathbf{J}$ for each pixel (x, y)
(2)	Apply non-maximum suppression
(3)	Find the top n pixels $(x_i, y_i), i = 1, ..., n$ with the highest saliency values
(4)	Crop patches $\mathbf{p_i}$ with (x_i, y_i) in the center
(5)	For each $\mathbf{p_i}$ compute the network prediction $f(\mathbf{p_i})$
(6)	Return the score of $\mathbf{I}$ as $F(\mathbf{I}) = \frac{1}{n} \sum_{i=1}^{n} f(\mathbf{p_i})$

and adopted the pyramid block and bottleneck block structure presented in Han, Kim, and Kim [HKK17]'s Pyramid Residual Networks (see Section 2.3.3).

Finally, we found one architecture, displayed in Figure 4.8, that performed exceptionally well with even fewer layers and parameters. We compare this architecture, denoted E-CustomNet-I, to the E-ResNet-32-I. For the E-CustomNet-I, we found that pre-training on Cifar-10 was not beneficial. Furthermore, each E-block-I in the E-CustomNet-I was augmented with residual connections (see Section 2.3.2), which we did not use in the E-ResNet-32-I. Again, an expansion factor $q = 2$ was used.

Models dealing with authentic distortions (LITW and Kon-IQ datasets) require larger input sizes and pre-training on larger datasets such as ImageNet. To evaluate E-nets-I for this more challenging task, we compared a ResNet-50 to the corresponding E-ResNet-50-I (see Section 4.1.1.2), both pre-trained on ImageNet.

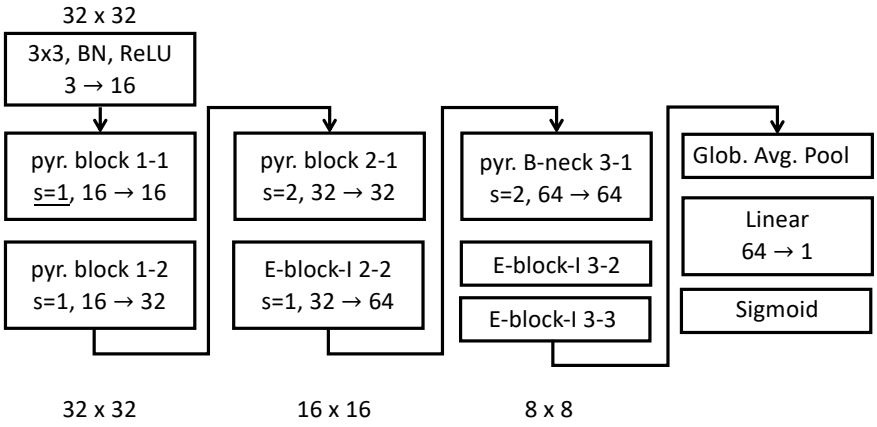


Figure 4.8: Schema of the E-CustomNet-I: using compositions of convolutional blocks and E-blocks, we can further reduce the number of blocks and parameters while improving the network's performance. Here, we used the pyramid blocks and bottleneck blocks ('pyr. B-neck') of Pyramid Nets [HKK17] (see also PyrNets in Section 2.3.3).

4.2.2 Experiments

Table 4.6 provides an overview of the experiments. All experiments were conducted using PyTorch [Pas+19].

4.2.2.1 Legacy dataset (artificial distortions)

We report results for 10 random 80/20 splits of the LIVE legacy database [SSB06], which contains 981 distorted images obtained from 29 reference images (see Figure 4.4 on page 55 for an example). Each image was rated by up to 29 subjects on a continuous scale ranging from 'poor' to 'excellent' quality. With these ratings, a score in the range $[1, 100]$ was calculated for each image, with 1 for the best quality and 100 for the worst quality. As in Bosse et al. [Bos+16], we used 17 images for training, 6 for validation and early stopping, and 6 for testing. The ResNet-32 and the E-ResNet-32-I were pre-trained on Cifar-10, see Section 4.1.1.1 – the E-CustomNet-I was not. We used a patch size of 32×32. For validation and early stopping, we randomly sampled 32 patches of each validation image. The scores obtained for the patches were averaged, and then the Pearson linear correlation coefficient (PLCC) and the Spearman's rank-order correlation coefficient (SROCC) between the prediction scores and the validation scores were computed. For testing, we used the model with the highest validation PLCC. We compared two sampling strategies: random sampling and attention-based sampling, i.e., for each test image, we either sampled 128 samples randomly or selected the 128 samples with the highest saliency (defined by the determinant of $\mathbf{J}$). Details on data augmentation and hyperparameters are given in Section A.5.1.

Table 4.6: Overview of the performed experiments. The distortions seen in the Legacy dataset are created artificially. The Kon-IQ and LITW datasets contain images with authentic distortions.

Model	Patch size	Patch selection	Dataset	Distortions	Pre-training
ResNet-32	32×32	Random	Legacy	Art.	Cifar-10
E-ResNet-32-I	32×32	Random	Legacy	Art.	Cifar-10
E-CustomNet-I	32×32	Random	Legacy	Art.	None
ResNet-32 (J)	32×32	Attention-based	Legacy	Art.	Cifar-10
E-ResNet-32-I (J)	32×32	Attention-based	Legacy	Art.	Cifar-10
E-CustomNet-I (J)	32×32	Attention-based	Legacy	Art.	None
ResNet-50	224×224	Random	LITW / Kon-IQ	Auth.	ImageNet
E-ResNet-50-I	224×224	Random	LITW / Kon-IQ	Auth.	ImageNet

4.2.2.2 Datasets with authentic distortions

With LITW, Ghadiyaram and Bovik [GB15] compiled a demanding benchmark containing 1,162 images with 350,000 scores from over 8,100 human observers. The images were captured mainly by using mobile devices in everyday situations. Thus, apart from distortions due to the device's processing chain, the photographer himself can impair image quality: due to over- and underexposure, motion-blur, and other sources, see Figure 4.3 on page 54 for examples. The currently largest labeled dataset with authentic distortions is Kon-IQ [Hos+20], containing 10 073 images of size 512×384 with 1.2 million quality ratings. We used the same approach for both datasets: as with the Legacy benchmark, we trained and tested CNNs for ten different 80/20 splits. On LITW, Kim et al. [Kim+17] obtained good results with a pre-trained ResNet-50, and we therefore compared a ResNet-50 with a corresponding E-ResNet-50-I, both pre-trained on ImageNet. Due to the large patch size of 224×224, attention-based patch selection yielded no benefit. Instead of 128 patches, each test image score was predicted with 25 random patches. More training details are given in Section A.5.2. To better evaluate the generalization capabilities of the models, all networks trained on LITW were tested on the full Kon-IQ dataset and vice-versa.

4.2.3 Results

In addition to the performance scores, we report the number of parameters in millions (M) to compare for efficiency. If possible, we determined the number of parameters from the respective paper's description. In many cases, the CNNs used in the literature were large and combined with even larger MLPs. In those cases, we report only the CNN's standard size as a lower bound. A '>' denotes that the size is even larger due to, e.g., an additional MLP. Note that with the E-nets, no additional regression models

were used. We do not report the number of parameters for BRISQUE [MMB12] and CORNIA [Ye+12].

Figure 4.9 presents results for all configurations tested on the Legacy dataset. First, note that our simple attention model (denoted by (J)) increased the median PLCC by 0.2% for the ResNet and by 0.4% for both E-nets (with %, we denote an absolute increase of 1/100, e.g., 0.975 with an 0.2% increase yields 0.977). Also, note that the E-nets-I with attention model outperformed the ResNet by 0.2%. Table 4.6 shows that E-nets-I yield comparable results to the SOTA, which is remarkable since the number of parameters is reduced by a substantial amount. Furthermore, we outperformed the E-ResNet-32-I in terms of mean PLCC by 0.1% using the E-CustomNet-I architecture (the minimum and maximum PLCC values also increased) with even fewer convolution layers and parameters. Note that the SROCC scores of both E-nets are better than the ResNet-32's and that the attention mechanism increased the SROCC scores of all three architectures.

The results for the ten splits of the LITW dataset are shown in Figure 4.10. Here, the E-net-I has a higher variance (0.02 vs. 0.015) with a lower minimum and a higher maximum score. The mean value is decreased by 0.3% (0.859 vs. 0.856), and the median value is decreased by 0.2% (0.860 vs. 0.858). Nevertheless, these results show that a comparable performance is possible with fewer convolution layers and significantly fewer parameters. Table 4.8 shows results for other SOTA approaches on the LITW dataset.

Regarding the Kon-IQ results shown in Table 4.9, the E-ResNet-50-I performed on par with the ResNet-50 baseline. One can observe a similar outcome for the cross-database results in Table 4.10. Here, the differences between the mean PLCC and SROCC values are within one standard deviation (that was equal for the ResNet-50 and the E-ResNet-50-I). When trained on LITW and evaluated on Kon-IQ, the standard deviation was 0.01. When trained on Kon-IQ and evaluated on LITW, the standard deviations were 0.005 and 0.004 for PLCC and SROCC, respectively.

4.2.4 Discussion

We used E-nets-I to asses perceptual image quality and obtained results comparable to the SOTA with fewer parameters and layers. We expected this outcome since IQA is based on human perception. Thus, models based on principles of human vision should be beneficial. In our case, E-nets explicitly model end-stopping with AND combinations of filters.

The strength of AND combinations is furthermore indicated in the attention model of Section 4.2.1.2. Here, AND combinations are essential to finding patches with much structure, i.e., having an intrinsic dimensionality (iD) of two.

Using the attention model improved the baseline ResNet-32 and the E-nets-I. In contrast to classification on Cifar-10, the E-ResNet-32-I outperformed the ResNet-32 on

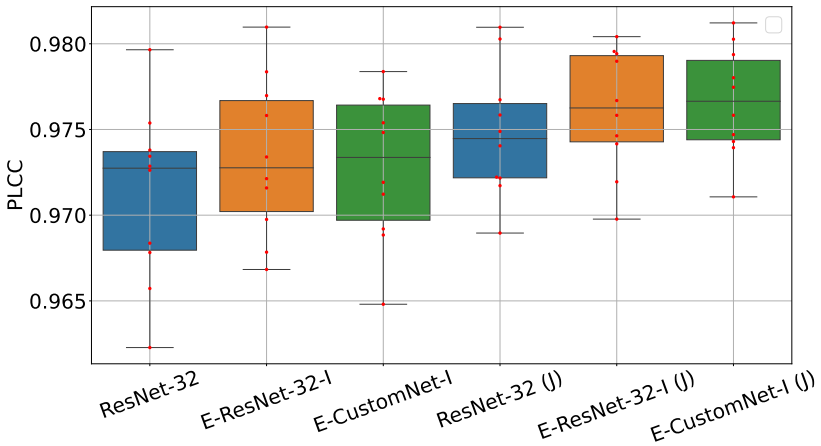


Figure 4.9: Results for the Legacy dataset obtained for ten different 80/20 splits: the first three boxes are obtained with the random sampling method for the ResNet-32 (blue, architecture shown in Figure 3.1), the E-ResNet-32-I (orange), and the E-CustomNet-I (green, architecture shown in Figure 4.8). The results for attention-based patch selection, marked with a (J), are illustrated with the last three boxes. Red dots indicate the actual test values obtained for each split.

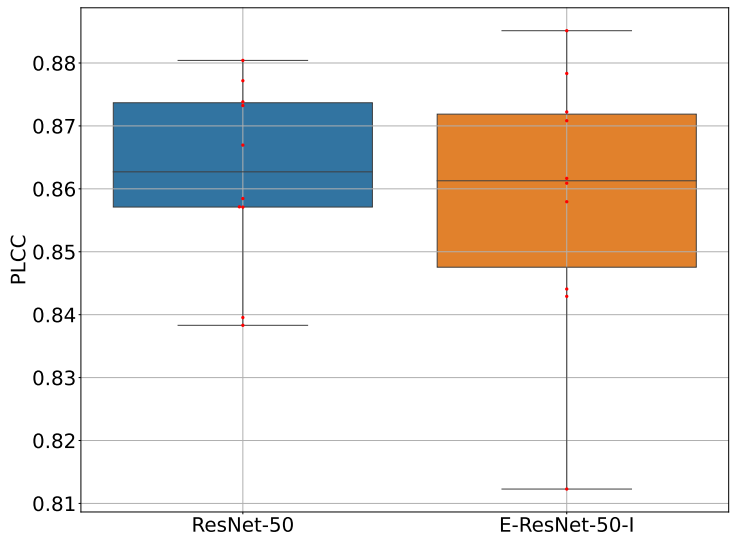


Figure 4.10: Results for the ResNet-50 and the corresponding E-ResNet-50-I on the LITW dataset.

Table 4.7: Results for the Legacy dataset (mean values): J denotes the use of the attention model. The column 'N. param. (M)' shows the number of parameters in millions.

Model	PLCC	SROCC	N. param. (M)
Mittal, Moorthy, and Bovik [MMB12]	0.942	0.940	–
Ye et al. [Ye+12]	0.935	0.942	–
Kang et al. [Kan+14]	0.953	0.956	0.72
Bosse et al. [Bos+16]	0.972	0.960	4.97
Kim, Nguyen, and Lee [KNL18]	0.977	0.975	0.40
Cheng, Takeuchi, and Katto [CTK17]	0.978	0.974	4.82
Bianco et al. [Bia+18]	0.98	0.97	21.58
Varga, Saupe, and Szirányi [VSS18]	0.98	0.98	>44.7
Zhang et al. [Zha+18a]	0.971	0.968	>138
Liu, Weijer, and Bagdanov [LWB17]	0.982	0.981	138
ResNet-32	0.971	0.959	0.46
ResNet-32 (J)	0.975	0.963	0.46
E-ResNet-32-I	0.973	0.962	0.17
E-ResNet-32-I (J)	0.976	0.965	0.17
E-CustomNet-I	0.973	0.961	0.14
E-CustomNet-I (J)	0.977	0.965	0.14

Table 4.8: Comparison of different approaches on the LITW dataset. Results for Mittal, Moorthy, and Bovik [MMB12] and Ye et al. [Ye+12] are reported from Kim et al. [Kim+17].

Model	PLCC	SROCC	N. param. (M)
Mittal, Moorthy, and Bovik [MMB12]	0.607	0.585	–
Ye et al. [Ye+12]	0.618	0.662	–
Bianco et al. [Bia+18]	0.908	0.889	21.6
Zhang et al. [Zha+18a]	0.869	0.851	>138
Varga, Saupe, and Szirányi [VSS18]	0.93	0.91	> 44.7
Kim et al. [Kim+17]	0.849	0.819	23.5
ResNet-50	0.859	0.843	23.5
E-ResNet-50-I	0.856	0.839	14.1

Table 4.9: Comparison of different approaches on the Kon-IQ dataset: all results, except for ResNet-50, E-ResNet-50-I, and Varga, Saupe, and Szirányi [VSS18], are reported from Hosu et al. [Hos+20].

Model	PLCC	SROCC	N. param. (M)
Mittal, Moorthy, and Bovik [MMB12]	0.707	0.705	–
Ye et al. [Ye+12]	0.808	0.780	–
ResNet-50	0.920	0.909	23.5
E-ResNet-50-I	0.918	0.907	14.1
Varga, Saupe, and Szirányi [VSS18]	0.95	0.92	>44.7
DeepBIQ (VGG-16)	0.886	0.872	>18.1
DeepBIQ (Inception)	0.911	0.907	>60
KonCept512	0.937	0.921	>60

Table 4.10: Cross-database results: for the LITW column, the networks were trained on Kon-IQ and tested on LITW, and vice-versa for the Kon-IQ column. All results, except for ResNet-50 and E-ResNet-50-I, are reported from Hosu et al. [Hos+20].

Model	LITW (PLCC / SROCC)	Kon-IQ
Mittal, Moorthy, and Bovik [MMB12]	0.598 / 0.561	–
Ye et al. [Ye+12]	0.644 / 0.621	–
ResNet-50	0.790 / 0.786	0.799 / 0.744
E-ResNet-50-I	0.786 / 0.782	0.793 / 0.738
DeepBIQ (VGG-16)	0.747 / 0.742	–
DeepBIQ (Inception)	0.821 / 0.804	–
KonCept512	0.848 / 0.825	–

the Legacy dataset. Accordingly, the inductive bias of the E-ResNet-32-I may be better suited for IQA with artificial distortions. On authentic distortions, the ResNet-50 outperformed the E-ResNet-50-I by a small margin. Still, given its increased efficiency, the E-ResNet-50-I is a viable alternative to the standard ResNet-50.

The results of the E-CustomNet-I show that more elaborate design rules than exchanging entire stacks yield E-nets that are very efficient while outperforming standard CNNs and being on par with many SOTA IQA methods (see Table 4.7). In the following sections, we will present other E-blocks and design rules that outperform standard CNNs while also being more robust to noise and adversarial attacks.

4.3 E-nets-R for image classification

Extending the promising results of the E-net-I, we present an updated version with the E-net-R. Among other changes described in detail in Section 3.7, we now apply **R**eLU non-linearities directly to each filter output of the filter pair (i.e., we change ϕ), thus, increasing the selectivity of the E-blocks. Accordingly, each AND combined filter pair only yields a considerable value if both filters yield a high positive activation; in all other cases, only a small or no activation is returned. Furthermore, we employ a different design rule to create E-nets that mix E-blocks-R with convolutional blocks within a stack.

As opposed to the E-net-I, we aimed to improve the performance of the employed baseline CNNs rather than making them more parameter efficient. Thus, although the number of parameters is not reduced as drastically – some E-nets-R are even slightly larger than their baselines – we present networks that perform better than their baselines on ImageNet and Cifar-10.

4.3.1 Experiments

We designed four different E-nets-R based on distinct baseline architectures: an E-net-R based on (i) the original ResNet (already described in Section 4.1.1.1), and (ii) the PyrNet trained on Cifar-10 [6], (iii) a ResNet-50, and (vi) a MobileNet-V2 both trained on ImageNet. These architectures differ in the numbers and types of blocks: basic blocks, pyramid blocks, bottleneck blocks, and mobile inverted bottlenecks are the main components of the ResNet, the PyrNet, the ResNet-50, and the MobileNet-V2 architecture, respectively (some of the blocks are depicted in Figure 3.2 on page 29). Compared to the Cifar-10 architectures (ResNet and PyrNet) with three stacks, the ImageNet architectures consist of more stacks: the ResNet-50 has four stacks, and the MobileNet-V2 has six stacks.

We transform these four baseline architectures defined above into E-nets-R using a straightforward design rule: *substitute each stack's first block with an E-block-R.* The input and output dimensions of the block are kept equal; only the internal operations differ. In previous experiments, we found that combinations of convolution blocks and E-blocks work best and that larger kernel sizes do not improve performance. We chose this particular design rule because it performed best among several experiments with different rules. [7] In addition, from a brain-inspired perspective, this rule seems sensible: one way to view a stack is that it constitutes a visual processing chain for a specific image scale. Hence, one would expect end-stopping to be more helpful at the beginning

[6] A PyrNet is a ResNet consisting of pyramid blocks instead of a basic blocks, see Section 2.3.3.

[7] After the publication, we further reviewed the impact of where to position E-blocks-M. On Cifar-10, we can show that there exist configurations that outperform the design rule proposed here. These results are presented in Section 4.7

of this chain. Thus, we replaced the first block of each stack. However, note that later stacks, e.g., the second and third stacks in the Cifar-10 networks, already work with highly processed inputs from the previous stacks. Therefore, one would expect a lower necessity of extracting 2D regions in later stacks. Indeed, when analyzing the angles between the filter vectors of a filter pair ($\gamma = \measuredangle(\mathbf{v}, \mathbf{g})$), we will show that highly selective neurons are more common in earlier stacks.

4.3.1.1 E-nets-R on Cifar-10

As in Section 4.1.1.1, we evaluated the ResNet architecture on Cifar-10. Each ResNet contains three stacks, each consisting of N blocks. We evaluated two types of the ResNet-20, ResNet-32, ResNet-44, and ResNet-56, with N=3, 5, 7, and 9 blocks, respectively (the numbers after the names indicate the number of linear layers – including convolutions). Since the first publication of the ResNet architecture, several additional blocks have been proposed; see Han, Kim, and Kim [HKK17] for an overview. As two baselines on Cifar-10, we used the original ResNet and a variant using the pyramid block that we denote PyrNet. For both variants, we created E-nets-R by replacing baseline blocks with E-blocks-R according to our design rule. We used the same number of blocks, but note that an E-block-R contains one additional convolution layer in each block. [8] The E-PyrNet-20-R, E-PyrNet-32-R, E-PyrNet-44-R, and E-PyrNet-56-R are based on the PyrNet: each stack's first block is an E-block-R, and all other blocks are pyramid blocks. Analogously, for an E-ResNet-R, each stack's first block is an E-block-R, and the remaining blocks are basic blocks. For all E-blocks-R, we used $q = 2$.

4.3.1.2 E-nets-R on ImageNet

Next, we evaluated the performance of E-nets-R with the larger ImageNet dataset. We compared the ResNet-50 to two E-ResNet-50-R: one smaller net with an expansion factor $q = 0.8$ and a slightly larger network with $q = 1$. In both cases, for each of the four stacks of the ResNet-50, the first block was replaced by an E-block-R to obtain the E-ResNet-50-R. Note that, if not explicitly mentioned, the term E-ResNet-50-R refers to the $q = 1$ variant.

We evaluated an E-MobileNet-V2-R based on the popular MobileNet-V2 architecture to validate our approach further. As with the ResNet, we replaced the first block of each stack with an E-block-R ($q = 3$).

[8] To keep consistent with our naming convention, we keep the original number. E.g., E-PyrNet-20-R is based on the PyrNet-20, having 20 linear layers. However, the E-PyrNet-20-R actually has 23 linear layers.

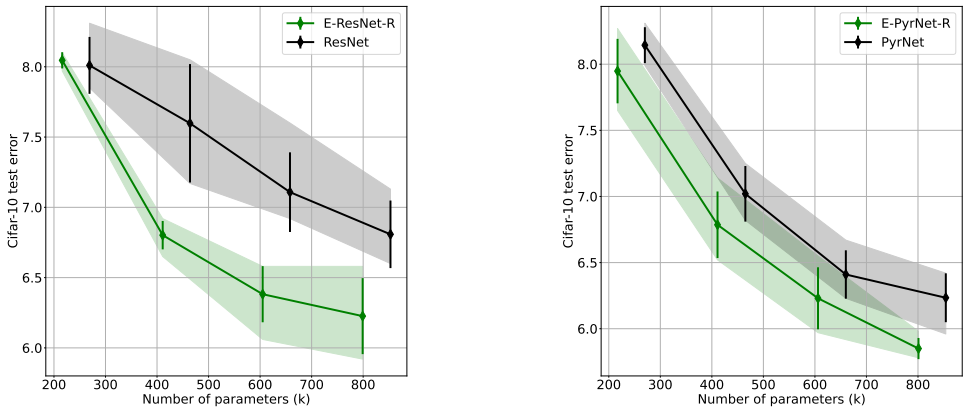


Figure 4.11: The y-axis displays the best test score (determined over all epochs) on Cifar-10 averaged over five runs, and the bars indicate the standard deviations. The transparent area indicates the range from the minimum to the maximum. Each diamond represents one network having a specific number of parameters (in thousands) denoted on the x-axis. On the left, the solid black line shows the baseline ResNet results with 20, 33, 44, and 56 layers, and the solid green line shows the results for the corresponding E-ResNets-R. On the right, the solid black line shows the baseline PyrNet, and the solid green line the results for the E-PyrNet-R.

4.3.2 Results

The results of the Cifar-10 experiments are shown in Figure 4.11: the left side compares the original ResNet to the E-ResNet-R, and the right side compares the PyrNet to the E-PyrNet-R. Each point of the two curves shows the best possible test error occurring over all training epochs averaged over five runs and for one particular network (i.e., one particular number of blocks). Note that the inclusion of E-blocks-R reduces the number of parameters. Overall, the E-nets-R are more compact and perform better with a lower test error and only a small overlap in the standard deviations.

Table 4.11 shows the results on ImageNet. Note that the E-ResNet-50-R ($q = 1$) performs better than the baseline ResNet-50; the validation error is reduced by almost 0.4. When considering the already compact MobileNet-V2 architecture, the E-MobileNet-V2-R performs better than the MobileNet with an error decreased by 0.2 without substantially increasing the number of parameters. We trained the MobileNet-V2 baseline network ourselves to obtain its validation error. For the ResNet-50, we report the value from the Tensorpack repository [Wu+16]. The ResNet-50 and E-ResNets-50-R performances, depending on the number of parameters, are illustrated in Figure 4.12.

Table 4.11: ImageNet validation errors for different E-nets-R and baselines: we transformed two baseline network architectures, the ResNet-50, and the MobileNet-V2, into E-nets-R here, denoted as E-ResNet-50-R and E-MobileNet-V2-R. The transformations are done by substituting specific blocks of the baseline networks with an E-block-R (see text). Additionally, by choosing different expansion factors q, we created an E-ResNet-50-R that is smaller than the baseline ($q = 0.8$) and one larger network ($q = 1$).

Model	Num. parameters (M)	Val. error
ResNet-50 (baseline)	25.5	23.61
E-ResNet-50-R (q=0.8)	24.3	23.80
E-ResNet-50-R (q=1)	25.9	23.24
MobileNet-V2 (baseline)	3.5	28.71
E-MobileNet-V2-R	3.6	28.53

4.3.3 Discussion

With the E-net-R, we presented an updated version of the E-net-I, focusing more on improving the performance of SOTA models instead of making them more parameter-efficient. We restructured the normalization step, making it less susceptible to a high input variance and outliers (see Section 3.7 on page 36 for more information). Furthermore, we used ReLU non-linearities to increase the selectivity of E-neurons. Finally, instead of creating entire stacks of E-blocks-R, we showed that mixing E-blocks-R and baseline blocks is a more appropriate design rule. With these changes, we improved the classification performance of several different architectures while – if ever – only slightly increasing the number of parameters.

However, Figure 4.11 shows that we also trained E-nets-R that had fewer blocks while outperforming larger baselines (for example, the E-PyrNet-44 having a lower test error than the PyrNet-56). Thus, also on Cifar-10, our initial hypothesis is confirmed that the use of end-stopped neurons can yield more efficient networks that use fewer operations.

Again, these results show how inspiration from biological vision can help to improve CNNs with the proposed simple design rule being of practical use: many CNN architectures can be transformed into E-nets-R quickly and easily. As before, we used no additional hyperparameter tuning.

4.4 E-nets and visual coding

In this Section, we analyze E-nets from a more biological point of view. We show that E-neurons-R, i.e., AND combined filter-pairs, are end-stopped to different degrees, depending mainly on the angle between the two filter vectors. In Section 3.7.1, we

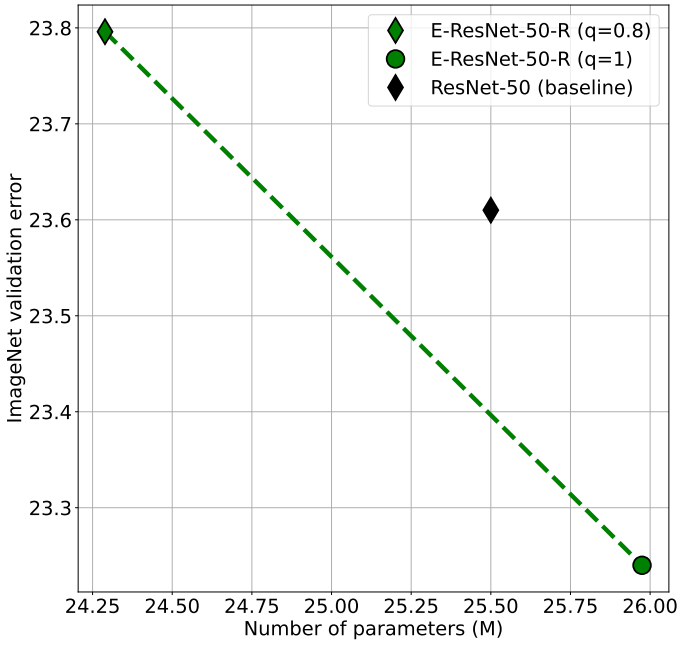


Figure 4.12: Number of parameters vs. ImageNet validation error for the ResNet-50 (black diamond) and two E-ResNets-R with different expansion factors q.

investigated the hyperselectivity [VF17] of E-neurons, showing that with a greater angle γ between the two filter vectors of an E-neuron, the hyperselectivity increased. Hyperselectivity is found in biological neurons and may lead to greater robustness against noisy input signals [Pai+20]. Indeed, we will show that E-nets-R are more robust against particular adversarial attacks and compression artifacts.

4.4.1 Entropy and degree of end-stopping

To further support the view that E-neurons-R are hyperselective depending on γ, we analyzed the entropy of the feature maps generated by different E-neurons-R. The results in Figure 4.13 show that the learned filters tend to have a γ larger than zero, i.e., the majority of E-neurons-R are hyperselective and that a high γ-value leads to lower entropy. This result was to be expected because the curvature of the exo-origin iso-response contour increases with γ, and so does the hyperselectivity. Details of how the entropy is computed are given in Section A.7.

To analyze the end-stopping behavior of the model neurons learned in the E-nets-R trained on Cifar-10 and ImageNet, we needed to quantify the degree of end-stopping.

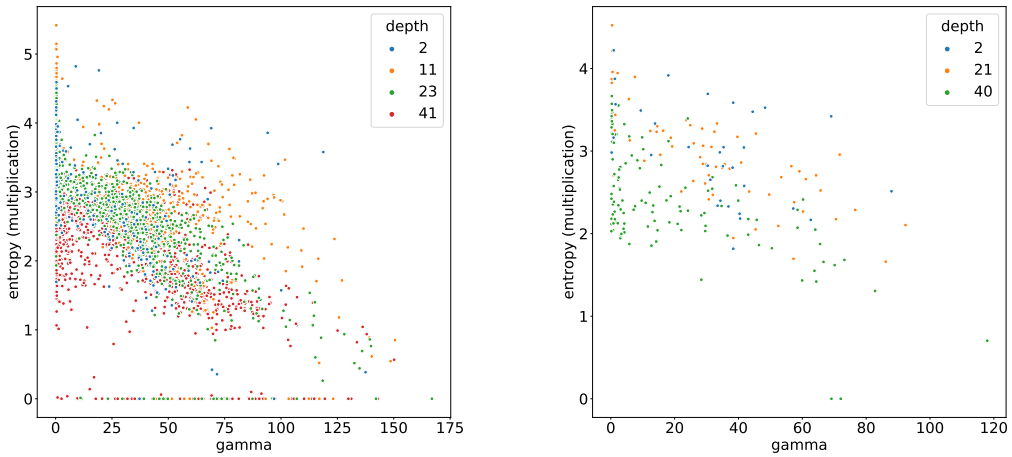
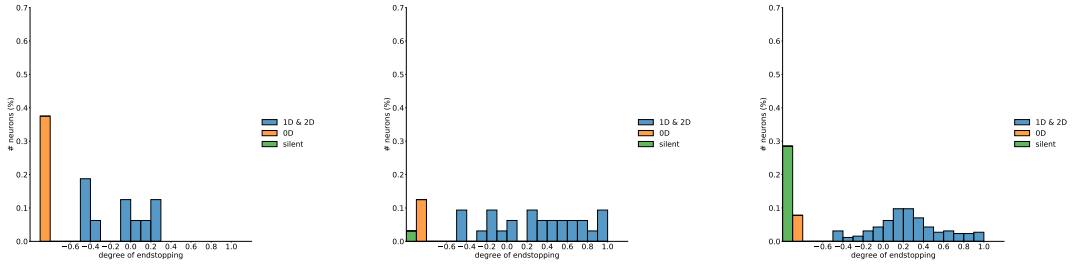


Figure 4.13: Scatter plots of entropy over hyperselectivity (indicated by the angle γ): each dot corresponds to an E-neuron-R. The color codes indicate the position of each neuron in the network, i.e., the number of convolution layers. The entropy of a particular E-neuron's feature map is estimated as described in Section A.7 and plotted against $\gamma = \measuredangle(\mathbf{v}, \mathbf{g})$. The left panel shows results for the E-ResNet-50-R trained on ImageNet with neuron positions 2, 11, 23, and 41. The right panel shows results for the E-ResNet-56-R trained on Cifar-10 and neuron positions 2, 21, and 40. Note the correlation between entropy and γ. Hyperselectivity is directly linked to γ as illustrated in Figure 3.6 on page 42.

To relate to physiological measurements, we started by analyzing the response of E-neurons-R to straight lines and line ends, but this turned out to be problematic because the E-nets use small 3×3 filters and sub-sample the input. To keep the analogy but with a more robust measure, we used a square as input and quantified the average responses to the uniform 0D regions, the straight 1D edges, and the 2D corners. The degree of end-stopping is then defined by the relation between 1D and 2D responses. To account for ON/OFF type responses, we used both a bright and a dark square. The results are shown in Figure 4.14, and the algorithm details are given in Section A.8.

Note that, similar to the actual neurons in cortical areas V1 and V2, the model neurons in the E-net-R are end-stopped to different degrees. Thus, end-stopping seems to be beneficial for both the ImageNet and Cifar-10 tasks since the emergence of end-stopping is here driven by the classification error. As expected, the multiplication in the E-block-R shifts the distribution towards a higher degree of end-stopping. However, the network could have learned filter pairs that do not lead to end-stopped E-neurons-R. The bias we introduce, i.e., multiplying filter pairs, simply makes it easier for the network to learn end-stopped representations.



(a) ReLU after 1st conv. PyrBlock (Cifar-10)

(b) $\mathbf{T}_2$ E-block-R (Cifar-10)

(c) $\mathbf{T}_2$ E-block-R (ImageNet)

Figure 4.14: Distribution of neurons plotted over the degree of end-stopping: distributions are shown for the first block of the first stack for different models. The left image shows results for the neurons of the first convolution, after BN and ReLU, of a pyramid block in a PyrNet-56. Middle: E-neurons-R ($\mathbf{T}_2$, see Equation 3.10 on page 36) of an E-PyrNet-56-R trained on Cifar-10. Right: $\mathbf{T}_2$ of an E-block-R for the E-ResNet-50-R trained on ImageNet. Blue bars show normalized histograms for the ratio $1 - \frac{1D}{2D}$ that quantifies the relation between responses to straight edges (1D) and corners (2D), see Section A.8. Neurons that responded to 0D regions (the center of a square) are excluded from the blue histogram and shown separately as orange bars. Neurons that do not respond at all (i.e., all 0D, 1D, and 2D responses are zero) are also excluded from the blue histogram and are shown as green bars.

The angle distributions in Figure 4.15 show that linear E-neurons-R are also learned since more than 15% of E-neurons-R have a γ-value near zero. I.e., these neurons are linear neurons. With increasing network depth, the number of linear E-neurons-R increases, indicating that hyperselectivity, and especially end-stopping, are more frequent in earlier stages of the visual processing chain.

4.4.2 Example E-units

As we have shown, the learned E-neurons are hyperselective and end-stopped to different degrees. However, these two properties do not fully specify an E-neuron. When analyzing the individual E-neurons in more detail, it is difficult to further specify them according to simple properties such as orientation or phase. Nevertheless, some E-neurons look as if they were taken from a textbook on 'how to model end-stopped neurons', and we show one example in Figure 4.16 on page 77.

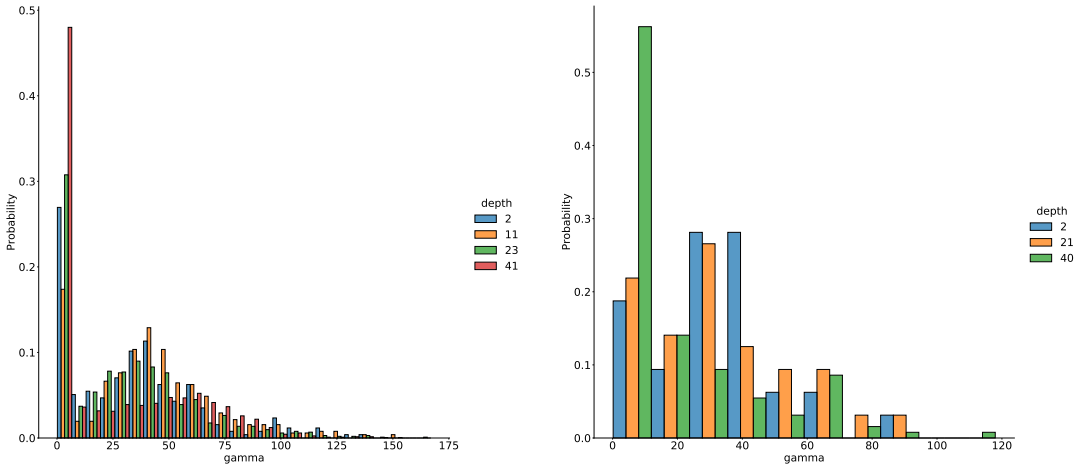


Figure 4.15: Distribution of E-neurons-R ($\mathbf{T}_2$) as a function of hyperselectivity (indicated by the angle $\gamma = \measuredangle(\mathbf{v}, \mathbf{g})$) and for different positions in the network: note that most neurons are hyperselective to different degrees and that hyperselectivity is reduced later in the network. The left panel shows results for the E-ResNet-50-R trained on ImageNet with neuron positions 2, 11, 23, and 41 (i.e., the number of convolution layers). The right panel shows results for the E-ResNet-56-R trained on Cifar-10 with neuron positions 2, 21, and 40.

4.4.3 E-neurons-R and robustness

4.4.3.1 Adversarial attacks

Although outperforming many alternative approaches on various vision tasks, CNNs are surprisingly sensitive to barely visible perturbations of the input images [Sze+14]. An adversarial attack on a classifier function f adds a noise pattern η to an input image $\mathbf{x}$ so that $f(\mathbf{x} + \eta)$ does not return the correct class $y = f(\mathbf{x})$. Furthermore, the attacker ensures that some p-norm of η does not exceed ϵ. In many cases, including this work, the infinity-norm is chosen, and the ϵ values are in the set $\{1/255, 2/255, ...\}$. Thus, for example, for $\epsilon = 1/255$, each 8-bit pixel value is at most altered by adding or subtracting the value one.

Goodfellow, Shlens, and Szegedy [GSS14] argue that the main reason for the sensitivity to adversarial examples (the term 'example' referring to the input $\mathbf{x} + \eta$) is due to the linearity of CNNs: with a high-dimensional input, one can substantially change a LN-neuron's output, even with small perturbations. Consider the output of an LN-neuron $f_{LN}(\eta)$ for an input perturbation η with dimension n. We choose η to be the sign function of the weight vector multiplied with ϵ: $\eta = sign(\mathbf{w}) \cdot \epsilon$. Thus, η roughly points in the direction of the optimal stimulus (which is also the gradient), but its infinity-norm does not exceed ϵ. Assuming that the mean absolute value of $\mathbf{w}$ is m, $f_{LN}(\eta)$ is approximately equal to ϵnm. Accordingly, a significant change in

the LN-neuron's output can be achieved by a small ϵ value if the input dimension n is large, which is usually the case for many vision-related tasks.

This gradient-ascent method can also be applied to non-linear neurons or sequences of neurons because, for a point, almost any function f can be linearly approximated in a sufficiently small region around that point. According to the approximation, the input needs to be moved along the positive gradient direction to increase the output optimally. The fast gradient sign method (FGSM) [GSS14] perturbs the original input image $\mathbf{x}$ by adding $\eta = \epsilon sign(\nabla f(\mathbf{x}))$. Another approach is to define η to be the gradient times a positive step size τ followed by clipping to $\eta \in [-\epsilon, +\epsilon]^n$. The clipped iterative gradient ascent (CIGA) greedily moves along the direction of the highest linear increase,

$$\begin{aligned} \eta_0 &= \mathbf{0} \\ \mathbf{q}_{i+1} &= \eta_i + \tau \nabla f(\mathbf{x} + \eta_i); \tau > 0 \\ \eta_{i+1}^j &= \min(\max(q_{i+1}^j, -\epsilon), \epsilon). \end{aligned} \tag{4.2}$$

with q_i^j being the j-th entry of the unbounded result $\mathbf{q}_i$ at the i-th iteration step. In the following, we use the CIGA to illustrate the connection between hyperselectivity and adversarial attacks. In our experiments, we employ the FGSM as it is a widely recognized adversarial attack method.

Regarding an iso-response contour plot, one can easily spot the direction of the gradient, which is orthogonal to an iso-response contour [Pai+20]. In Figure 4.17 on page 78, the left panel shows the gradient for an LN-neuron being parallel to the optimal stimulus (black line). As long as the initial input yields a non-zero gradient, each step of the CIGA maximally increases the LN-neuron output. Thus, the algorithm's effectiveness is only bounded by ϵ but widely independent of the initial input $\mathbf{x}$. For a step size larger than ϵ, CIGA finds the optimal solution in one step.

We now investigate the effects of the CIGA on a simplified version of an E-neuron-R [9]:

$$F(\mathbf{x}) = \mathbf{x}^T \mathbf{v} \mathbf{g}^T \mathbf{x}. \tag{4.3}$$

Note that in the following particular example, the input is chosen to yield non-negative projections on $\mathbf{v}$ and $\mathbf{g}$; thus, we can remove the ReLUs. The resulting gradient is:

$$\nabla_F(x) = (\mathbf{v}^T \mathbf{x})\mathbf{g} + (\mathbf{g}^T \mathbf{x})\mathbf{v}. \tag{4.4}$$

The effectiveness of an iteration step strongly depends on the current position. The highest possible increase would be obtained along the line defined by the optimal stimulus. In Figure 4.17 on the right, this is the black line. If the initial input is located on this line, any step in the gradient direction yields an optimal increase of

[9] We here argue that hyperselectivity leads to robustness against adversarial attacks. Thus, although the following example focuses on E-neurons-R, the reasoning can be applied to all E-neurons.

the E-neuron output. However, for any other position with a non-zero gradient, an unbounded iteration step would move toward the optimal stimulus line. The blue curve in Figure 4.17 shows the path for several iterations of the CIGA: starting above the optimal stimulus line, each step slowly converges to the optimal stimulus line, eventually moving almost parallel to it. Once the ϵ threshold of 1 is reached in the horizontal dimension, the (now bounded) path runs parallel to the vertical dimension to increase the neuron output further. The optimal solution is found once the ϵ bound is also reached in the vertical dimension.

The crucial difference when comparing E-neurons-R with LN-neurons is that for an E-neuron, there are numerous conditions (depending on τ, $\mathbf{x}$, γ, and ϵ) where a CIGA would need several steps to find an optimal solution. This reduced effectiveness of the CIGA illustrates why hyperselective neurons are more robust against adversarial attacks: e.g., if $\mathbf{x}$'s projection on $\mathbf{v}$ and $\mathbf{g}$ is far away from the optimal stimulus line, or τ is chosen poorly, or the attack is computed with too few iterations, an adversarial example might not increase the E-neuron output by much. Note that single neurons are usually not the target of adversarial attacks; instead, the gradient is determined by the classification loss function. Still, the argument holds that hyperselective neurons are harder to activate than LN-neurons which may result in increased robustness.

To test this hypothesis, we created new Cifar-10 test sets $\mathcal{S}_{\epsilon_i} = \{FGSM(\mathbf{x}, \epsilon_i) : \mathbf{x} \in \mathcal{X}_{C10}\}$ derived from the original test set $\mathcal{X}_{C10}$. Here, we focused on the most subtle adversarial attacks: we created one test set $\mathcal{S}_{1/255}$, where each test image was perturbed by using FGSM with $\epsilon = 1/255$. Results for larger ϵ-values are shown in Section B.1.1 (see Table B.1 and Table B.2). To exclude the hypothesis that the better accuracy (with perturbations) is because the E-nets-R already generalize better, we present results where we measure the percentage of changed predictions (POCP) of the classifier f:

$$\mathrm{POCP}(f, \Gamma, \theta) = \frac{100}{|\mathcal{X}_{C10}|} \sum_{\mathbf{x} \in \mathcal{X}_{C10}} \mathbb{1}(f(\mathbf{x}) \neq f(\Gamma(\mathbf{x}, \theta))); \tag{4.5}$$

$\mathbb{1}$ being the indicator function returning a one for a true statement and a zero else. Γ is some function (here, FGSM) that perturbs the original image $\mathbf{x}$ based on some parameter θ. We evaluated this metric for each of the two architectures we trained on the original Cifar-10 training set (see Section 4.3.1); no additional adversarial training scheme was employed. As shown in Figure 4.18 on page 79, 40% to 50% of the predictions did change. However, for both baseline models, substituting some of the LN-neurons with E-neurons-R increased the robustness against FGSM attacks.

4.4.3.2 JPEG compression

The results on adversarial attacks reiterate that CNN predictions can be significantly altered by deliberate and subtle attacks (we show some examples in Figure A.1 in the Appendix). Unfortunately, this lack of robustness creates problems of practical

relevance beyond such attacks. For example, JPEG compression can create artifacts that have similar effects. To evaluate robustness against JPEG artifacts, we created the Cifar-10 test sets $\mathcal{S}_{Q_i} = \{JPEG(\mathbf{x}, Q_i) : \mathbf{x} \in \mathcal{X}_{C10}\}$; with $JPEG(\mathbf{x}, Q)$ being the JPEG-compressed version of the original image $\mathbf{x}$ with a quality rate $Q \in \{1, 2, ..., 100\}$, 100 being the original image. A low quality indicates a high compression with stronger artifacts (example images are given in Figure A.2 in the Appendix). In Figure 4.19, we show the results for the low compression test set $\mathcal{S}_{90}$ and other results in Section B.1.2.

In addition to adversarial attacks, using E-neurons-R increased the robustness against JPEG artifacts. However, even a moderate compression alters up to ten percent of the CNNs' predictions.

4.4.4 Discussion

As the inspiration for E-nets comes from biological vision, we have analyzed the selectivity of E-units-R to relate them to what is known about visual neurons. We could show that E-units-R are indeed end-stopped to different degrees. The emergence of end-stopping in a network that learns based on only the classification error demonstrates that end-stopping benefits image classification. The entropy analysis shows that with increasing selectivity, the entropy of a feature map is reduced. This result follows statistical findings that 2D regions are rare in natural images [ZBW93], the statistical finding being one of our key motivations to model end-stopped cells.

Of course, the considerations above cannot be taken to imply that biological vision implements an E-net architecture, especially as the E-nets implement additional and typical deep-network operations such as linear recombinations that increase the entropy of the representation. In other words, much of what well-performing deep networks do is not something one would necessarily consider to be optimal.

It is known that sparse-coding units are more selective than typical CNN units, i.e., more selective than linear neurons with point-wise non-linearities. Thus, sparse-coding units are less prone to certain adversarial attacks [Pai+20]. This hyperselectivity has been quantified with the curvature of the iso-response contours. We could show that the iso-response contours of E-units-R are curved, with the degree of curvature depending on the angle between the multiplied feature vectors (see Section 3.7.1 on page 37), and that a large number of hyperselective units emerge in E-nets-R trained for object recognition. Furthermore, our results show that E-nets-R are more robust against adversarial attacks and compression artifacts, which is, again, due to the vision-inspired E-neurons-R.

In the following, we will focus on alternative formulations of the AND combination using the (i) minimum operation and (ii) the logarithm. First, using the minimum operation instead of multiplication can be faster (E-net-M), and the E-neuron-M output only grows linearly instead of quadratically when increasing the optimal stimulus, which can facilitate optimization. Alternatively, we propose using the logarithmic

function to efficiently multiply more than two filters which could further increase the hyperselectivity of a neuron (E-net-L) because such a neuron could have exo-origin iso-response contours in several directions.

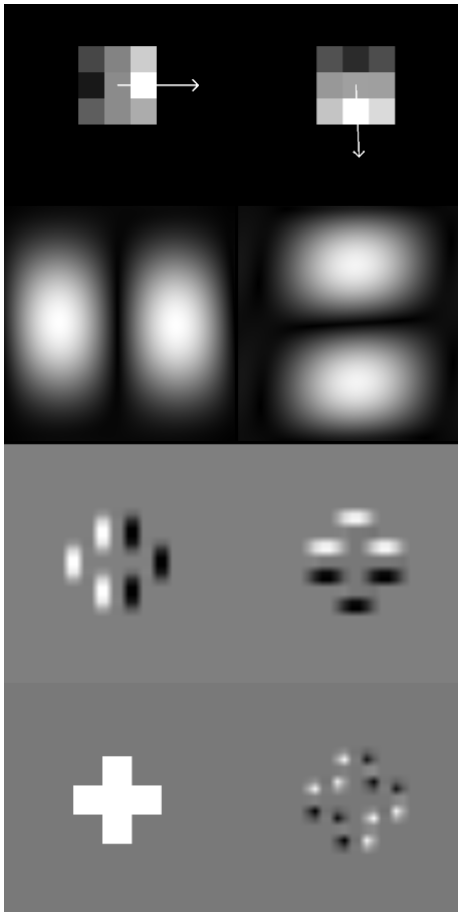


Figure 4.16: An example of learned filter pairs: the top row shows the two learned 3×3 filters (arrows indicate orientation), and the row below the corresponding Fourier spectra. The third row depicts the responses of the two filters to the image shown bottom left (image used as $\mathbf{T}_1$ to illustrate the selectivity of the E-unit). The bottom-right panel shows the response of the E-unit (the product of the filter responses). Such textbook units, however, are relatively rare. This particular unit has emerged in the E-ResNet-56-I trained on Cifar-10.

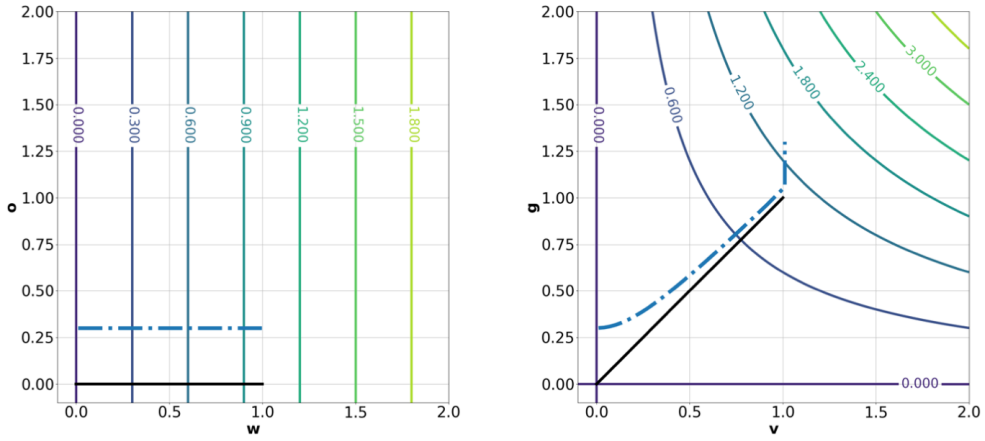


Figure 4.17: Iso-response contours and iteration path of the CIGA for left, an LN-neuron and right, an E-neuron-R: the LN-neuron's weight vector is $\mathbf{w} = (1, 0)^T$ and $\mathbf{o} = (0, 1)^T$ is an orthogonal vector. The E-neuron's filter-pair is $\mathbf{v} = (1, 0)^T$ and $\mathbf{g} = (0, 1)^T$. The black solid lines point in the directions of the respective optimal stimulus. The blue dashed lines show the respective iteration path of the CIGA (see Equation 4.2). All other colored solid lines show iso-response contours; the number on each line shows the function value of the contour. For each neuron, CIGA aims to find a perturbation η with $\|\eta\|_\infty \leq \epsilon = 1$ that maximally increases the output $f(\mathbf{x}+\eta)$. $\mathbf{x} = (0., 0.3)^T$ is the initial input to the neurons; $\tau = 0.001$ is the step size, and we computed a total of 10000 iterations. The CIGA quickly finds an optimal solution for the LN-neuron since any step along the positive gradient (parallel to the optimal stimulus, orthogonal to the iso-response contours) optimally increases the function value. For the E-neuron, the iteration path moves towards the optimal stimulus, then almost parallel to it, and finally moves upwards once the bound on ϵ is reached along the $\mathbf{v}$-axis. This longer, more complex optimization path shows that CIGA is less effective for a hyperselective E-neuron, indicating that E-neurons are more robust against adversarial attacks.

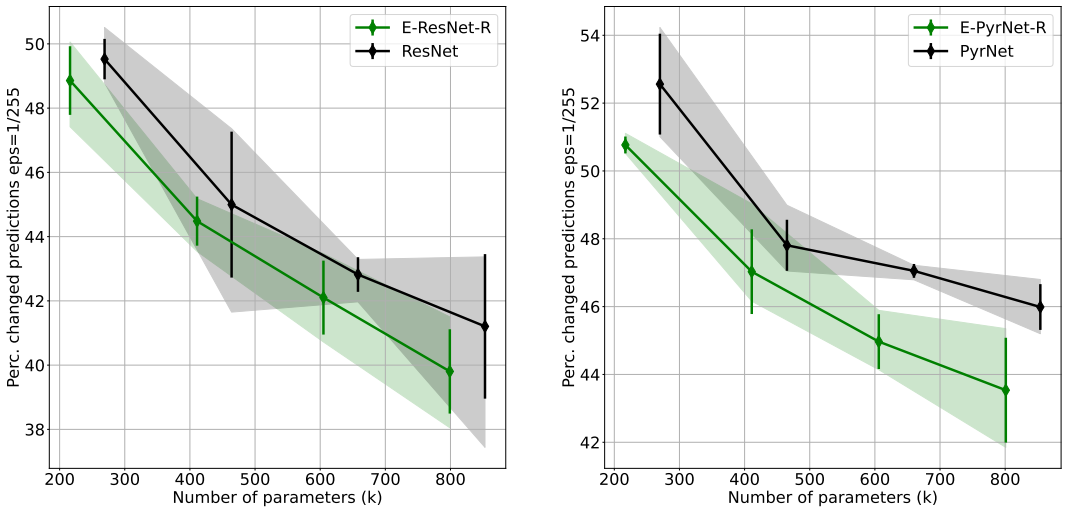


Figure 4.18: POCP (see Equation 4.5) on the adversarial example test-set $\mathcal{S}_{1/255}$: a lower value indicates that a network is more robust against attacks created by the FGSM.

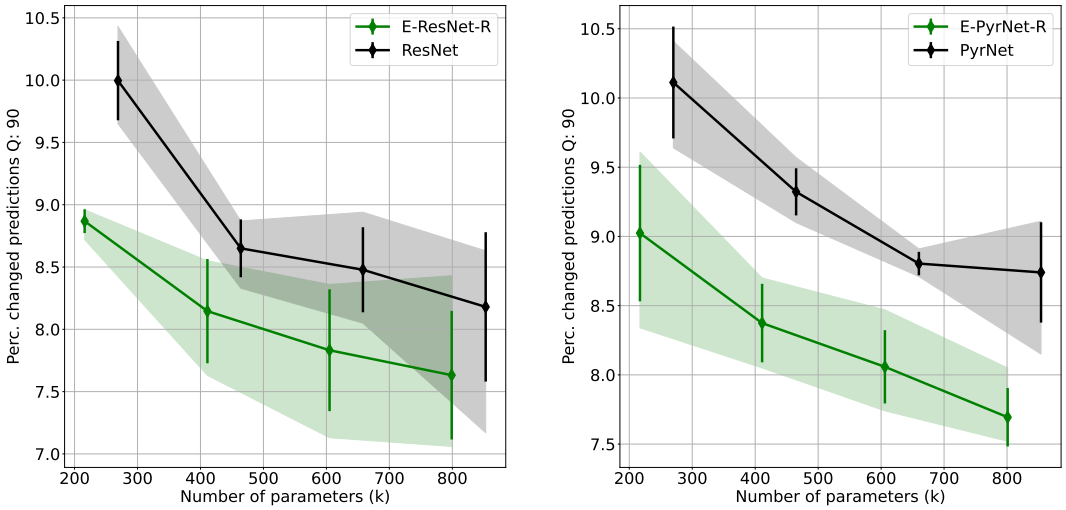


Figure 4.19: POCP for the JPEG-compressed Cifar-10 test set $\mathcal{S}_{90}$: a lower value indicates that a network is more robust against JPEG artifacts.

4.5 E-nets-M for image classification

For the E-net-I and E-net-R, AND combinations were based on multiplications. Thus, judging from the previous results, the pairwise multiplications may be the main reason for E-nets outperforming their respective baselines. For example, one explanation could be that pairwise multiplications resemble a polynomial kernel method, the primary motivation behind factorized bilinear (FB)-nets (see Section 3.6.1). However, in this section, we aim to show that the logical AND combination, being essential for end-stopped neurons, is the main contributor to the performance of E-nets. To this end, we substitute the multiplication with another valid logical AND combination: the minimum operation.

To enable a comparison to E-nets-R, we replicate a subset of the previous Cifar-10 experiments with E-nets-M. Furthermore, we investigate the effect of our design rule on the DenseNet (see Section 2.3.4), another well-known SOTA CNN.

On a side note, E-nets-M can compute results faster since computing the element-wise minimum operation of two tensors takes slightly less time than multiplying them. Comparing the time needed for a forward pass with an E-block-M to the time needed with an E-block-R shows that the E-block-M is faster, but not by a significant amount (see Table B.5 in the Appendix).

4.5.1 Experiments

Similar to E-nets-R, we hypothesize that E-nets-M introduce a practical inductive bias to a CNN architecture: the minimum of a learned filter-pair, an E-neuron-M, explicitly models end-stopped neurons and leads to efficient representations. Furthermore, unlike traditional model neurons, E-neurons-M are hyperselective, a property that should make them more robust, e.g., to compression artifacts. We have run two experiments to investigate these conjectures.

4.5.1.1 Classification performance on Cifar-10

We compared models with different depths created from two reference network architectures: the PyrNet and the DenseNet. For each model, we created an E-net-M version using the previous design rule: *substitute the first block of each stack with an E-block-M.* We compared models with different depths, defined by the number of blocks per stack. For the PyrNets, we created models with $N \in \{3, 5, 7, 9\}$ blocks (for more information, see Section 4.3.1.1). We trained DenseNets with $N \in \{3, 9, 16\}$ blocks and a growth-rate $k = 12$, corresponding to networks with $(L = 22, k = 12)$, $(L = 58, k = 12)$, and $(L = 100, k = 12)$ in the DenseNet paper's naming convention. For all E-blocks-M, we set the expansion factor $q = 2$, and we trained and tested all models on Cifar-10 with a standard procedure for data augmentation (see Section A.2.1). Each net was trained for five random seeds, altering the training batch order and random initialization. Details

on training hyperparameters are given in Section A.2.2. Note that we did not employ an early stopping strategy but instead report the final test results after the last training epoch.

4.5.1.2 Robustness to compression artifacts

Since E-nets-M have neurons that are hyperselective, one can expect the networks to be more robust against perturbations. To evaluate robustness, each model trained in the classification experiment was tested on JPEG-compressed versions of the Cifar-10 test set $\mathcal{X}_{C10}$. We created nine sets $\mathcal{S}_Q = \{JPEG(\mathbf{I}, Q) : \mathbf{I} \in \mathcal{X}_{C10}\}$ with quality $Q \in \{90, 80, ..., 10\}$, $JPEG(\mathbf{I}, Q)$ being the compression function for an input image $\mathbf{I}$. For $Q = 100$, the original image is returned (no compression); $Q = 1$ would yield maximal compression with images that are barely recognizable (see Figure A.2 on page 148 for examples). As a robustness metric, we report the POCP (see Equation 4.5 on page 74).

4.5.2 Results

The Cifar-10 test errors of each model are shown in Figure 4.20. Note that, in all configurations, the E-nets-M had a lower mean test error than the corresponding baseline networks while using fewer parameters.

Next, we analyze the robustness of the networks. Figure 4.21 shows the POCP for the quality levels $Q \in \{90, 80, 70, 60\}$. Compared to the PyrNet and DenseNet, the E-nets-M were less likely to change their output when presented with compressed images. However, even the more robust E-nets-M remained sensitive to compression artifacts given that 8% of the predictions changed with a slightly altered test set (such as $\mathcal{S}_{90}$). See the Appendix (Section B.2) for additional results.

Comparing the E-net-M to the E-net-R Figure 4.22 compares the Cifar-10 results for the E-PyrNets-M (see Section 4.5.1.1) and E-PyrNets-R (see Section 4.3.1.1). We report the mean test error after the last training epoch. The two experiments have two minor differences: they were trained on different seeds, and slightly different normalization values were used during data augmentation (ImageNet mean and standard deviation vs. Cifar-10 mean and standard deviation). However, the effect of both differences is negligible. The results show that the E-net-M slightly outperforms the E-net-R, yielding better test scores for smaller models and having a smaller standard deviation.

4.5.3 Discussion

The results show that the presented E-nets-M behave similarly to E-nets-R: again, we could improve well-established CNNs in terms of robustness against JPEG artifacts

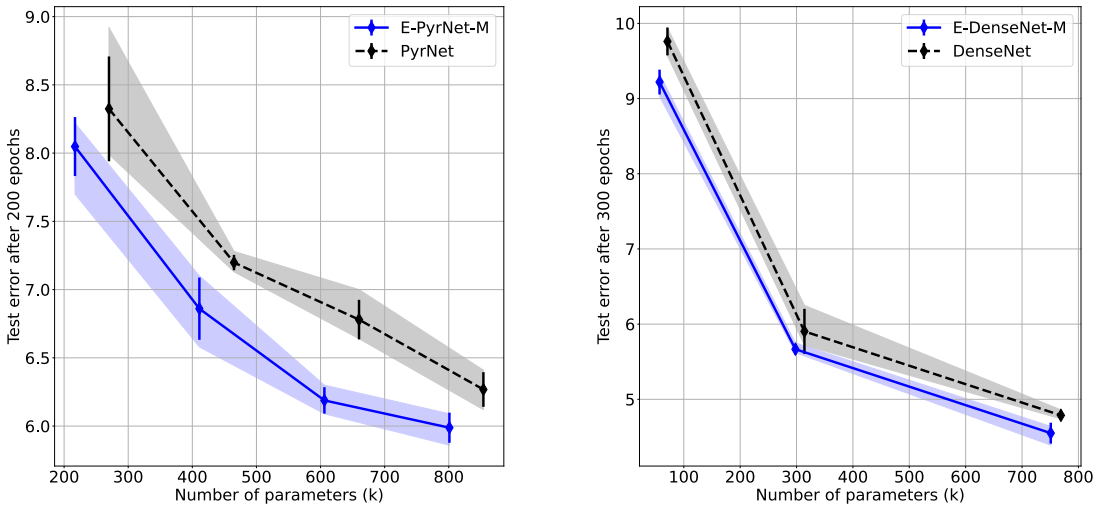
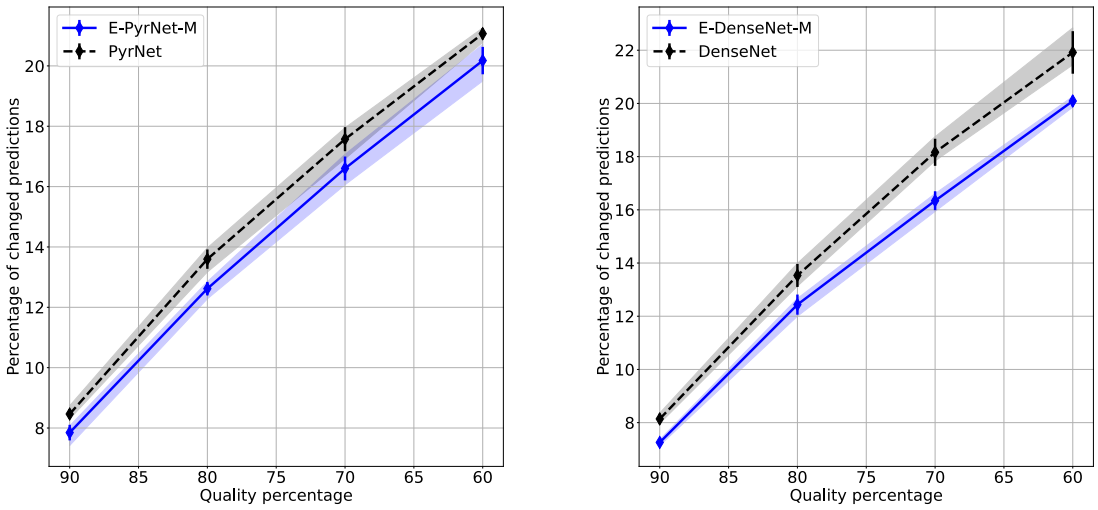


Figure 4.20: Cifar-10 test error results: black dashed lines show the baseline results, solid blue lines the E-net-M results. Each diamond denotes the mean error of a model with a specific number of blocks. The error bars indicate the standard deviation of each model. The colored areas show the distance between the minimum- and maximum values. The x-axis shows the number of parameters (in steps of thousands), increasing with the number of blocks. The y-axis is the test error after 200 and 300 epochs for the PyrNets (a) and DenseNets (b), respectively.

and performance on Cifar-10. In addition, our approach of creating an E-net-M from a standard CNN also worked for a DenseNet. Considering that the DenseNet's structure is quite different from a ResNet and a MobileNet-V2, this result indicates that our approach may apply to various CNNs.

Comparing the Cifar-10 classification results (see Figure 4.22), the E-PyrNet-M yields better results than the E-PyrNet-R by a small margin, maybe due to having more linear 2D region detectors (see Section 3.8.1 on page 43). Furthermore, from a practitioner's point of view, the E-net-M offers a slightly increased computation speed, making the E-net-M the preferable architecture over the E-net-R.

Lastly, we think the results show that efficient coding and hyperselectivity of end-stopped neurons are feasible concepts to improve CNNs. However, how exactly an end-stopped neuron is implemented – e.g., the AND combination being a multiplication or a minimum operation – is of lesser importance.



(a) PyrNet (b) DenseNet

Figure 4.21: POCP depending on JPEG quality: the x-axis shows the quality parameter Q for the JPEG compression algorithm (which can vary from 100 to 1). The y-axis denotes, in relative terms, how many predictions were different from the original predictions (with $Q = 100$) when using a more compressed test set.

4.6 E-nets-L for image classification

This Section presents E-nets-L as an alternative to E-nets-I and E-nets-R. Instead of explicitly computing the element-wise multiplication of filter outputs, E-nets-L use convolutions in *log-space* that can indirectly compute large products. Following our usual terminology, E-nets-L are CNNs that contain one or several E-blocks-L or the smaller base structure – called E-layer-L.

So far, in this thesis, we presented the applications, experiments, and results in chronological order, i.e., the E-net-I publications preceded the E-net-R and the E-net-M publications. In contrast, the results of this section are taken from our first publication on the topic of AND combinations for CNNs [GMB20b]. When writing the paper, we focused on creating networks that only consist of E-blocks-L. However, as more recent results show, having stacks that mix E-blocks with standard convolution blocks yields E-nets that perform better. Thus, in hindsight, we would have designed some experiments differently. Nevertheless, a precursor of a network employing this mixture is presented in Subsection 4.6.2.3. There, we substitute the first four convolution blocks of a SimpleNet [Has+16] with a specific E-block-L. Furthermore, note that the proposed indirect product (the E-layer-L) could be easily used as a ζ function for the general E-block (see Section 3.5 on page 31).

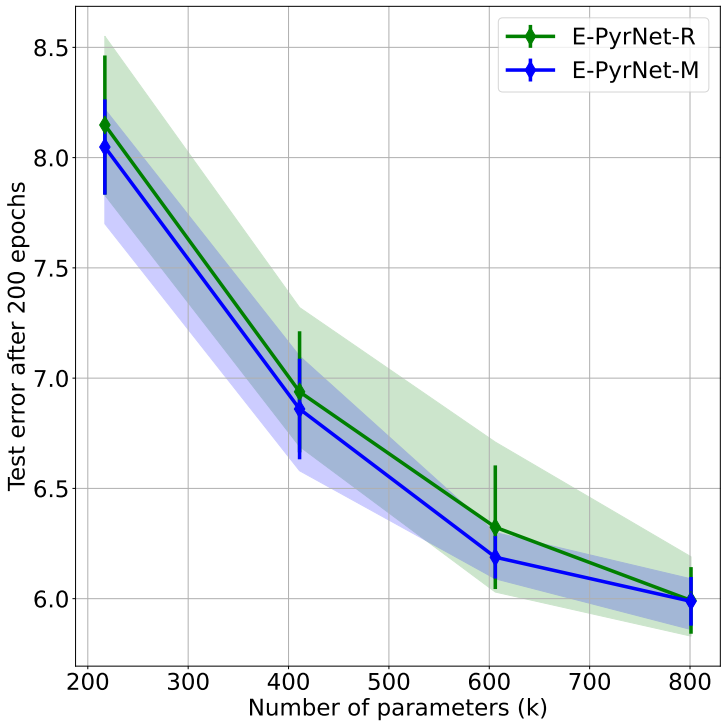


Figure 4.22: Comparing the Cifar-10 test error after the last training epoch for the E-PyrNets-M (blue) and E-PyrNets-R (green).

4.6.1 Methods

4.6.1.1 Convolution in log-Space

Our approach makes use of the logarithm's product rule:

$$\alpha log(x) + \beta log(y) = log(x^{\alpha} \cdot y^{\beta}). \tag{4.6}$$

Accordingly, if we compute the logarithm of an input tensor $\mathbf{T}_{in} \in \mathbb{R}^{h \times w \times d_0}$ and filter it with a kernel $\mathbf{W} \in \mathbb{R}^{k \times k \times d_0 \times d_1}$, we essentially compute a large product with the elements of $\mathbf{T}_{in}$ weighted exponentially by the elements of $\mathbf{W}$. Thus, we compute the logarithm of the (i, j)-th pixel of the m-th output feature map of $\mathbf{T}_{out}$ using:

$$log(\mathbf{T}_{out}[i, j, m]) = log(\prod_{a,b,n} \mathbf{T}_{in}[i + a, j + b, n]^{\mathbf{W}[a,b,n,m]}) \tag{4.7}$$

$$= \sum_{a,b,n} log(\mathbf{T}_{in}[i + a, j + b, n])\mathbf{W}[a, b, n, m]; \tag{4.8}$$

with $a, b \in \{\lfloor -k/2 \rfloor, ..., \lfloor k/2 \rfloor\}$, $i \in \{1, ..., h\}$, $j \in \{1, ..., w\}$, $n \in \{1, ..., d_0\}$, and $m \in \{1, ..., d_1\}$. d_0 is the input, and d_1 is the output dimension of the tensor, i.e., the

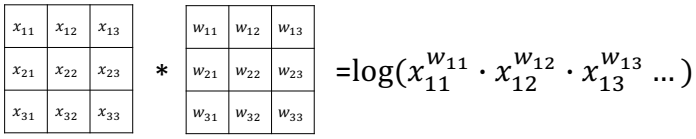


Figure 4.23: Convolution in log-space Example 1: a multiplicative combination of pixel values. The input signal $(x_{11}, x_{12}, ...)$ is first transformed into log-space. Subsequently, it is convolved with the filter on the right. Equation 4.6 shows that a large product is computed, which can be transformed back to the original input space by exponentiation. This approach allows for highly non-linear activations and sparse signals since, for example, single-pixel values can inhibit the entire output.

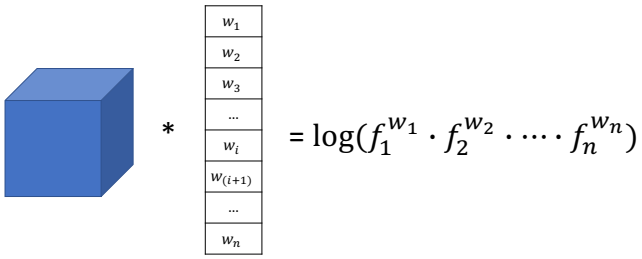


Figure 4.24: Convolution in log-space Example 2: large element-wise AND combinations of feature maps can be realized via 1×1 convolutions. As in the example of Figure 4.23, the input tensor (blue cube) of shape $h \times w \times n$ is transformed using the logarithm. Subsequently, a 1×1 convolution in log-space yields an element-wise product of exponentially-weighted feature maps $f_i^{w_i}, i = 1, ..., n$. This structure can model hyperselective neurons (see Section 3.7.1 on page 37) that are only excited if multiple input neurons are activated.

number of feature maps.
Figure 4.23 illustrates how particular pixel positions can suppress each other, a mechanism that a standard CNN may only realize with a sequence of many convolution layers. Figure 4.24 shows how a 1×1 convolution in log-space can implement an *orientation-selective* operation. This layer can model neurons that only produce a large output if all filters in a certain subset (e.g., several specific orientation filters) are activated. Thus, including neurons that learn the element-wise multiplication of two filters, as proposed with the E-net-I and E-net-R. In contrast to E-net-I and E-net-R, the number of filters is not restricted to two.

4.6.1.2 E-layers-L

Using highly non-linear terms such as products, logarithms, and exponentials with gradient descent and backpropagation is a challenging task. This fact becomes obvious when looking at the derivatives $\frac{d}{dx} log_2(x) = \frac{1}{x ln(2)}$ and $\frac{d}{dx} 2^x = 2^x ln(2)$, both being very volatile functions. Similar to our explicit multiplication models, we rely on scaling

operations, such as BN, to prevent vanishing and exploding values coming from large products. Furthermore, ReLUs and offsets avoid unstable training when computing the logarithm.

The left structure in Figure 4.25 presents the processing chain of an E-layer-L: first, all negative values are discarded by a ReLU, and an offset of 0.5 is added to the signal. The operation is needed because the logarithm of negative values is not defined, and the output reaches negative infinity when the input converges to 0. We use the base-2 logarithm; thus, with the proposed signal cutoff, the resulting output is in $[-1, \infty]$. As a next step, the actual convolution is computed. Hereafter, we apply BN to limit the output magnitude since exponentiation is subsequently applied. Here, we do not train any additional re-scaling weights commonly used in BN. Hence, the output has zero mean and a standard deviation of 1. After exponentiation, we apply another BN with a learned factor and offset, giving the network the capacity to re-scale the signal (for more information on BN, see Section 2.3.1).

4.6.1.3 E-block-L

In most experiments, we propose an architecture consisting only of E-blocks-L we denote as E-CustomNet-L. The E-CustomNet-L is designed to be orientation-selective; thus, the E-layers-L use 1×1 convolutions (see the example in Figure 4.24). Accordingly, each E-layer-L needs to be combined with typical *convolution blocks* (convolution, BN, and ReLU), which can model oriented filters. The suppression according to pixel positions (see example in Figure 4.23) is not modeled here. However, we evaluate E-layers-L with 3×3 convolutions in a hybrid architecture with the experiment described in Section 4.6.2.3.

4.6.1.4 Shallow E-CustomNets-L

As with other E-nets, we hypothesize that a network with E-layers-L could be just as powerful a feature extractor as a standard CNN, though needing fewer operations. However, when using a shallower E-net-L, we need to ensure that the size of the receptive field, which mainly depends on the number of convolutions and pooling operations, is large enough. Otherwise, a shallow E-net-L may not be able to cover the necessary context to encode the image effectively. To this end, we use a multi-scale structure called E-block-L, shown in Figure 4.25 on the right. The input is processed in parallel by two convolution blocks that differ in the pooling layer positions. The left block resembles a typical CNN architecture, where the convolution block is applied first, and then the output is pooled (see, for example, the VGG-16 [SZ14]). On the right side, the pooling operation is done before the convolution. Essentially, the right path of the architecture allows for increasing the receptive field by a factor of 2. After applying the E-layers-L, both paths are again mixed via a 1×1 convolution.

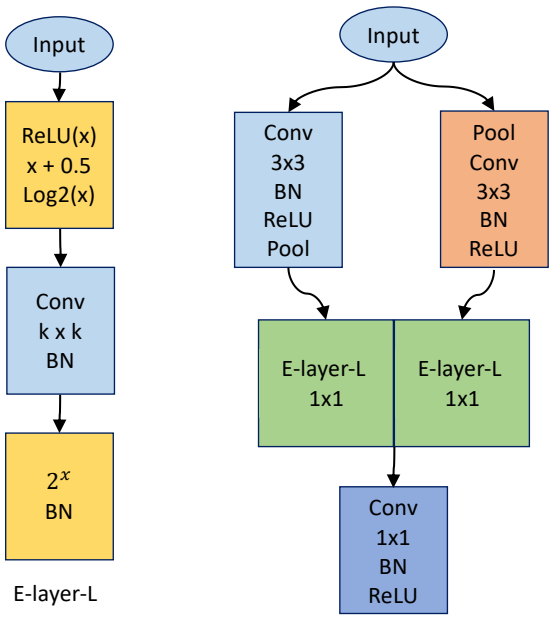


Figure 4.25: Structure of the E-layer-L (left) and E-block-L (right). Left side: before computing a convolution with kernel size k, we transform the input to the log2-space, ensuring that the minimum value is not below -1. Subsequently, the convolution is computed. Via BN, the output is normalized to zero mean and a standard deviation of 1 since large values can otherwise increase quickly in the exponentiation of the signal (2^x). Hereafter, we employ another BN at the end of the layer output, as it is commonly used in deep learning architectures. Right side: to increase the receptive field quickly, we use a parallel multi-scale structure, where one path is directly downscaled by max-pooling. The paths are later concatenated and linearly combined by a 1×1 convolution with subsequent BN and ReLU.

4.6.2 Experiments

To test the assumption that E-CustomNets-L with fewer, highly non-linear layers perform on par with deeper standard CNNs, we conducted several experiments on the ImageNet and Cifar-10 benchmarks. Additionally, we evaluated a hybrid network in Section 4.6.2.3 that uses E-blocks-L in the early layers and uses convolutions in deeper layers of the architecture.

4.6.2.1 E-CustomNet-L on ImageNet

Compared to a regular convolution, E-blocks-L do not increase the number of computations by much, and we assume that shallower networks are feasible. Accordingly, we tested a lightweight architecture on ImageNet. We used the proposed parallel architecture (see Section 4.6.1.4) to increase the receptive fields in the network. The

Table 4.12: Structure of the E-CustomNet-L trained on ImageNet: the model contained four E-blocks-L (see Figure 4.25 on the right) and a starting feature dimension of 64. Each E-block-L contained max-pooling layers, and with each E-block-L, the number of feature maps was doubled. We used a 1×1 convolution to map the output of the fourth block to the latent feature dimension 256. Subsequently, global average pooling was applied, followed by a linear layer.

Input	Layer/Block
$224^2 \times 3$	E-block-L
$112^2 \times 64$	E-block-L
$56^2 \times 128$	E-block-L
$28^2 \times 256$	E-block-L
$14^2 \times 512$	E-block-L
$7^2 \times 1024$	conv 1×1
$7^2 \times 256$	glob-avg-pool
256	linear
1000	softmax

network consisted of 4 E-blocks-L with a kernel size of one (see Figure 4.6.1.1). The input dimensions of each block and layer are presented in Table 4.12. The initial feature dimension was 64. We provide further training details in Section A.3.2.

4.6.2.2 E-CustomNets-L on Cifar-10

On Cifar-10, we evaluated architectures similar to the one presented above. Each architecture was parameterized by the number of blocks and the initial feature dimension (e.g., the Network in Table 4.12 has four blocks [10] with an initial feature dimension of 64). We tested a model with two blocks and an initial feature dimension of 128, comprising 500k trainable parameters; 3 models with three blocks and an initial feature dimension of 32, 64, and 128, with 162k, 570k, and 2.1M parameters, respectively. Furthermore, we tested a model with four blocks and an initial feature dimension of 64, with 2.2M parameters. Additional training details are given in Section A.2.2. We compared our results to the ALL-CNN-C net from Springenberg et al. [Spr+15] and a smaller version of the EfficientNet [TL19a; kua20]. We evaluated the minimal test error over all training epochs for each model.

[10] As each E-block-L has a pooling layer, an E-CustomNet-L stack contains one E-block-L

4.6.2.3 Stack substitution

In the above-described experiments, we propose architectures almost entirely consisting of E-blocks-L. However, we argued before that the biological motivation of using product terms to identify end-stopped signals rather justifies using those multiplications in the early stages of the visual processing chain.

Accordingly, we tested whether a hybrid model with product terms in early layers followed by convolution blocks could be a better approach. The starting point of our hybrid model is based on the SimpleNet [Has+16]. We substituted its first stack with different alternatives. The term *first stack* denotes all convolutions before the first max-pooling layer. Its input dimension was $32 \times 32 \times 3$, and it returned a tensor with the shape $32 \times 32 \times 128$. We compared models with four different first stacks:

0) The original stack. It contained four 3×3 convolutions with subsequent ReLU and BN with output dimensions 64, 128, 128, and 128, respectively.

1) The first stack of the *E-SimpleNet-L* contained a 1×1 convolution that expanded the three-dimensional input to 18 dimensions, followed by a DW convolution with kernel size 3, yielding a tensor with 18 feature maps. After BN, an E-layer-L (k=3) plus ReLU was applied with output dimension 128. This design choice was inspired by the mobile inverted bottleneck presented in the MobileNet-V2 [San+18a].

2) We used a single 3×3 convolution layer (input dimension 3 to output dimension 128) with BN and ReLU.

3) We used two 3×3 convolution layers, each with with BN and ReLU in sequence. The first layer's input dimension was three, and the output dimension 128. For the second layer, the input dimension was 128, and the output dimension 128.

We compared the average test errors over four runs with different starting seeds (different initial weights and batch order during training). The experiments were conducted on a single machine equipped with a GeForce RTX 2080 GPU, using PyTorch [Pas+19].

4.6.3 Results

On ImageNet, our network yielded a training accuracy (in percent) of 63 and a validation accuracy of 66 with 2.59M parameters.

The Cifar-10 results are listed in Table 4.13. Here, the EfficientNet-based model achieved a test error of 9.17 with 2.91M parameters. The All-CNN-C network we trained achieved a test error of 13.06 with 1.36M parameters. With a mean test error of 8.78, the E-net-L with three blocks, an initial feature dimension of 128, and 2.1M parameters yielded the best result in this comparison.

The results of the substitution experiments (see Section 4.6.2.3) are given in Table 4.14. In addition to the mean test error, the mean training and inference time for one epoch

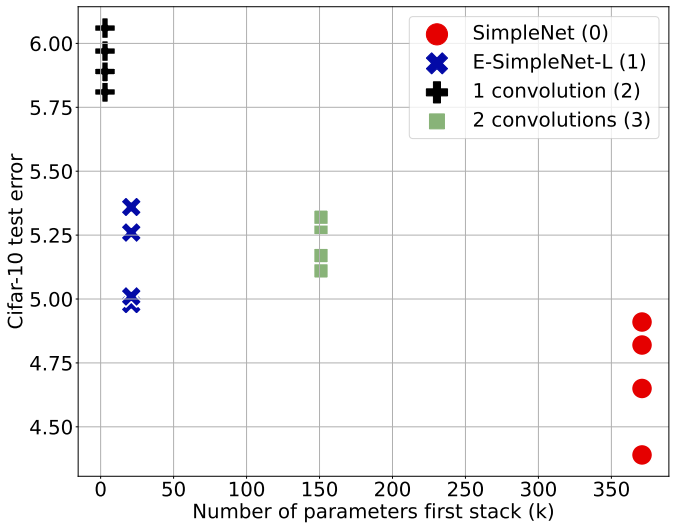


Figure 4.26: Results for substituting the first stack of the SimpleNet: the x-axis shows the number of parameters of the modified first stack (in steps of thousands). For each network, four runs were performed.

on the training- and test set, respectively, are presented. The plot in Figure 4.26 shows the Cifar-10 test error of each run on the y-axis and the parameter sizes of the different first stacks on the x-axis. With a mean test error of 5.25, our proposed E-SimpleNet-L outperformed networks with the first stack consisting of one or two convolutions. The original SimpleNet architecture yielded the best result, with 4.79.

4.6.4 Discussion

This section introduced E-nets-L as an alternative design principle. The main element of the network, the E-layer-L, transforms its input into the log-space, applies convolution, and transforms the output back by exponentiation. This way, the AND terms required for end-stopped cells are computed via an implicit multiplication.

On ImageNet, the presented shallow E-net-L can yield good results: its top-1 accuracy is better than the AlexNet's [KSH12] accuracy (66 vs. 62.5). At the same time, the E-net-L's number of parameters is smaller by a factor of 23. Figure 4.27 lists the top contenders on the ImageNet benchmark with less than 5 million parameters, showing that our model reaches a viable result.

Furthermore, on the Cifar-10 benchmark, our E-nets-L yielded comparable results to the two baseline models.

Table 4.13: Results on Cifar-10: we compared several E-nets-L, varying in size and being parameterized by the number of E-blocks-L ('Num. blocks') and the first feature dimension ('F-dim'). We compared these models to other networks. For the ALL-CNN-C, we could not reproduce the results presented in the paper (9.08 vs. 13.06).

Num. blocks	F-dim	Num. parameters (k)	Model	Error
3	32	161	E-block-L	12.89
2	128	491	E-block-L	11.25
3	64	570	E-block-L	10.53
N/A	N/A	1369	ALL-CNN-C [Spr+15]	9.08
N/A	N/A	1369	ALL-CNN-C (ours)	13.06
3	128	2133	E-block-L	8.78
4	64	2211	E-block-L	10.03
N/A	N/A	2912	EfficientNet	9.17

Table 4.14: Results for the SimpleNet first stack substitution on Cifar-10: the table shows the number of parameters of the first stack in steps of thousands ('stack num. param. (k)') and the mean test error values (± standard deviation) of four runs with different random seeds. Furthermore, the mean (± standard deviation) training time and inference time (in seconds) for one epoch on the training and test sets, respectively, are shown.

Model	Stack num. param. (k)	Mean error	Mean train. time (sec)	Mean test time (sec)
SimpleNet (0)	371	4.79 ± 0.21	24.71 ± 0.16	2.09 ± 0.14
E-SimpleNet-L (1)	21	5.25 ± 0.21	14.70 ± 0.22	1.44 ± 0.14
One convolution (2)	3	6.10 ± 0.10	12.71 ± 0.18	1.37 ± 0.09
Two convolutions (3)	151	5.34 ± 0.06	17.30 ± 0.13	1.72 ± 0.01

As shown with other E-nets, we can further reduce the number of operations of a sizeable CNN by creating an E-SimpleNet-L. This deep network yields no significant loss in performance while replacing four convolutions with only one block incorporating an E-Layer-L (see Table 4.14). Remarkably, reducing the number of convolutions did not yield a similar effect. Furthermore, the E-SimpleNet-L had a significantly lower training- and test time. Hence, the E-layer-L is another simple yet effective plug-and-play module that can possibly make any well-performing CNN more efficient.

4.6.5 Outlook

In the context of this thesis, E-nets-L can be seen as another way to employ AND combinations into CNNs to incorporate a beneficial inductive bias. However, using them to replicate the block architectures presented in the previous sections has no apparent

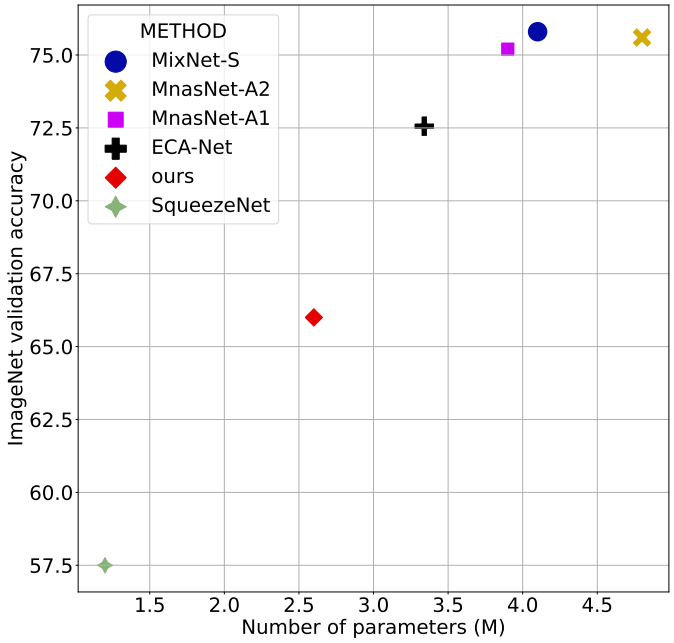


Figure 4.27: ImageNet top-1 accuracy of SOTA networks with less than 5 million parameters [Ian+16; TL19b; Tan+19; Wan+20] (results taken from [ST20]).

advantage: replacing a multiplication of two filters by an implicit multiplication – involving a transformation into log-space, a weighted sum, and a transformation back – is inefficient. Nevertheless, it is surprising that gradient descent works this well with highly non-linear operations such as the logarithm and exponentiation. Hence, establishing this proof-of-concept, i.e., making implicit multiplication feasible with modern CNNs, is the main contribution of this section.

E-nets-L may play a critical role in future work on AND combinations in CNNs. As Equation 4.8 shows, once in log-space, one can achieve large products efficiently via convolution, providing the means to (i) having single pixels inhibit entire outputs (see Figure 4.23) and (ii) being able to AND combine far more than two filters (see Figure 4.24). These attributes cannot be easily achieved with E-nets-I, E-nets-R, or E-nets-M. Regarding (ii), E-blocks-L could be redesigned to resemble an E-block-R (see Section 3.7 on page 36), but instead of combining only two feature maps, they combine three, four, or even all incoming feature maps. AND combining more filters implies a substantial increase in hyperselectivity (see Section 3.7.1 on page 37) which could further increase the robustness of such an E-net (see Section 4.4.3 on page 72). However, we cannot safely assume a direct correlation between hyperselectivity and robustness.

Even less so, we cannot directly correlate hyperselectivity and generalization. Still, further investigations into E-nets-L appear to be a viable field to improve CNNs with AND combinations.

4.7 Finding the optimal E-net design rule

Regarding the advances in CNN design since the ResNet, a typical scheme can be observed: a new main building block is proposed that is employed throughout a base structure with a certain number of stacks, blocks, and a certain width of a block at a specific position [TL19a; LZY21; How+17; Hua+17]. Looking at our research and the works of others that employ multiplications in CNNs (see Section 1.4), said scheme is not used – although there are exceptions [Chr+20]. E-nets, bilinear CNNs, and networks inspired by the Volterra kernel insert multiplications at specific positions within a CNN. This decision can be due to computational limits, as, for example, many value pairs can arise with an increasing number of feature maps. However, in our early experiments in Section 4.1.1.1, we have also witnessed that networks consisting only of E-blocks-I are prone to overfitting. Thus, designing an effective E-net implies finding the correct positions to substitute a standard convolution block with an E-block.

This fact reiterates how crucial a proper inductive bias is and bears a resemblance with the comparison of MLPs to CNNs (see, for example, Section 2.2.1). E.g., suppose we substitute all mobile inverted bottlenecks of a MobileNet-V2 with E-blocks-M [11] (keeping $q = 6$). This E-MobileNet-V2 has a higher capacity than the baseline CNN: each E-block-M can mimic the behavior of the original mobile inverted bottleneck by simply learning $\gamma = 0°$ for all filter pairs of the block. However, it can also create end-stopped neurons. Thus, in principle, we could train a model that uses AND combinations only where they are beneficial for generalization (certain E-neurons have $\gamma > 0°$). Unfortunately, such a model does not emerge during training. Instead, AND combinations are employed throughout the model, resulting in overfitting. Similarly, although having the capacity to learn convolutions, an MLP usually does not.

Hence, to create E-nets that improve SOTA CNNs, we need to impose a clear inductive bias that only allows end-stopped neurons at specific positions in the feature hierarchy of a deep network. So far, with E-nets-R and E-nets-M, we focused on the design rule 'substitute the first block of each stack'. Based on previous experiments, we chose this rule from a set of candidate design rules that we deemed biologically plausible.

However, we did not provide a systematic evaluation of the impact of E-block positioning. To this end, this section presents an exhaustive evaluation for E-PyrNets-M based on the PyrNet trained on Cifar-10 (for more details, see Section 4.5.1.1).

4.7.1 Models as bitstrings

Given an E-block and a baseline architecture consisting mainly of convolution blocks, creating an E-net comes down to several binary (1 or 0) decisions: for each standard block, *do we substitute it with an E-block (1), or do we keep it (0)?* Accordingly, for a

[11] We use E-blocks-M here because they behave linearly when $\gamma = 0°$, as discussed in Section 3.8.1.

baseline CNN with B blocks, each possible E-net can be expressed as a bitstring, and there are 2^B different E-net models.

For example, the bitstring of the E-PyrNet-20-M with three blocks per stack ($N = 3$) from Section 4.5.1.1 is "100 100 100" [12]; in the remainder of this section, we will refer to this model as the *default-model*. As has been shown, this E-PyrNet-20-M outperforms the baseline PyrNet-20. However, we cannot be sure if this model is the best out of all 512 possibilities. To see if better alternatives exist, we trained all 512 possible E-nets-M, using the same hyperparameters described in Section A.2.2, on two seeds. Then, we ranked all models, including the baseline and the default-model, by the mean test error (over two seeds) after 200 epochs.

Surprisingly, Table 4.15 shows that our previous approach only ranked 41st; the baseline was only 145th. Accordingly, the correct number and positions of substitutions play a vital role in the performance of an E-PyrNet-20-M: more than seventy percent of all E-PyrNets-20-M models perform worse than the baseline, as can be seen in Figure 4.28. However, almost ten percent of the models perform better than the default-model, showing that there exist E-PyrNets that outperform the results we presented in the previous sections.

4.7.1.1 Rules of thumb

We further analyzed the results using decision tree regression [13] to predict the mean test error of each model based on the E-block-M positions. The tree was constrained to a maximal depth of ten and to having at most 20 leaf nodes. Each split node minimized the squared error across all features. As features, we used the total number of E-blocks-M and the number of E-blocks-M in each stack. For example, the model "100 101 111" contains 6 E-blocks-M, with one, two, and three E-blocks-M in the first, second, and third stack, respectively. In addition, we computed the mean position of the E-blocks-M:

$$\text{Mean position} = \frac{\sum_{i=1}^{9} i \cdot m^i}{\sum_{i=1}^{9} m^i + 0.001}. \tag{4.9}$$

Here, m^i is the i-th entry of the model's bitstring.

The leaf node with the smallest test error prediction of 8.2 (the baseline error is 8.45) contained 83 samples with a mean squared error of 0.035. We can separate these 83 models from the rest of the dataset with the following feature splits:

- The number of E-blocks-M is less than 4.
- There is at most one E-block-M in the third stack.

[12] We use spaces between bits to separate stacks from one another.

[13] `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html#sklearn.tree.DecisionTreeRegressor`

- The mean position is less than 5.6.

Looking at models that performed (sometimes considerably) worse, only eight out of 256 samples with more than 4 E-blocks-M had a lower mean test error than the baseline.

In summary, no clear design rule can be derived; however, when designing an E-PyrNet-M, one can adhere to this *rule of thumb:* 'the number of E-blocks-M should be less than half the total number of blocks, and the blocks should be positioned in the first half of the network'.

Note that these rules are in accordance with the design rule of the default-model and with our preliminary considerations derived from biology: end-stopped neurons are rather located at the beginning of the visual processing chain, and they do not make up the bulk of neurons in the visual cortex.

4.7.1.2 Transition to MNIST

Knowing that we can find E-PyrNets-20-M that perform very well on one dataset (Cifar-10), the question is how well these models perform on other data. To gather insights, we trained all 512 models on MNIST again using two seeds (for more details on how we trained the models, see the Appendix, Section A.4). We have ranked each model by the maximum of the MNIST and Cifar-10 ranks (*max-rank*). For each dataset, the rank was determined via the mean test error, i.e., the model(s) with the lowest test error had the lowest rank, and the model(s) with the highest test error had the highest rank. We have rounded the test error values to three decimal places. Therefore, some models may share the same rank value.

Table 4.16 shows the results on Cifar-10 and MNIST for the ten best models, the best model on MNIST, the best model on Cifar-10, the baseline, and the default-model. Interestingly, the best model had a max-rank of 8, and only two models were among the top ten in both datasets. Accordingly, optimal performance on Cifar-10 does not imply optimal performance on MNIST and vice-versa. Still, 77 out of 512 models performed better than the baseline on both datasets.

Another tree regression showed that the rule of thumb is feasible to obtain E-PyrNets-20-M with an average max-rank of 49.5 (the baseline ranked 31st and 83rd on MNIST and Cifar-10, respectively). A tree regression, only focusing on the MNIST test score, revealed that having few E-blocks in the second stack yielded good results: from all possible E-PyrNets-20-M configurations, there are 24 models having fewer than four E-blocks, a mean position not higher than 5.4, and no E-blocks in the second stack; the average mean test score of those models was 0.402 (baseline: 0.455). For additional information, Figure 4.29 shows the two-dimensional histogram of the Cifar-10 and MNIST test errors and the confusion matrix.

In conclusion, our results indicate that we can improve CNN models even further by carefully choosing where to place E-blocks. However, good results obtained with one

dataset may not transform effortlessly to another – a problem that many CNNs face [TSS22].

Moreover, one could interject that almost 85% of all possible E-PyrNets-20-M performing worse than the baseline on at least one dataset is a bad result. However, this argument disregards that many of the subpar E-nets also did not match our views on where end-stopping happens in the visual cortex. Thus, we did expect that we could have discarded a majority of the possible configuration because they do not incorporate a bio-inspired inductive bias. For example, in hindsight, we did not need to train more than half of all possible models since they contained too many E-blocks (i.e., more than three E-blocks). In addition, if we had discarded all models with E-blocks located in the second half of the model (the mean position is larger than 4.5), we would have reduced the number of models to 50. In our results, of those 50 models, 47 outperformed the Cifar-10 baseline, 35 outperformed the MNIST baseline, and 34 outperformed both baselines, see Figure 4.30. Note that this subset of models contained the model with the best mean test score on Cifar-10, the model with the third-best mean test score on MNIST, and the model with the best max-rank.

When additionally considering the results of previous sections, we hypothesize that the bio-inspired rule of thumb is applicable to finding E-nets that obtain good results when classifying natural images. So the search for an E-net is not at all like looking for a needle in a haystack. On the other hand, one may wonder if there exist datasets where E-nets perform better with more E-blocks or E-blocks located at the end of the feature hierarchy. Especially when regarding IQA, E-nets-I with rather deep blocks indeed achieved good results (see Section 4.2.2). Thus, a good starting point for future research would be an E-block positioning analysis, as presented above, for image classification and IQA using E-nets-M and E-nets-I. Gathering more data here could help evaluate the impact of (i) the particular block design and (ii) the particular task.

Of course, the models evaluated here were small enough to train each possible E-PyrNet-20-M. The following section will present a method to find larger well-performing E-PyrNets-M, where an exhaustive search is not feasible.

4.7.2 Using a genetic algorithm for larger E-nets

Given the promising results on the smaller E-PyrNet-20-M (N=3, 512 possible models), the next step is to find E-PyrNet-M configurations for larger models. However, with $N = 5$, 7, and 9 blocks per stack, the number of choices quickly increases to over 32 thousand, 2 million, and 130 million possible E-PyrNets-M. Thus, we need to solve a discrete search problem with a vast search space, where each query (i.e., training a model to determine the test error) can take several hours. Accordingly, an exhaustive search is not feasible.

However, using a genetic algorithm is an obvious choice to finding good solutions for a discrete search problem, given that bitstrings are often an introductory example

Table 4.15: Test results on Cifar-10 for different E-PyrNet-20-M models: each model's bitstring defines which block of the baseline PyrNet is substituted with an E-block-M. We trained each model on two different seeds, and the table shows the mean, minimal, and maximal test error after 200 epochs of training. The first five columns show the best E-PyrNet-20-M models, as indicated by the rank column. The baseline ("000 000 000") and the default-model from Section 4.5.1.1 ("100 100 100") are only the 145th and 41st best models out of all 512 possible networks.

Rank	Bitstring	Cifar-10 Test error		
		mean	min	max
1	100 100 000	7.775	7.54	8.01
2	011 010 100	7.780	7.71	7.85
3	001 110 000	7.810	7.54	8.08
4	001 100 010	7.870	7.70	8.04
5	001 010 100	7.875	7.85	7.90
41	100 100 100	8.150	7.85	8.45
145	000 000 000	8.445	8.33	8.56

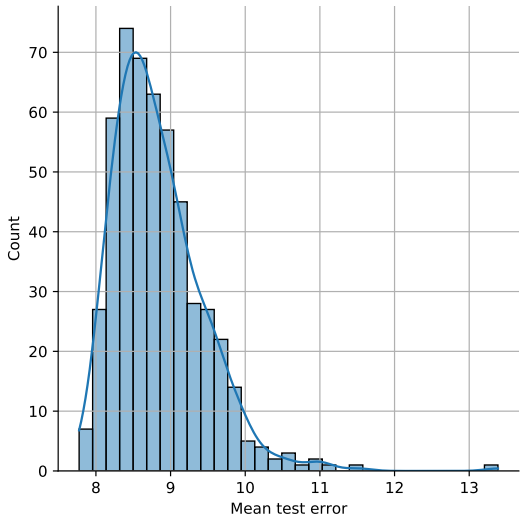


Figure 4.28: Distribution of mean test error values on Cifar-10 of all 512 possible E-PyrNets-20-M (including the baseline, having a mean test error of 8.45). The height of a bar shows the number of models having a test score within a certain interval, defined by the bar's x-axis position and width.

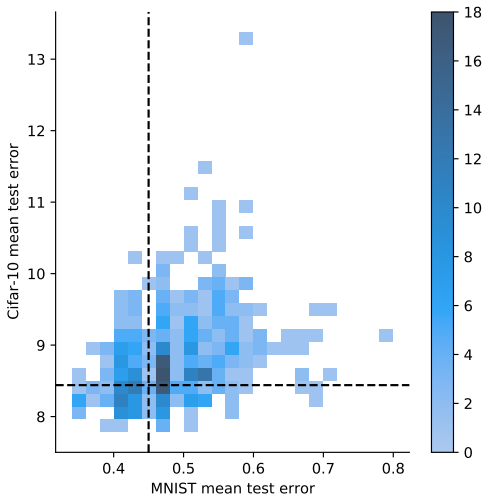
Table 4.16: Test results on MNIST and Cifar-10: in addition to evaluating all possible E-PyrNets-20-M on Cifar-10, we trained and tested all configurations on MNIST. We ranked each model by the max-rank, the maximum of the rank on MNIST (R-MNIST) and Cifar (R-Cifar-10). The table shows the top ten models based on the max-rank, the best model on Cifar-10, the best model on MNIST, the default-model, and the baseline.

Rank	Bitstring	MNIST mean	MNIST min	MNIST max	Cifar-10 mean	R-MNIST	R-Cifar-10
1	000 100 000	0.360	0.35	0.37	7.985	4	8
2	001 110 000	0.390	0.39	0.39	7.810	10	3
3	010 110 000	0.405	0.40	0.41	8.020	13	12
4	101 010 100	0.400	0.37	0.43	8.030	12	14
5	001 000 010	0.410	0.36	0.46	8.025	15	13
6	011 000 010	0.415	0.41	0.42	8.015	16	11
7	011 000 001	0.360	0.30	0.42	8.045	4	17
8	110 100 000	0.415	0.38	0.45	8.055	17	19
9	110 010 000	0.395	0.39	0.40	8.065	11	20
10	001 010 100	0.425	0.42	0.43	7.875	20	5
28	100 100 000	0.460	0.30	0.62	7.775	33	1
47	100 100 100	0.485	0.44	0.53	8.150	41	31
103	111 000 001	0.335	0.33	0.34	8.350	1	67
144	000 000 000	0.455	0.43	0.48	8.445	31	83

to genetic algorithms [RR02]. To some degree, the algorithm recreates the process of natural selection: a population of entities (in our case models) is tested for their fitness (in our case, the test error), and only the fittest entities of this population survive to pass on their genes (in our case, bitstrings) to another population, creating new genes by recombination and mutation. This iteration step is repeated several times such that better entities prevail. We can easily transfer this approach to our search problem: we select a population of models, train and evaluate them, and use these models to generate a new population.

The pseudo-code for the genetic algorithm is shown in Algorithm 1, its parameters are:

- The probability of using an and-combination instead of an or-combination when recombining two bitstrings (P_{and}).
- The mutation probability (P_{mut}).
- How many recombined survivors (N_{recomb}) are in the new population, see Algorithm 2.
- How many mutated survivors are in the new population (N_{surv}), see Algorithm 3.
- How many entities are drawn randomly for the new population (N_{rand}).



(a) Histogram

Cifar-10		MNIST: Better	MNIST: Worse
	Better	77	64
	Worse	90	281

(b) Table

Figure 4.29: (a): two-dimensional histogram of each E-PyrNet-20-M's test error on MNIST (x-axis) and Cifar-10 (y-axis). The color indicates the number of models, with darker bins containing more models. Black dashed lines show the test errors of the baseline model. (b): the corresponding confusion matrix reports the number of E-PyrNets-20-M that are better or worse than the baseline. Out of 512 models, 77 performed better than the baseline on MNIST and Cifar-10 (lower-left quadrant in the histogram plot). Ninety models performed better on MNIST but not on Cifar-10 (upper left quadrant in the plot); 64 performed better on Cifar-10 but not on MNIST (lower right quadrant in the plot), and 281 models performed equal or worse than the baseline (including the baseline, upper right quadrant in the plot).

Another critical factor is the starting population $\mathcal{S}_0 = \{m_0, m_1, ...\}$, where m_i is a bitstring. It is sensible to start with bitstrings that, for example, follow the rule of thumb in Section 4.7.1.1 instead of drawing bitstrings randomly.

We used the proposed genetic algorithm to find better-performing E-PyrNets-M for 5, 7, and 9 blocks per stack (i.e., E-PyrNets-32-M, E-PyrNets-44-M, and E-PyrNets-56-M). For each architecture, we used the same hyperparameters $P_{and} = 0.7$, $P_{mut} = 0.05$, $N_{surv} = 2$, $N_{recomb} = 3$, $N_{rand} = 1$. We evaluated twelve training runs in each iteration step, i.e., six different models for two different seeds. We chose the population size based on our hardware setup: four GeForce RTX 2080 GPUs with 11 GByte RAM each. For the smaller models, we were able to run three training sessions concurrently on one GPU. However, this was not possible for $N = 9$.

The genetic algorithm's hyperparameters were determined using the data from the $N = 3$ exhaustive search. For these data, determining the fitness was reduced to looking up the test error of the already trained model. Thus, we could run several genetic

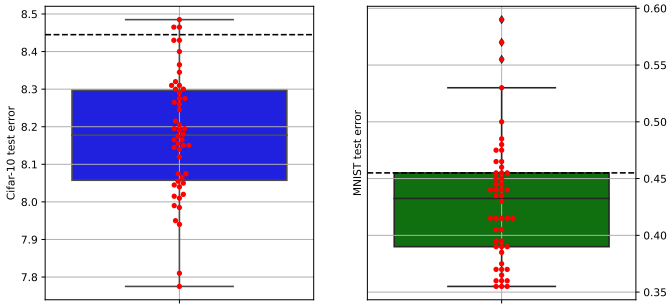


Figure 4.30: Results for models adhering to our rule of thumb: the majority of the 512 possible E-PyrNets-20-M do not match our biological considerations about end-stopping in the visual cortex. They have too many end-stopped neurons, or their end-stopped neurons are located in deeper layers. Models that are more sensible from a biological point of view yielded better results on MNIST and Cifar-10. Here we show the results of a subset containing 50 models. Each model has at most three E-blocks, and the mean position (see Equation 4.9) is at most 4.5. From those 50 models, 34 outperformed the baseline (black dashed lines) both on Cifar-10 (left panel) and MNIST (right panel). Red dots show the mean test errors of individual models.

algorithms with different hyperparameters. Lastly, we chose the hyperparameters, where the best possible solution was found with the smallest number of iterations. We did not alter the model training hyperparameters (see Section A.2) from Section 4.5.1.1.

We ran the genetic algorithms for 36, 12, and 30 iterations, yielding 216 ($N = 5$), 72 ($N = 7$), and 180 ($N = 9$) models, respectively. We started with $N = 5$, ran $N = 7$, and finished with $N = 9$. The starting population of each genetic algorithm was handpicked based on the results of the previous model. We report the results for the best ten models and, if available, the baseline and default-model in Table B.11 ($N = 5$), Table B.12 ($N = 7$), and Table B.13 ($N = 9$) in the Appendix.

We obtained E-PyrNets-M that performed better than the baseline on average. For $N = 5$, the mean test error of model "00011 01000 00010" was 6.43 (baseline: 7.2, based on Figure 4.20 on page 82). For $N = 7$, the mean test error of model "0000011 0100000 0000001" was 5.93 (baseline: 6.97). For $N = 9$, the mean test error of model "000110010 111000000 000000000" was 5.55 (baseline: 6.55). Thus, especially for larger models, E-PyrNets-M had a significant margin of one percent between the baseline mean test error and the E-PyrNet-M test error.

4.7.3 An evaluation with more runs

To increase the statistical validity of the above-mentioned findings, we compared the best E-PyrNets-M, the baseline, and the default-model on Cifar-10 for five runs instead of only two. For $N = 3$, we used the two best E-PyrNets-20-M based on the Cifar-10 mean test error (see Table 4.15) and the model with the best max-rank (see Section 4.7.1.2) on Cifar-10 and MNIST (Table 4.16). For the larger models, we each used the three best models based on the Cifar-10 genetic algorithm results (see Table B.11, Table B.12, and Table B.13).

Algorithm 1 Genetic E-net search

Require: $\mathcal{S}_0, P_{and}, P_{mut}, N_{mut}, N_{recomb}, N_{rand}, N_{iter}$
  $\mathcal{S}_{trained} \leftarrow \{\}$
  $k \leftarrow 0$
  while $k < N_{iter}$ **do**
    Train each model $m_j \in \mathcal{S}_k$ and compute the test error y_j
    $\mathcal{S}_{trained} \leftarrow \mathcal{S}_{trained} \cup \mathcal{S}_k$
    $m_{1st}, m_{2nd} \leftarrow$ Two models with the smallest y out of $\mathcal{S}_{trained}$
    $\mathcal{S}_{recomb} \leftarrow \text{RECOMBINE}(\mathcal{S}_{trained}, P_{and}, P_{mut}, N_{recomb}, m_{1st}, m_{2nd})$
    $\mathcal{S}_{mut} \leftarrow \text{MUTATE}(\mathcal{S}_{trained}, P_{mut}, N_{mut}, m_{1st})$
    $\mathcal{S}_{rand} \leftarrow \text{MUTATE}(\mathcal{S}_{trained}, \frac{1}{2}, N_{rand}, m_{zero})$ ▷ m_{zero} is the all-zero string
    $k \leftarrow k + 1$
    $\mathcal{S}_k \leftarrow \mathcal{S}_{recomb} \cup \mathcal{S}_{mut} \cup \mathcal{S}_{rand}$
  end while

Figure 4.31 shows the test error curves for the baseline, the default-model, and the best models obtained from the genetic algorithm or the exhaustive search. Here, for each N, we selected the best-performing of the three tested configurations (for the results of all models, see Table B.14). Using more seeds, the experiment did not reproduce the high error differences from the genetic algorithm results. Thus, sampling more runs for evaluation shows a downside of the genetic algorithm search: to process more E-PyrNet-M configurations, we reduce the number of runs for each E-PyrNet-M, decreasing the statistical validity of the results. Thus, the genetic algorithm overfits to models that perform exceptionally well on a few different runs. Accordingly, the genetic algorithm may select a different best configuration when started with different or more seeds.

Nevertheless, we could further improve the baseline and outperform the default-model. Thus, using a genetic algorithm is feasible but not optimal. On the upside, even better configurations may exist, showing that the overall idea of using AND combinations can enormously improve conventional CNNs. Thus, an efficient search strategy remains an open research question that could lead to very potent E-nets.

Algorithm 2 Create a set by recombining two strings

procedure RECOMBINE($\mathcal{S}_{trained}, P_{and}, P_{mut}, N_{recomb}, m_{1st}, m_{2nd}$)
  $\mathcal{S}_{recomb} \leftarrow \{\}$
  while $|\mathcal{S}_{recomb}| < N_{recomb}$ **do**
    for $i \in \{0, ..., |m_{1st}| - 1\}$ **do** ▷ Iterate over all bits
      $r \leftarrow \mathcal{U}(0, 1)$ ▷ Draw one number from the uniform distribution
      if $r \leq P_{and}$ **then**
        $m^i_{out} \leftarrow m^i_{1st} \wedge m^i_{2nd}$
      else
        $m^i_{out} \leftarrow m^i_{1st} \vee m^i_{2nd}$
      end if
    end for
    $m_{out} \leftarrow$ MUTATE-STRING(m_{out}, P_{mut})
    if $m_{out} \notin \mathcal{S}_{trained}$ **then**
      $\mathcal{S}_{recomb} \leftarrow \mathcal{S}_{recomb} \cup \{m_{out}\}$
    end if
  end while
end procedure

Algorithm 3 Create a mutated set

procedure MUTATE($\mathcal{S}_{trained}, P_{mut}, N_{mut}, m$)
  $\mathcal{S}_{mut} \leftarrow \{\}$
  while $|\mathcal{S}_{mut}| < N_{mut}$ **do**
    $m_{out} \leftarrow$ MUTATE-STRING(m, P_{mut})
    if $m_{out} \notin \mathcal{S}_{trained}$ **then**
      $\mathcal{S}_{mut} \leftarrow \mathcal{S}_{mut} \cup \{m_{out}\}$
    end if
  end while
end procedure

Algorithm 4 Mutation of one string

procedure MUTATE-STRING(m, P_{mut})
  for $i \in \{0, ..., |m| - 1\}$ **do**
    $r \leftarrow \mathcal{U}(0, 1)$
    if $r \leq P_{mut}$ **then**
      $m^i \leftarrow \neg m^i$
    end if
  end for
end procedure

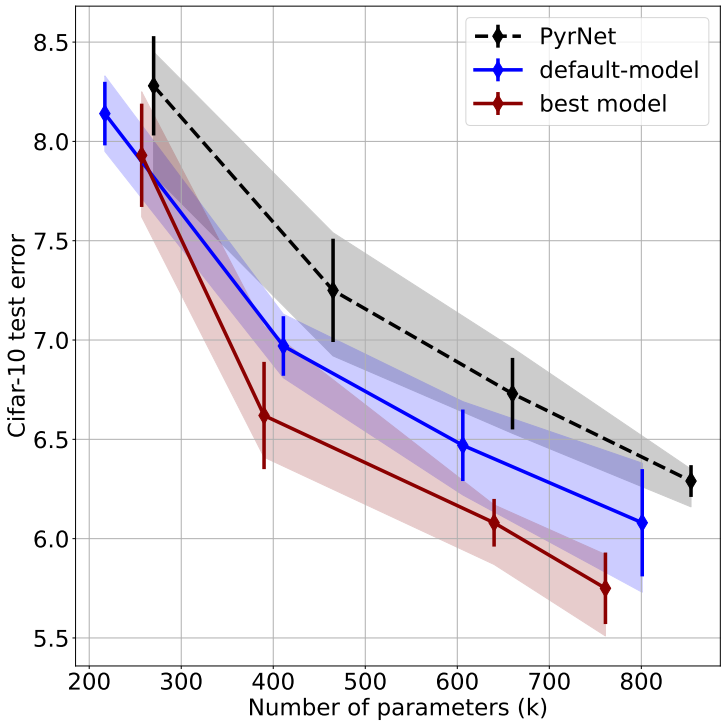


Figure 4.31: Cifar-10 test error comparison: diamonds show the mean test error after 200 epochs averaged over five runs, with error bars indicating the standard deviation. The transparent area shows the range from the minimal to the maximal test error. The baseline results are black, and blue shows the default-model results. The results for the best E-PyrNet-M configurations derived from the genetic algorithm (or the exhaustive search for $N = 3$) are shown in red.

4.8 E-nets for medical image segmentation

So far, we have shown that E-blocks can improve CNNs used for image classification. However, apart from this essential task in computer vision, a wide range of applications and challenges can be tackled with CNNs and, thus, with E-nets. Hence, to further research the capabilities of E-nets, this section examines their performance in medical image segmentation. We selected two distinct medical image segmentation tasks, reproduced the results of the best CNN approaches, and investigated if we could improve them with our methods. Note that this section is based on a joint project with Tilman Wegener, a Master's student under my supervision [Weg21].

Segmentation is the classification of each pixel. It is used to obtain a fine-grained map of *what* is *where* in an image. This automated fine-grained object location can be viable in the medical field: for example, in radiology, software can help detect tumors, lesions, or organs in CT scans, X-ray scans, or other modalities, see Hesamian et al. [Hes+19] for an overview. Similarly, dermatologists may obtain a second opinion using a computer vision method to locate moles and determine if they should be tested for skin cancer [Cod+18].

Another application for medical image segmentation – for example in research – is the automated evaluation of microscopic cell images. Note that on this subject, I published works that were not directly linked to E-nets. The publications covered topics such as CNN-based image segmentation of axons [Grü+20a; Pal+21] and white blood cells [Grü+21], how to use active learning to decide which images to annotate next [Grü+20b], and how to employ semi-supervised learning in cell segmentation [GM20]. In addition, I participated in the 2018 MoNuSeg Challenge [Kum+19; GB].

Fully convolutional networks and U-nets Looking at the development of the SOTA in segmentation, one can observe many similarities to image classification. Again, the use of CNNs led to a significant performance boost, and many of today's best-performing methods are based on CNNs. Coming from approaches involving thresholding, watershed, and many more handcrafted techniques [Sze10; Min+22], CNNs were first employed in sliding window methods, where each window was classified separately [Cir+12].

Then, Long, Shelhamer, and Darrell [LSD15] proposed a so-called fully convolutional network (FCN) that can be trained and tested far more efficiently. FCNs only use convolutions and do not mix them with MLPs as older models like the AlexNet [KSH17] or VGG-16 [SZ14] do. In fact, fully-connected layers are replaced with 1×1 convolutions. This approach is sensible because, on a pixel level, a sequence of 1×1 convolutions with non-linearities is an MLP. In classification, an MLP, transforms a input vector $\vec{x} \in \mathbb{R}^{d_{in}}$ into a representation- or class-vector $f(\vec{x}) = \vec{z} \in \mathbb{R}^{d_{out}}$. Similarly, 1×1 convolutions transform each feature vector $\mathbf{X}[i, j, :] \in \mathbb{R}^{d_{in}}$ at pixel position (i, j) into a different vector.

An FCN can be trained with a batch of b input images $\mathbf{X} \in \mathbb{R}^{b \times h \times w \times d_{in}}$ and the corresponding segmentation maps $\mathbf{Y} \in \mathbb{R}^{b \times h \times w}$ denoting which pixel belongs to which of the c classes. The loss function of a batch is averaged over each pixel and each sample (see Equation C.5 in the Appendix for an example).

The two main components of an FCN are the encoder and decoder. The encoder is often a classification CNN (without fully-connected layers) that transforms an often large input image (e.g., 224×224 pixels and three channels) to a latent representation tensor with smaller height and width but more channels (e.g., $8 \times 8 \times 2048$). For example, the first FCN approach used a modified pre-trained VGG-16 [SZ14] as the encoder. The decoder processes and upsamples the representation tensor back to the original height and width with one channel for each object class – e.g., $224 \times 224 \times 10$ for ten classes. In Long, Shelhamer, and Darrell [LSD15], the decoder takes tensors from different layers of the encoder, processes each tensor to have as many feature maps as object classes, and upsamples them to the original image size via transposed convolution. [14]

This early approach is not without flaws because the outputs are based on downsampled feature maps. Downsampling (i.e., using strided convolutions or max-pooling) is necessary for efficiently processing an input image since it quickly increases the receptive field of a CNN, giving intermediate neurons the needed context. However, downsampling also destroys spatial information that cannot be recovered by upsampling.

Ronneberger, Fischer, and Brox [RFB15] provided a remedy to this problem by introducing the U-net. The encoder consists of several *stages*. [15] For the U-net architecture, each stage contains two convolutions and a max-pooling layer. Analogously, the decoder has the same number of stages where each stage starts with an upsampling operation (transposed convolution or bilinear upsampling) followed by two convolutions. An encoder stage and a decoder stage are at the same *level*, if they have the same input dimensions $h \times w$. With the U-net, at each level, the output of the encoder stage is concatenated to the upsampled decoder stage input – this operation is called *skip connection*. Thus, low-level spatial information is preserved and passed on to the decoder. For example, the input of the decoder's last stage also contains object boundaries extracted from the first stage of the encoder. The U-net won the international symposium on biomedical imaging (ISBI) challenge for segmentation of neuronal structures in electron microscopic stacks by a large margin and is, until today, the standard architecture for medical image segmentation. However, there are now U-nets of many different forms. For example, many networks use encoders that differ from the original.

[14] In a simplified view, transposed convolutions are reversed convolutions. I.e., one pixel is mapped to an entire patch of size $k \times k$ – k being the kernel size.

[15] Regarding only the encoder, a stage and a stack are the same and can come in many forms. However, unlike a stack consisting of many blocks, a stage may only consist of a few layers.

In this section, we will create and examine E-nets based on the SA-U-net [Guo+21], being a particular U-net architecture that was created to segment blood vessels in retina images. Furthermore, we investigate a transfer learning approach (based on the DSNet [Has+20]) using the ImageNet-trained E-ResNet-50-I from Section 4.1.1.2 and E-ResNet-50-R from Section 4.3.1 as encoders for skin lesion segmentation.

4.8.1 Methods

We investigate datasets that are binary segmentation tasks. Thus, for each pixel, the CNN must decide if a pixel belongs to the object (foreground) or not (background). In many cases, the class distribution is heavily skewed in favor of the background pixels. Hence, even a trivial classifier, assigning any pixel to the background class, may obtain a misleading accuracy above 90%. Thus, despite image segmentation being essentially a pixel-wise classification, the standard evaluation regime for classification does not suffice to judge the quality of a model. Hence, apart from estimating accuracy, an appropriate evaluation needs additional metrics. When presenting the results, we mainly focus on the Matthews correlation coefficient (MCC); however, we also report the Dice score and the Jaccard index. For more information, see Section C in the Appendix.

4.8.1.1 Retinal vessel segmentation

With the Spatial Attention (SA-) U-net, Guo et al. [Guo+21] achieved SOTA performance on two retinal vessel segmentation datasets: digital retinal images for vessel extraction (DRIVE) [Nie+04] and CHASE [Fra+12]. The SA-U-net is based on the original U-net. However, each convolution is followed by a *DropBlock* layer [GLL18], a ReLU, and BN. Furthermore, the model extends the middle stage with a spatial attention module. Figure 4.32 shows the architecture. We refer to the different stages as Down1, Down2, Down3, Middle, Up1, Up2, and Up3. The FCN was re-implemented in PyTorch. For DropBlock, we used the Code provided by Ghiasi, Lin, and Le [GLL18].

We create E-SA-U-nets, by substituting standard stages with E-stages, i.e., stages that contain E-blocks. It is sensible to challenge the notion that designs that worked best on classification will also be the best on segmentation. For example, in early experiments using BN yielded better results than using instance normalization (IN). Accordingly, we mainly use two E-blocks that are slight derivations of the E-block-R: instead of IN, (i) the E-block-B uses BN as ϕ, and (ii) the E-block-BR uses BN and a subsequent ReLU as ϕ. Additionally, we investigate the impact of removing the ReLU for the E-block-R; we call this block the E-block-C. Table 4.17 gives an overview of the mentioned E-blocks. Moreover, we compared E-blocks with and without residual connections. We used the postfix '-NR' to notify that the residual connection was removed.

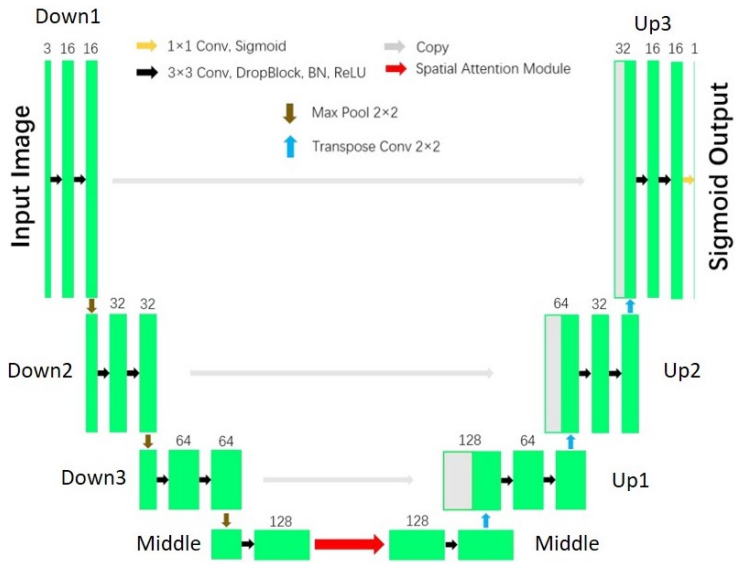


Figure 4.32: SA-U-net: the model consists of seven stages; each stage consists of two 3×3 Conv - DropBlock - BN - ReLU sequences. Furthermore, the Middle stage is extended with a spatial attention module. Our experiments test the effectiveness of substituting different stages with E-blocks on retinal vessel segmentation. Figure adopted from Guo et al. [Guo+21].

Since our previous results showed that combinations of convolution blocks and E-blocks are beneficial, we investigated several combinations of the SA-U-net's standard convolution block (denoted as Conv) and E-blocks. Here, 'Conv' is a 3×3 convolution with DropBlock regularization, BN, and ReLU. An overview is given in Table 4.18.

4.8.1.2 Skin lesion segmentation

At the time conducting our segmentation experiments, the DSNet presented by Hasan et al. [Has+20] was one of the best skin lesion segmentation approaches for the international skin imaging collaboration (ISIC) 2017 [Cod+18] and PH2 [Men+13] datasets. Again, the model is based on a U-net using skip connections. The model uses an ImageNet pre-trained DenseNet-121 [Hua+17] as the encoder. Each decoder stage consists of a 3×3 DW convolution followed by a 1×1 convolution, ReLU, BN, and bilinear upsampling. The final up-stage has an output dimension of 128 that is further processed by a series of convolutions, normalization layers, and non-linearities to produce a segmentation mask.

We used a similar decoder in our experiments but exchanged the DenseNet encoder with a ResNet-50. We compare this baseline, we denote Res-DSNet, to models using the ImageNet-trained E-ResNets-50 from Section 4.1.1.2 and Section 4.3.1.2, respectively. The second and fourth stacks were replaced with a single E-block-I to create an

Table 4.17: Overview of different E-blocks of the E-SA-U-net: an E-block is defined by the AND combination (ζ) of two DW convolution outputs that may be further processed by a function ϕ (see Section 3.5 on page 31 for more information).

Block	ϕ	ζ	res. Connection?
E-block-B	BN	Multiplication	Yes
E-block-B-NR	BN	Multiplication	No
E-block-BR	BN and ReLU	Multiplication	Yes
E-block-BR-NR	BN and ReLU	Multiplication	No
E-block-C	IN	Multiplication	Yes
E-block-R	IN and ReLU	Multiplication	Yes

Table 4.18: Different E-block and conventional block combinations we used for the E-SA-U-net.

Name	Description
E-block-BR-Conv-E-block-B	E-block-BR, 3×3 Conv, DropBlock, BN, ReLU, E-block-B
Conv-E-block-BR	3×3 Conv, DropBlock, BN, ReLU, E-block-BR
E-block-BR-Conv	E-block-BR, 3×3 Conv, DropBlock, BN, ReLU
E-block-R-Conv	E-block-R, 3×3 Conv, DropBlock, BN, ReLU

E-ResNet-50-I. For the E-ResNet-50-R, each stack's first block was replaced by an E-block-R. We denote the two models as E-DSNet-50-I and E-DSNet-50-R. See Figure 4.33 for more details.

4.8.2 Experiments

4.8.2.1 Retinal vessel segmentation

We start by comparing the SA-U-net to E-SA-U-nets. We either changed only a single stage of the baseline network or multiple stages, investigating how E-block positioning affected the results. In addition, we analyzed the impact of several E-net hyperparameters, such as using a residual connection or the impact of the expansion factor. All experiments were conducted on the DRIVE (see Section C.2 in the Appendix) dataset containing 20 training images and 20 test images each.

Replacing single stages with different E-blocks We created E-net models by substituting one out of the seven stages of the SA-U-net (see Figure 4.32). We evaluated eight different E-stages. We used four E-stages consisting of a single E-block: the E-block-B, the E-block-B-NR, the E-block-BR, and the E-block-BR-NR (see Table 4.17).

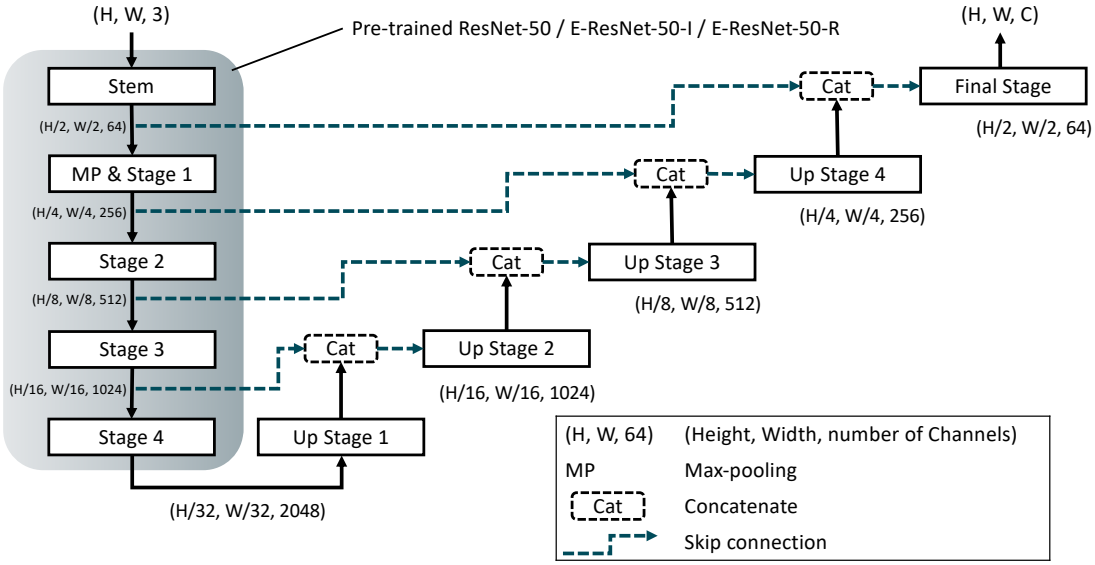


Figure 4.33: Res-DSNet models: we compared the baseline, using a ResNet-50 encoder, to an E-DSNet-I with the E-ResNet-50-I from Section 4.1.1.2 as encoder, and to an E-DSNet-R with the E-ResNet-50-R from Section 4.3.1.2 as encoder. For more information on the ResNet-50's structure, see Table 4.2 on page 49.

Furthermore, we used four different combinations of the SA-U-net standard convolution block with E-blocks (see Table 4.18).

Replacing multiple stages with E-blocks Next, we replaced more than one stage of the SA-U-net with combinations of the eight above-mentioned E-stages. In total, we evaluated 18 different models.

Analyzing different E-block parameters We furthermore quantified the impact of different E-block hyperparameters on model generalization. For single E-blocks, we added or removed the residual connections, investigated the role of the expansion factor q and evaluated using IN instead of BN. We compared these results to a Conv-E-block combination.

4.8.2.2 Skin lesion segmentation

We trained the Res-DSNet and the corresponding E-net versions and compared these results to the original DSNet results presented in Hasan et al. [Has+20]. The models and the Res-DSNet were trained on the 2 000 training images and tested on the 600 test images of the ISIC 2017 dataset. Additionally, we used the PH2 dataset [Men+13] for testing. The data are further described in Section C.3 in the Appendix. We used

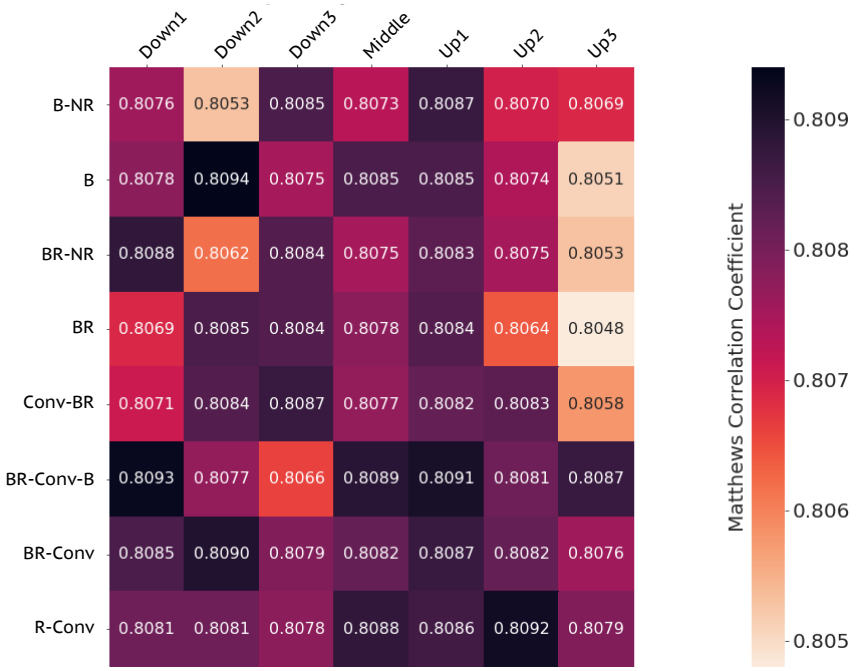


Figure 4.34: Mean MCC over five runs when replacing a single stage with an E-stage. The baseline mean MCC was 0.8079. We removed the term 'E-block' for better readability in the row labels. Column labels describe which stage was exchanged with an E-block. Figure adapted from [Weg21].

the DSNet training hyperparameters (see Section C.3.4 in the Appendix) to train all models.

4.8.3 Results

4.8.3.1 Retinal vessel segmentation

Replacing single stages with different E-blocks Figure 4.34 shows the mean MCC (see Appendix Section C.1 for details) of five runs. 32 out of 56 E-SA-U-nets outperformed the baseline model which had a mean MCC of 0.8079. We obtained the best E-SA-U-net, having a mean MCC of 0.8094, using an E-block-B in stage Down2.

Combining convolution blocks with E-blocks resulted in models more invariant to position changes. Using E-stages in later stages of the network, e.g., Up2 and Up3, primarily resulted in a subpar performance. Considering our position analysis in Section 4.7 and our preliminary considerations, this result is unsurprising.

From the results, we cannot derive a single best approach when comparing the eight different E-stages. Furthermore, residual connections or a ReLU non-linearity were only beneficial for certain positions.

Table 4.19: Best five out of 18 models when replacing multiple stages (e.g., denoted by D1 for 'Down1' and U1 for 'Up1') with E-stages: results were averaged over five runs. A † denotes that the E-stage 'E-block-BR-Conv-E-block-B' was used.

Replaced stages	Dice	Accuracy	MCC
Reproduced SA-U-net	0.8230 Max: 0.8248	0.9694 Max: 0.9700	0.8079 Max: 0.8100
D1† U1†	0.8242 Max: 0.8259	0.9692 Max: 0.9700	0.8090 Max: 0.8107
D1† D3†	0.8243 Max: 0.8259	0.9689 Max: 0.9698	0.8088 Max: 0.8104
D1† Middle† U1†	0.8238 Max: 0.8259	0.9692 Max: 0.9698	0.8085 Max: 0.8106
D1 (E-block-BR-NR) D2 (E-block-B)	0.8234 Max: 0.8254	0.9687 Max: 0.9697	0.8080 Max: 0.8102
D1† D2†	0.8234 Max: 0.8254	0.9687 Max: 0.9697	0.8080 Max: 0.8102

Replacing multiple stages with E-blocks Table 4.19 presents the results for the best five out of 18 evaluated models with more than one stage being substituted with an E-stage. The table shows the mean Dice score, accuracy, and MCC averaged over five runs. Each of the five models outperformed the baseline in terms of Dice score and MCC. In this experiment, the best model yielded an MCC of 0.8090. The results further indicate that 'E-block-BR-Conv-E-block-B' is a promising E-stage to improve an SA-U-net with E-blocks.

Study with different E-block parameters Table 4.20 shows the results of the hyperparameter analysis based on ten runs. We used a model where the stages Down1 and Down3 were exchanged with single E-blocks employing residual connections (MCC = 0.8079). Removing the residual connections reduced the performance (MCC = 0.8073). Thus, using residual connections was beneficial for models containing multiple E-stages. A combination of E-blocks and convolution blocks further increased the performance (MCC = 0.8088). Note that this larger model roughly contains the same number of trainable parameters as the baseline. Although a deeper model improved performance, using a wider model by increasing the expansion factor was not beneficial (MCC = 0.8073). Furthermore, exchanging IN with BN decreased performance as well (MCC =

Table 4.20: E-block hyperparameter analysis: we replaced the stages 'Down1' (D1) and 'Down3' (D3) in the SA-U-net with different E-stages. Results are based on ten runs.

Replaced stages	Dice	Accuracy	MCC
D1 (E-block-BR)	0.8232	0.9691	0.8079
D3 (E-block-B)	Max: 0.8255	Max: 0.9699	Max: 0.8101
Remove residual connection?			
D1 (E-block-BR-NR)	0.8225	0.9689	0.8073
D3 (E-block-B-NR)	Max: 0.8249	Max: 0.9698	Max: 0.8094
Use † = 'E-block-BR-Conv-E-block-B' combination to replace stage?			
D1†	0.8243	0.9689	0.8088
D3†	Max: 0.8259	Max: 0.9698	Max: 0.8104
Use inverted bottleneck with $q = 2$ instead of $q = 1$?			
D1 (E-block-BR ($q = 2$))	0.8224	0.9693	0.8073
D3 (E-block-B ($q = 2$))	Max: 0.8249	Max: 0.9699	Max: 0.8094
Use IN instead of BN after DW convolution?			
D1 (E-block-R)	0.8216	0.9691	0.8064
D3 (E-block-C)	Max: 0.8231	Max: 0.9697	Max: 0.8076

0.8064). Accordingly, comparing classification to medical image segmentation shows that E-block hyperparameters affect a model's performance differently; for example, Table 4.11 on page 68 shows that wider models perform better on the ImageNet classification dataset while here, a narrower model performed better.

Nevertheless, using E-blocks, we improved this SOTA approach in retina vessel segmentation.

4.8.3.2 Skin lesion segmentation

Table 4.21 shows the results for the DSNet, the Res-DSNet baseline, the E-DSNet-50-I, and the E-DSNet-50-R on the ISIC 2017 test dataset. The E-DSNet-R performed best considering the Jaccard index and the MCC. Looking at the Jaccard index, all ResNet-50 models outperformed the DSNet, having a lower sensitivity but higher specificity.

Table 4.21: ISIC 2017 results: the DSNet results are reported from Hasan et al. [Has+20].

Model	Jaccard	Sensitivity	Specificity	MCC
DSNet [Has+20]	0.775	0.875	0.955	-
Res-DSNet	0.783 Max: 0.786	0.868 Max: 0.875	0.960 Max: 0.968	0.828 Max: 0.831
E-DSNet-I	0.776 Max: 0.783	0.852 Max: 0.863	0.967 Max: 0.972	0.823 Max: 0.828
E-DSNet-R	0.783 Max: 0.788	0.865 Max: 0.873	0.963 Max: 0.970	0.831 Max: 0.835

Table 4.21 shows the results for the PH2 dataset. Surprisingly, among the ResNet models, the E-DSNet-I performed best, although having fewer parameters than the other approaches. However, the DSNet outperformed the ResNet models, having a higher specificity and a lower sensitivity.

One explanation for the better performance of the original DSNet on the PH2 dataset could be the difference in network depth: compared to the original DenseNet-121 (with 121 linear layers), the ResNet-50 and E-ResNets-50 are shallower, which can often decrease the performance. Thus, an E-DenseNet-121 or a deeper E-ResNet-101 or E-ResNet-152 may outperform the DSNet.

4.8.4 Discussion

Using E-neurons, we have enhanced SOTA approaches in medical image segmentation. Accordingly, our bio-inspired methods for recognizing natural images are also viable for medical image segmentation. Furthermore, our results again show that transfer learning with E-nets is feasible. However, not all design choices that led to better results in image recognition were beneficial for segmentation. E.g., residual connections helped in certain situations but not all, and the E-ResNet-50-I – being smaller but on par with the ResNet-50 on ImageNet – sometimes outperformed the larger E-ResNet-50-R, which generalized better on ImageNet.

Moreover, IN was not beneficial in medical image segmentation – even though we obtained better results with IN than with BN in image classification. We hypothesize that IN may have a certain advantage (A) over BN but also some disadvantages (B). Furthermore, the balance of A and B may change depending on the dataset.

Concerning (A): IN ensures that each filter output feature map has a zero mean and standard deviation of one. Thus, reducing the likelihood of extreme outliers and exploding values when multiplying two feature maps.

Table 4.22: PH2 results: all networks were trained on the ISIC 2017 dataset. The PH2 dataset was only used for testing. The DSNet results are reported from Hasan et al. [Has+20].

Model	Jaccard	Sensitivity	Specificity	MCC
DSNet [Has+20]	0.870	0.929	0.969	-
Res-DSNet	0.843 Max: 0.853	0.975 Max: 0.979	0.920 Max: 0.933	0.856 Max: 0.867
E-DSNet-I	0.847 Max: 0.855	0.976 Max: 0.978	0.924 Max: 0.932	0.860 Max: 0.866
E-DSNet-R	0.837 Max: 0.844	0.981 Max: 0.984	0.910 Max: 0.916	0.851 Max: 0.856

Concerning (B): IN is, for example, invariant to constant changes. I.e., given a batch consisting of an input tensor $\mathbf{T}$ and $\mathbf{T} + c$ for some constant c, the output of an IN layer produces two identical output tensors. However, BN yields two different outputs. It is conceivable that this invariance, i.e., removing information, can be a disadvantage.

Since they are captured in a more controlled setup, the variation of the retina vessel and skin lesion images is lower than that of images from ImageNet or Cifar-10. Hence, extreme outliers of feature maps may be less likely for the used segmentation datasets – reducing the advantage (A). Accordingly, in image classification, IN may outperform BN because the advantage (A) outweighs the disadvantage (B). However, in medical image segmentation, BN may be a better choice because B outweighs A.

Nevertheless, using E-nets in medical image segmentation is a promising research field to further enhance CNN performance. However, the results show that additional research is needed in this particular domain because the best practices to model end-stopped cells derived from image classification did not readily transfer to medical image segmentation.

The gains we obtained here were smaller than in the previous sections – maybe because we have not yet found an optimal E-block design for segmentation. On the other hand, we did improve results that were (at the time when the experiments were conducted) the best approaches on the respective datasets. Thus, it is remarkable that E-neurons can increase even the performance of these highly optimized approaches.

As shown in Section 4.7, the number of blocks and their positions are a crucial hyperparameter, and adequate solutions can be found, for example, using our genetic algorithm. If it takes a long time to train the networks, e.g., when the dataset is large, the genetic algorithm may only evaluate a few of the many possible E-nets; hence, an optimal E-net may not be found. However, one typical problem of medical

image segmentation tasks – the lack of labeled data [16] – can be an advantage here: training a model on a dataset such as DRIVE, containing only 20 training images, is usually a matter of minutes. Accordingly, many E-nets can be evaluated, increasing the probability of finding an E-net that strongly improves the baseline model. Thus, future research on the positioning of E-blocks may prove that E-nets are especially useful for medical image segmentation.

[16] E.g., the workload of creating labels for microscopic axon images is high since labeling requires expert knowledge and time. Hence, time constraints usually only allow labeling a small subset of k images from a large pool of unlabeled data. We proposed an active learning approach picking the best k images that should be labeled next to improve a segmentation model's generalization [Grü+20b].

Chapter 5

Discussion

The main goal of this work was to invent new and better deep networks in computer vision. Generally, two routes can lead to an improved network, either (i) data-driven, i.e., focusing on improving the data or increasing the amount of data and how well a deep network scales with an abundance of data, or (ii) improving the model's inductive bias, e.g., by changing a deep network's structure to be more suited for a task. The data-driven approach can be observed with the recent development of vision transformers [Dos+20]. These models scale well with ample data but may perform subpar with smaller datasets. Thus, in many applications, these models may not be helpful. Compared to vision transformers, convolutional neural networks (CNNs) rather belong to the bias-driven approach, where expert knowledge about the nature of the machine learning tasks and the data enhances generalization. In our research, we took this second route, aiming to improve the inductive bias of CNNs further.

A proper bias is often inspired by domain knowledge: something expected to be beneficial for a specific task is explicitly incorporated into a machine learning model. In our case, we expected the focus on image regions with an intrinsic dimensionality (iD) of two to be beneficial, as evidence is plenty for the importance of these particular regions. In the visual cortex, some cells are mainly activated by these 2D regions [ZB90] – a property called end-stopping. In this work, approaches to explicitly integrate end-stopping into CNNs were presented, creating a novel type of CNN architecture suitable for various vision tasks.

5.1 E-neurons improve CNNs

We started from the reasoning that CNNs – like the mammalian visual system – have layers of neurons representing the incoming data with a hierarchy of features that become more complex with each layer [HW68; Fuk80]. When moving along that hierarchy, somewhere, end-stopped cells extract 2D features. This phenomenon likely happens in basic CNNs because it was shown that larger sequences of convolutions with non-linearities could model end-stopping [BZ98]. However, a more efficient approach –regarding the number of operations– would be to find the positions in the feature

hierarchy where end-stopping is needed and substitute a larger number of convolutions with fewer end-stopped operations.

We developed E-blocks that can model end-stopped behavior and created E-nets by swapping a subset of convolution blocks of a baseline CNN with E-blocks. For example, by exchanging two sequences of convolution blocks with two single E-blocks, we transformed a baseline ResNet-50 into an E-ResNet-50-I (see Section 4.1.1.2). On ImageNet, the E-ResNet-50-I performed just as well as the original while using fewer parameters and operations. Moreover, the E-ResNet-50-I performed on par with a ResNet-50 in a transfer learning setup for image quality assessment (IQA) (see Section 4.2.2) and segmentation (see Section 4.8.2.2). The results indicate that our initial reasoning is sound, and, in terms of efficiency, the new E-ResNet-50-I is better than its baseline.

Moreover, by exchanging single convolution blocks with single E-blocks – thus, focusing less on reducing the number of operations – we created the E-ResNet-50-R and E-MobileNet-V2-R that outperformed the ResNet-50 and MobileNet-V2 on ImageNet, respectively (see Section 4.3.1.2). Furthermore, on Cifar-10, we created E-nets-R and E-nets-M that generalized better than their baseline (ResNet, PyrNet, and DenseNet), often while having fewer parameters, fewer operations, and being more robust to adversarial attacks and JPEG compression artifacts (see Sections 4.3.1.1, 4.4.3, and 4.5.1). Furthermore, using E-nets, we improved state-of-the-art (SOTA) approaches for medical image segmentation (see Section 4.8). Hence, we indeed created E-nets that often outperform conventional CNNs in many different aspects.

5.2 E-neurons are end-stopped

In Section 1.3, we argue that mammalian vision systems contain, among many other cell types, end-stopped cells. These cells focus on image regions with an iD of two. Statistical analysis showed that in natural images, these 2D signals appear less often than 1D or 0D signals [BW00; ZBW93]; thus, looking at information theory, 2D signals contain higher amounts of information. This point is furthermore supported by the fact that knowing only the 2D regions in a gray-scale image suffices to reconstruct the image [MB00]. In that same regard, Section 4.2 shows that we successfully used the structure tensor to detect 2D image patches that we then used to improve our image quality scoring methods.

Our proposed architecture contains E-neurons that can detect 2D regions by learning two oriented filters and AND combining the filter results (see Section 3.4). The model analysis in Section 4.4 shows that our E-blocks indeed display end-stopped behavior: we investigated the degree of end-stopping using a specific test image and quantified the entropy (i.e., sparseness) of activations that increased with the angle between filter-pair vectors.

Still, one may argue that other factors than end-stopping explain the promising results. Mainly that multiplicative pairwise interactions increase a model's capacity. When focusing solely on the multiplication aspect, E-nets indeed resemble factorized bilinear (FB)-CNNs [Li+17b] that are not directly inspired by biological vision but rather by incorporating a Volterra kernel [Vol59] into specific layers – similar to a kernel SVM [CV95]. In Section 3.6.1, our analytical results show an intersection between the sets of possible solutions of an FB-layer and an E-block. However, some solutions can only be modeled by an E-block and not by an FB-layer – and vice versa. Furthermore, as opposed to bilinear FB-CNNs, where FB-layers work best when located at the end of the model, E-nets generalize best with E-blocks located in the first half of the model (see Section 4.7.1.1). In addition, E-nets-M, which use the minimum operation as AND combination, yield comparable results to E-nets-R, which use multiplications. Hence, we assume that the critical aspect of our architectures is the learning of filter pairs that are logically AND combined, not the pairwise multiplicative interactions, indicating that end-stopping is indeed a crucial factor as to why E-nets perform better than their baseline CNNs.

5.3 E-neurons are hyperselective and more robust

The curvature analysis of iso-response contours [GH12] is an essential tool in works investigating the intersection of biological vision and computer vision [VF17; Pai+20]. Neurons can have a hyperselective property, possibly leading to increased robustness against adversarial attacks [Pai+20]. Adopting the curvature analysis in Section 3.7.1, we could show analytically that E-blocks are indeed hyperselective, depending on the filter-pair angle γ. This outcome is further backed by our empirical results regarding the entropy of E-neurons in Section 4.4.1. Given that our models contain hyperselective neurons, our results did show that E-nets were more robust against adversarial attacks and JPEG compression artifacts (see Section 4.4.3.1 and Section 4.5.1.2).

Thus, regarding the end-stopped and hyperselective properties of E-blocks, we could verify that our models display the intended bio-inspired behavior. In contrast to many works in the biological vision field that work with less optimized models, we could provide practical methods to improve SOTA CNNs on ImageNet, Cifar-10, and medical image segmentation.

5.4 E-nets are easy to create

In Section 4.3 and Section 4.5.1, we propose a simple design rule to transform baseline networks into E-nets that require no additional hyperparameter tuning to obtain a better generalization for image classification. Our method improved several different architectures, e.g., the DenseNet [Hua+17], the MobileNet-V2 [San+18a], or the

PyramidNet [HKK17], indicating that other architectures may also perform better when transforming them into E-nets. The advantage of our method is that an E-net is easy to implement, easy to test, and may yield model improvements in terms of generalization, robustness, and parameter efficiency. Thus, we provide practitioners with a new and practical approach to enhance their CNN-based solutions.

Compared with other works on CNN architectures, the substitution approach we employ is often overlooked. In general, a specific block 'X' is constructed, and a model architecture containing only block 'X' is optimized in terms of input resolution, depth, and width (see, for example, the EfficientNet [TL19a] and others [San+18a; He+16]). We think block substitution is a viable alternative search direction for better architectures. However, as a downside, the number of possible solutions is often vast, and it is unclear if the global optimum can be found. We observed the same difficulty in our results: the proposed design rule works well, but, as seen in Section 4.7, even better solutions often exist.

5.5 Open questions

The optimal positioning of E-blocks is an entirely new hyperparameter that strongly affects model quality. As the discrete search space grows exponentially with the number of blocks, solving the positioning problem may be the most crucial next research step for E-nets. As the results in Section 4.7 show, placing the E-blocks-M correctly has a strong potential to improve E-nets.

We presented a genetic algorithm as a first approach to finding better E-nets-M. However, this method still involves training many models and is thus only feasible on smaller datasets. Alternatively, one could attempt to learn which blocks should be substituted and which should remain convolution blocks by having a neuron that provides a convex combination of the linear- and AND combined output:

$$\mathbf{T}_2[i,j,m] = \lambda\zeta(\phi(\mathbf{x}^T\mathbf{v}), \phi(\mathbf{x}^T\mathbf{g})) + (1-\lambda)\phi(\mathbf{x}^T\mathbf{v}). \tag{5.1}$$

Here, the variable $\lambda \in [0,1]$ determines the preferred output. Via regularization, λ can be forced to be either zero or one. This approach could, in principle, be very efficient in determining which blocks should display which behavior. So far, in our first attempts, we had no issues training the models, but we did not obtain E-nets that were better than the baseline.

Let us assume we would have an efficient approach to determine the optimal E-block positioning for any given dataset. This approach could be used to better understand the data, for example, if we could determine certain features that correlate with a specific optimal positioning. Accordingly, one may better understand the differences between types of images, e.g., differences in depicted objects, styles, and modalities. Maybe there are even datasets where deep E-blocks, located at the end of the feature

hierarchy, are beneficial. What properties would define such a dataset? Would there even be one? Because, at least when working with natural images, such a dataset seems unlikely, as end-stopping rather happens in earlier stages of the feature hierarchy.

However, CNNs are not restricted to only processing images. E.g., CNNs often match the performance of recurrent networks when applied to sequential data such as text or audio [BKK18]. Accordingly, investigating E-block positioning of different input types may lead to interesting insights. For example, one could determine if end-stopping is more beneficial in a different position when comparing audio data to visual data.

Apart from where to position E-blocks, we are not sure if we have found the best E-block design yet. Our results in Section 4.8 show that the design choices beneficial for image classification can be suboptimal for image segmentation. It is yet to be determined if an optimal block structure that is the best for all different image tasks exists. However, keeping the *no-free-lunch*-theorem [WM97] in mind, it is more likely that each task has its own optimal E-block design.

Using the E-layer-L to compute large products of filters efficiently is another area for promising research, especially on CNN robustness. Multiplying more than two oriented filters may yield a neuron that is more hyperselective because there can exist more directions orthogonal to the optimal stimulus that can reduce the neuron's output. Increased selectivity may yield increased robustness – but this hypothesis needs to be appropriately investigated.

Finally, the fact that there are still challenges left should not be seen too pessimistically: so far, we have found methods and models that work well – but we also found evidence that they could work even better. Thus, we consider the study of AND combinations of filter pairs in deep networks to be a promising research field for the future.

5.6 Conclusion

We reached our initial goal of creating novel improved CNNs using a bio-inspired inductive bias. Our results show that the biological behaviors we intended to model could indeed be observed in E-nets and that this property benefits generalization and efficiency. Furthermore, E-neurons are hyperselective and thus increase the robustness of E-nets, for example, against adversarial attacks. Our methods are practical and easy to implement and can become a standard optimization step to further improve CNNs in vision tasks. Moreover, although we have already improved many established CNN architectures in the course of this work, our results strongly indicate that the success story of E-nets might not end here.

Bibliography

[Ber+09] J Bergstra, G Desjardins, P Lamblin, and Y Bengio. "Quadratic polynomials learn better image features (Technical Report 1337)". In: *Département d'Informatique et de Recherche Opérationnelle, Université de Montréal* (2009).

[Ber19] David Berga. "Understanding Eye Movements: Psychophysics and a Model of Primary Visual Cortex". PhD thesis. July 2019.

[Bia+18] Simone Bianco, Luigi Celona, Paolo Napoletano, and Raimondo Schettini. "On the use of deep learning for blind image quality assessment". In: *Signal, Image and Video Processing* 12.2 (2018), pp. 355–362.

[BKK18] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling". In: *arXiv preprint arXiv:1803.01271* (2018).

[Bos+16] Sebastian Bosse, Dominique Maniry, Thomas Wiegand, and Wojciech Samek. "A deep neural network for image quality assessment". In: *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2016, pp. 3773–3777.

[Bra00] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* 25 (11 2000), pp. 120–123.

[BW00] E. Barth and A. B. Watson. "A Geometric Framework for Nonlinear Visual Coding". In: *Optics Express* 7.4 (2000), pp. 155–165.

[BW06] Pietro Berkes and Laurenz Wiskott. "On the analysis and interpretation of inhomogeneous quadratic forms as receptive fields". In: *Neural computation* 18.8 (2006), pp. 1868–1895.

[BZ98] E. Barth and C. Zetzsche. "Endstopped operators based on iterated nonlinear center-surround inhibition". In: *Human Vision and Electronic Imaging*. Vol. 3299. International Society for Optics and Photonics. 1998, pp. 67–78.

[CAC12] Burak Ömür Cakir, Peter Adamson, and Cemal Cingi. "Epidemiology and economic burden of nonmelanoma skin cancer." In: *Facial plastic surgery clinics of North America* 20.4 (2012), pp. 419–422.

[Cho+16] Aruni Roy Chowdhury, Tsung-Yu Lin, Subhransu Maji, and Erik Learned-Miller. "One-to-many face recognition with bilinear cnns". In: *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2016, pp. 1–9.

[Cho17] François Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.

[Chr+20] Grigorios G Chrysos, Stylianos Moschoglou, Giorgos Bouritsas, Yannis Panagakis, Jiankang Deng, and Stefanos Zafeiriou. "P-nets: Deep Polynomial Neural Networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 7325–7335.

[Cir+12] Dan Ciresan, Alessandro Giusti, Luca Gambardella, and Jürgen Schmidhuber. "Deep neural networks segment neuronal membranes in electron microscopy images". In: *Advances in neural information processing systems* 25 (2012), pp. 2843–2851.

[CJ20] Davide Chicco and Giuseppe Jurman. "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation". In: *BMC genomics* 21.1 (2020), p. 6.

[Cod+18] N. C. F. Codella, D. Gutman, M. E. Celebi, B. Helba, M. A. Marchetti, S. W. Dusza, A. Kalloo, K. Liopyris, N. Mishra, H. Kittler, and A. Halpern. "Skin lesion analysis toward melanoma detection: a challenge at the 2017 international symposium on biomedical imaging (ISBI), hosted by the international skin imaging collaboration (ISIC)". In: *IEEE 15th international symposium on biomedical imaging*. `https://challenge.isic-archive.com/data` (accessed April 8, 2021). 2018, pp. 168–172.

[CSDS16] Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. "Capacity and trainability in recurrent neural networks". In: *stat* 1050 (2016), p. 29.

[CTK17] Zhengxue Cheng, Masaru Takeuchi, and Jiro Katto. "A pre-saliency map based blind image quality assessment via convolutional neural networks". In: *2017 IEEE International Symposium on Multimedia (ISM)*. IEEE. 2017, pp. 77–82.

[CV95] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[Dai+21] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. "Coatnet: Marrying convolution and attention for all data sizes". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 3965–3977.

[Den+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[Die00] Thomas G. Dietterich. "Ensemble Methods in Machine Learning". In: *Multiple Classifier Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–15.

[Din+21] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. "Repvgg: Making vgg-style convnets great again". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 13733–13742.

[Din+22] Mingyu Ding, Bin Xiao, Noel Codella, Ping Luo, Jingdong Wang, and Lu Yuan. "DaViT: Dual Attention Vision Transformers". In: *arXiv preprint arXiv:2204.03645* (2022).

[Dos+20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[FC19] Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *International Conference on Learning Representations*. 2019.

[Fra+12] M. M. Fraz, P. Remagnino, A. Hoppe, B. Uyyanonvara, A. R. Rudnicka, C. G. Owen, and S. A. Barman. "An ensemble classification-based approach applied to retinal blood vessel segmentation". In: *IEEE transactions on biomedical engineering* 59.9 (2012), pp. 2538–2548.

[Fra+12] Muhammad Moazam Fraz, Paolo Remagnino, Andreas Hoppe, Bunyarit Uyyanonvara, Alicja R Rudnicka, Christopher G Owen, and Sarah A Barman. "Blood vessel segmentation methodologies in retinal images – a survey". In: *Computer methods and programs in biomedicine* 108.1 (2012), pp. 407–433.

[Fuk80] K Fukushima. "Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position." In: *Biological Cybernetics* 36.4 (1980), pp. 193–202.

[Gao+16] Yang Gao, Oscar Beijbom, Ning Zhang, and Trevor Darrell. "Compact bilinear pooling". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 317–326.

[GB] Philipp Grüning and Erhardt Barth. *Nuclei Segmentation using Sequential Multi-Objective U-Nets*.

[GB15] Deepti Ghadiyaram and Alan C Bovik. "Massive online crowdsourced study of subjective and objective picture quality". In: *IEEE Transactions on Image Processing* 25.1 (2015), pp. 372–387.

[GB21a] Philipp Grüning and Erhardt Barth. "Bio-inspired Min-Nets Improve the Performance and Robustness of Deep Networks". In: *SVRHM 2021 Workshop@ NeurIPS*. 2021. URL: `https://openreview.net/forum?id=zxxdFLB8F24`.

[GB21b] Philipp Grüning and Erhardt Barth. "FP-Nets for Blind Image Quality Assessment". In: *Journal of Perceptual Imaging* 4.1 (2021), pp. 10402–1–10402–13.

[GB23] Philipp Grüning and Erhardt Barth. "Efficient coding in human vision as a useful bias in computer vision and machine learning (under review)". In: *Journal of Perceptual Imaging* (2023).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, Massachusetts: MIT Press, 2016.

[GH12] Tim Gollisch and Andreas VM Herz. "The iso-response method: measuring neuronal stimulus integration with closed-loop experiments". In: *Frontiers in neural circuits* 6 (2012), p. 104.

[Gil77] Charles D Gilbert. "Laminar differences in receptive field properties of cells in cat primary visual cortex". In: *The Journal of physiology* 268.2 (1977), pp. 391–421.

[GLL18] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. "DropBlock: a regularization method for convolutional networks". In: *Advances in neural information processing systems*. Vol. 31. `https://github.com/miguelvr/dropblock` (accessed April 10, 2021). 2018.

[GM20] Philipp Grüning and Amir Madany Mamlouk. "Deep Neural-Gas Clustering for Instance Segmentation across Imaging Experiments". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–8.

[GMB20a] Philipp Grüning, Thomas Martinetz, and Erhardt Barth. "Feature Products Yield Efficient Networks". In: *arXiv preprint arXiv:2008.07930* (2020).

[GMB20b] Philipp Grüning, Thomas Martinetz, and Erhardt Barth. "Log-Nets: Logarithmic Feature-Product Layers Yield More Compact Networks". In: *Artificial Neural Networks and Machine Learning – ICANN 2020*. Cham: Springer International Publishing, 2020, pp. 79–91.

[GMB22] Philipp Grüning, Thomas Martinetz, and Erhardt Barth. “FP-nets as novel deep networks inspired by vision”. In: *Journal of Vision* 22.1 (2022), pp. 8–8.

[Goo83] Nelson Goodman. *Fact, fiction, and forecast*. Harvard University Press, 1983.

[Gor22] Sonja Gorjanc. *Elliptic, Hyperbolic, Parabolic and Planar Points of a Surface*. `https://www.grad.hr/itproject_math/Links/sonja/gausseng/ehpp/ehpp.html` (accessed December 29, 2022). 2022.

[Gra17] Gavia Gray. *sequential-imagenet-dataloader*. `https://github.com/BayesWatch/sequential-imagenet-dataloader` (accessed February 20, 2021). 2017.

[GRMB72] Charles G Gross, CE de Rocha-Miranda, and DB Bender. “Visual properties of neurons in inferotemporal cortex of the Macaque.” In: *Journal of neurophysiology* 35.1 (1972), pp. 96–111.

[Gro02] Charles G Gross. “Genealogy of the “grandmother cell””. In: *The Neuroscientist* 8.5 (2002), pp. 512–518.

[Grü+20a] Philipp Grüning, Alex Palumbo, Svenja Kim Landt, Lara Heckmann, Leslie Brackhagen, Marietta Zille, and Amir Madany Mamlouk. “Robust and Markerfree in vitro Axon Segmentation with CNNs”. In: *International Conference on Wireless Mobile Communication and Healthcare*. Springer. 2020, pp. 274–284.

[Grü+20b] Philipp Grüning, Alex Palumbo, Marietta Zille, Erhardt Barth, and Amir Madany Mamlouk. “A task-dependent active learning method for axon segmentation with CNNs”. In: *Proceedings on Automation in Medical Engineering* 1.1 (2020), pp. 025–025.

[Grü+21] Philipp Grüning, Falk Nette, Noah Heldt, Ana Cristina Guerra de Souza, and Erhardt Barth. “Direct Inference of Cell Positions using Lens-Free Microscopy and Deep Learning”. In: *Medical Imaging with Deep Learning*. PMLR. 2021, pp. 219–227.

[GSS14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: *arXiv preprint arXiv:1412.6572* (2014).

[Guo+21] Changlu Guo, Márton Szemenyei, Yugen Yi, Wenle Wang, Buer Chen, and Changqi Fan. “Sa-unet: Spatial attention u-net for retinal vessel segmentation”. In: *2020 25th international conference on pattern recognition (ICPR)*. `https://github.com/clguo/SA-UNet` (accessed April 10, 2021). IEEE. 2021, pp. 1236–1242.

[Has+16] Seyyed Hossein HasanPour, Mohammad Rouhani, Mohsen Fayyaz, and Mohammad Sabokrou. "Lets keep it simple, using simple architectures to outperform deeper and more complex architectures". In: *arXiv preprint arXiv:1608.06037* (2016).

[Has+20] Md. Kamrul Hasan, Lavsen Dahal, Prasad N. Samarakoon, Fakrul Islam Tushar, and Robert Martí. "DSNet: automatic dermoscopic skin lesion segmentation". In: *Computers in biology and medicine* 120 (2020), p. 103738.

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[Hes+19] Mohammad Hesam Hesamian, Wenjing Jia, Xiangjian He, and Paul Kennedy. "Deep learning techniques for medical image segmentation: achievements and challenges". In: *Journal of digital imaging* 32.4 (2019), pp. 582–596.

[HKK17] Dongyoon Han, Jiwhan Kim, and Junmo Kim. "Deep pyramidal residual networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 5927–5935.

[Hos+20] Vlad Hosu, Hanhe Lin, Tamas Sziranyi, and Dietmar Saupe. "KonIQ-10k: An ecologically valid database for deep learning of blind image quality assessment". In: *IEEE Transactions on Image Processing* 29 (2020), pp. 4041–4056.

[How+17] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[How+18] Jeremy Howard et al. *imagenet-fast*. `https://github.com/fastai/imagenet-fast/tree/master/imagenet_nv` (accessed February 20, 2021). 2018.

[HSS18] Jie Hu, Li Shen, and Gang Sun. "Squeeze-and-excitation networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7132–7141.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[Hua+17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[Hua+19] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, and Zhifeng Chen. “Gpipe: Efficient training of giant neural networks using pipeline parallelism”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 103–112.

[HW65] David H Hubel and Torsten N Wiesel. “Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat”. In: *Journal of neurophysiology* 28.2 (1965), pp. 229–289.

[HW68] David H. Hubel and Torsten N. Wiesel. “Receptive fields and functional architecture of monkey striate cortex”. In: *The journal of physiology* 195.1 (1968), pp. 215–243.

[Ian+16] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size”. In: *arXiv preprint arXiv:1602.07360* (2016).

[IS15] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *International Conference on Machine Learning*. 2015, pp. 448–456.

[Jad20] Shruti Jadon. “A survey of loss functions for semantic segmentation”. In: *2020 IEEE conference on computational intelligence in bioinformatics and computational biology*. 2020, pp. 1–7.

[Jäh93] Bernd Jähne. *Spatio-temporal image processing: theory and scientific applications*. Springer, 1993.

[JL43] McCulloch JL. “A logical calculus of ideas immanent in nervous activity”. In: *Bull. of Math. Biophysics* 5 (1943), pp. 115–133.

[Kan+14] Le Kang, Peng Ye, Yi Li, and David Doermann. “Convolutional neural networks for no-reference image quality assessment”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1733–1740.

[KB15] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *ICLR (Poster)*. 2015.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. *Learning multiple layers of features from tiny images*. Tech. rep. University of Toronto, 2009.

[Kim+17] Jongyoo Kim, Hui Zeng, Deepti Ghadiyaram, Sanghoon Lee, Lei Zhang, and Alan C Bovik. "Deep convolutional neural models for picture-quality prediction: Challenges and solutions to data-driven image quality assessment". In: *IEEE Signal processing magazine* 34.6 (2017), pp. 130–141.

[Kim20] Hoki Kim. "Torchattacks: A pytorch repository for adversarial attacks". In: *arXiv preprint arXiv:2010.01950* (2020). `https://github.com/Harry24k/adversarial-attacks-pytorch` (accessed February 20, 2021).

[KNH10] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. *CIFAR-10 (Canadian Institute for Advanced Research)*. `http://www.cs.toronto.edu/~kriz/cifar.html` (accessed January 3, 2018). 2010.

[KNL18] Jongyoo Kim, Anh-Duc Nguyen, and Sanghoon Lee. "Deep CNN-based blind image quality predictor". In: *IEEE transactions on neural networks and learning systems* 30.1 (2018), pp. 11–24.

[Kol+20] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. "Big transfer (bit): General visual representation learning". In: *European conference on computer vision*. Springer. 2020, pp. 491–507.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[kua20] kuangliu. *pytorch-cifar*. `https://github.com/kuangliu/pytorch-cifar` (accessed March 24, 2020). 2020.

[Kum+19] Neeraj Kumar, Ruchika Verma, Deepak Anand, Yanning Zhou, Omer Fahri Onder, Efstratios Tsougenis, Hao Chen, Pheng-Ann Heng, Jiahui Li, Zhiqiang Hu, Yunzhi Wang, Navid Alemi Koohbanani, Mostafa Jahanifar, Neda Zamani Tajeddin, Ali Gooya, et al. "A multi-organ nucleus segmentation challenge". In: *IEEE transactions on medical imaging* 39.5 (2019), pp. 1380–1391.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[LC10] Eric Cooper Larson and Damon Michael Chandler. "Most apparent distortion: full-reference image quality assessment and the role of strategy". In: *Journal of electronic imaging* 19.1 (2010), p. 011006.

[LeC+89] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.

[LeC+98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[LH17] Ilya Loshchilov and Frank Hutter. “SGDR: Stochastic Gradient Descent with Warm Restarts”. In: *International Conference on Learning Representations*. 2017.

[Li+17a] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. “Pruning Filters for Efficient ConvNets”. In: *International Conference on Learning Representations*. 2017.

[Li+17b] Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. “Factorized bilinear models for image recognition”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2079–2087.

[Li+21] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. “A survey of convolutional neural networks: analysis, applications, and prospects”. In: *IEEE transactions on neural networks and learning systems* (2021).

[Lin70] Seppo Linnainmaa. “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors”. PhD thesis. Master’s Thesis (in Finnish), Univ. Helsinki, 1970.

[LRM15] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. “Bilinear CNN models for fine-grained visual recognition”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1449–1457.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[Lu20] Lu Lu. “Dying ReLU and Initialization: Theory and Numerical Examples”. In: *Communications in Computational Physics* 28.5 (2020), pp. 1671–1706.

[LWB17] Xialei Liu, Joost van de Weijer, and Andrew D Bagdanov. “Rankiqa: Learning from rankings for no-reference image quality assessment”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1040–1049.

[LZY21] Duo Li, Aojun Zhou, and Anbang Yao. *mobilenetv2.pytorch*. `https://github.com/d-li14/mobilenetv2.pytorch` (accessed February 20, 2021). 2021.

[Ma+17] Kede Ma, Wentao Liu, Kai Zhang, Zhengfang Duanmu, Zhou Wang, and Wangmeng Zuo. "End-to-end blind image quality assessment using deep neural networks". In: *IEEE Transactions on Image Processing* 27.3 (2017), pp. 1202–1213.

[Mar80] S Marĉelja. "Mathematical description of the responses of simple cortical cells". In: *JOSA* 70.11 (1980), pp. 1297–1300.

[MB00] Cicero Mota and Erhardt Barth. "On the uniqueness of curvature features". In: *Dynamische Perzeption*. Vol. 9. 2000, pp. 175–178.

[Men+13] T. Mendonça, P. M. Ferreira, J. S. Marques, A. R. S. Marcal, and J. Rozeira. "PH2 – A dermoscopic image database for research and benchmarking". In: *35th annual international conference of the IEEE engineering in medicine and biology society*. `https://www.fc.up.pt/addi/ph2%20database.html` (accessed April 10, 2021). 2013, pp. 5437–5440.

[Min+22] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. "Image Segmentation Using Deep Learning: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.7 (2022), pp. 3523–3542. DOI: `10.1109/TPAMI.2021.3059968`.

[MK90] Bartlett W Mel and Christof Koch. "Sigma-pi learning: On radial basis functions and cortical associative learning". In: *Advances in neural information processing systems*. 1990, pp. 474–481.

[MMB12] Anish Mittal, Anush Krishna Moorthy, and Alan Conrad Bovik. "No-reference image quality assessment in the spatial domain". In: *IEEE Transactions on image processing* 21.12 (2012), pp. 4695–4708.

[MP43] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[MTT78] J Anthony Movshon, Ian D Thompson, and David J Tolhurst. "Spatial summation in the receptive fields of simple cells in the cat's striate cortex." In: *The Journal of physiology* 283.1 (1978), pp. 53–77.

[Nie+04] M. Niemeijer, J. J. Staal, B. Ginneken, M. Loog, and M. D. Abramoff. "DRIVE: digital retinal images for vessel extraction". In: *Methods for evaluating segmentation and indexing techniques dedicated to retinal ophthalmology* (2004). `https://drive.grand-challenge.org/` (accessed April 10, 2021).

[OF96] Bruno A Olshausen and David J Field. "Emergence of simple-cell receptive field properties by learning a sparse code for natural images". In: *Nature* 381.6583 (1996), pp. 607–609.

[Okt+18] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, et al. "Attention u-net: Learning where to look for the pancreas". In: *arXiv preprint arXiv:1804.03999* (2018).

[ÖÖ20] Şaban Öztürk and Umut Özkaya. "Skin lesion segmentation with improved convolutional neural network". In: *Journal of digital imaging* 33 (2020), pp. 958–970.

[Pai+20] Dylan M. Paiton, Charles G. Frye, Sheng Y. Lundquist, Joel D. Bowen, Ryan Zarcone, and Bruno A. Olshausen. "Selectivity and robustness of sparse coding networks". In: *Journal of Vision* 20.12 (Nov. 2020), pp. 10–10. ISSN: 1534-7362. DOI: 10.1167/jov.20.12.10.

[Pal+21] Alex Palumbo, Philipp Grüning, Svenja Kim Landt, Lara Eleen Heckmann, Luisa Bartram, Alessa Pabst, Charlotte Flory, Maulana Ikhsan, Sören Pietsch, Reinhard Schulz, Christopher Kren, Norbert Koop, Johannes Boltze, Amir Madany Mamlouk, and Zille Marietta. "Deep learning to decipher the progression and morphology of axonal degeneration". In: *Cells* 10.10 (2021), p. 2539.

[Pas+] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, et al. *Pytorch ImageNet example*. `https://github.com/pytorch` (accessed June 2, 2020).

[Pas+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035.

[PC15] Soo-Chang Pei and Li-Heng Chen. "Image quality assessment using human visual DOG model fused with random forest". In: *IEEE Transactions on Image Processing* 24.11 (2015), pp. 3282–3292.

[Pha+21] Hieu Pham, Zihang Dai, Qizhe Xie, and Quoc V Le. "Meta pseudo labels". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 11557–11568.

[Pon+15] Nikolay Ponomarenko, Lina Jin, Oleg Ieremeiev, Vladimir Lukin, Karen Egiazarian, Jaakko Astola, Benoit Vozel, Kacem Chehdi, Marco Carli, Federica Battisti, et al. "Image database TID2013: Peculiarities, results and perspectives". In: *Signal processing: Image communication* 30 (2015), pp. 57–77.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional networks for biomedical image segmentation". In: *International conference on medical image computing and computer-assisted intervention*. 2015, pp. 234–241.

[RHM+86] David E Rumelhart, Geoffrey E Hinton, James L McClelland, et al. "A general framework for parallel distributed processing". In: *Parallel distributed processing: Explorations in the microstructure of cognition* 1.45-76 (1986), p. 26.

[RHW85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[Rib+20] Edgar Riba, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski. "Kornia: an open source differentiable computer vision library for pytorch". In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2020, pp. 3674–3683.

[Rid+21] Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. "ImageNet-21K Pretraining for the Masses". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*. 2021.

[Ros57] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[Ros77] David Rose. "Responses of single units in cat visual cortex to moving bars of light as a function of bar length". In: *The Journal of physiology* 271.1 (1977), p. 1.

[RR02] Colin Reeves and Jonathan E Rowe. *Genetic algorithms: principles and perspectives: a guide to GA theory*. Vol. 20. Springer Science & Business Media, 2002.

[Rus+05] Nicole C Rust, Odelia Schwartz, J Anthony Movshon, and Eero P Simoncelli. "Spatiotemporal elements of macaque v1 receptive fields". In: *Neuron* 46.6 (2005), pp. 945–956.

[Rus+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[San+18a] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. "MobileNetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[San+18b] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Mądry. "How does batch normalization help optimization?" In: *Advances in neural information processing systems*. 2018, 2488—2498.

[SGS15] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. "Training very deep networks". In: *Advances in neural information processing systems* 28 (2015).

[Sil75] AM Sillito. "The contribution of inhibitory mechanisms to the receptive field properties of neurones in the striate cortex of the cat." In: *The Journal of physiology* 250.2 (1975), pp. 305–329.

[Spr+15] J Springenberg, Alexey Dosovitskiy, Thomas Brox, and M Riedmiller. "Striving for Simplicity: The All Convolutional Net". In: *ICLR (workshop track)*. 2015.

[SSB06] Hamid R Sheikh, Muhammad F Sabir, and Alan C Bovik. "A statistical evaluation of recent full reference image quality assessment algorithms". In: *IEEE Transactions on image processing* 15.11 (2006), pp. 3440–3451.

[ST] Robert Stojnic and Ross Taylor. *Cifar-10 Leaderboard*. `https://paperswithcode.com/sota/image-classification-on-cifar-10` (accessed May 9, 2022).

[ST20] Robert Stojnic and Ross Taylor. *ImageNet Leaderboard*. `https://paperswithcode.com/sota/image-classification-on-imagenet` (accessed March 24, 2020). 2020.

[SZ14] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[Sze10] Richard Szeliski. *Computer vision: algorithms and applications*. Springer science & business media, 2010.

[Sze+14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. "Intriguing properties of neural networks". In: *2nd International Conference on Learning Representations, ICLR 2014*. 2014.

[Sze+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[Tan+19] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. "Mnasnet: Platform-aware neural architecture search for mobile". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2820–2828.

[TL19a] Mingxing Tan and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114.

[TL19b] Mingxing Tan and Quoc V Le. "Mixconv: Mixed depthwise convolutional kernels". In: *CoRR, abs/1907.09595* (2019).

[Tou+21] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. "Going deeper with image transformers". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 32–42.

[TSS22] Lukas Tuggener, Jürgen Schmidhuber, and Thilo Stadelmann. "Is it enough to optimize cnn architectures on imagenet?" In: *Frontiers in Computer Science* 4.1041703 (2022).

[Tu+21] Zhengzhong Tu, Yilin Wang, Neil Birkbeck, Balu Adsumilli, and Alan C Bovik. "UGC-VQA: Benchmarking blind video quality assessment for user generated content". In: *IEEE Transactions on Image Processing* 30 (2021), pp. 4449–4464.

[UGL17] Evgeniya Ustinova, Yaroslav Ganin, and Victor Lempitsky. "Multi-region bilinear convolutional neural networks for person re-identification". In: *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE. 2017, pp. 1–6.

[UVL16] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Instance normalization: The missing ingredient for fast stylization". In: *arXiv preprint arXiv:1607.08022* (2016).

[Vap99] Vladimir N Vapnik. "An overview of statistical learning theory". In: *IEEE transactions on neural networks* 10.5 (1999), pp. 988–999.

[Vas+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[VD18] Tom Veniat and Ludovic Denoyer. “Learning time/memory-efficient deep architectures with budgeted super networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3492–3500.

[Vel79] Flavio R. Velasco. *Thresholding using the ISODATA clustering algorithm*. Tech. rep. Maryland university college park computer science center, 1979.

[VF17] Kedarnath P Vilankar and David J Field. “Selectivity, hyperselectivity, and the tuning of V1 neurons”. In: *Journal of vision* 17.9 (2017), pp. 9–9.

[Vig+11] Eleonora Vig, Michael Dorr, Thomas Martinetz, and Erhardt Barth. “Intrinsic dimensionality predicts the saliency of natural dynamic scenes”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.6 (2011), pp. 1080–1091.

[Vol59] Vito Volterra. *Theory of functionals and of integral and integro-differential equations*. Mineola: Dover Publications Inc., 1959.

[VSS18] Domonkos Varga, Dietmar Saupe, and Tamás Szirányi. “DeepRN: A content preserving deep architecture for blind image quality assessment”. In: *2018 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE. 2018, pp. 1–6.

[VWB16] Andreas Veit, Michael J Wilber, and Serge Belongie. “Residual networks behave like ensembles of relatively shallow networks”. In: *Advances in neural information processing systems* 29 (2016).

[Wan+20] Qilong Wang, Banggu Wu, Pengfei Zhu, Peihua Li, Wangmeng Zuo, and Qinghua Hu. “ECA-Net: Efficient Channel Attention for Deep Convolutional Neural Networks”. In: June 2020, pp. 11531–11539. DOI: 10.1109/CVPR42600.2020.01155.

[Wat85] Satosi Watanabe. *Pattern recognition: human and mechanical*. John Wiley & Sons, Inc., 1985.

[Weg21] Tilman Wegener. “Feature Product Networks for Medical Image Segmentation”. MA thesis. University of Lübeck, 2021.

[WH18] Yuxin Wu and Kaiming He. “Group normalization”. In: *Proceedings of the european conference on computer vision*. 2018, pp. 3–19.

[WM97] David H Wolpert and William G Macready. “No free lunch theorems for optimization”. In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

[Wu+16] Yuxin Wu et al. *Tensorpack*. https://github.com/tensorpack/tensorpack/tree/master/examples/ResNet (accessed February 20, 2020). 2016.

[XY17] Lingxi Xie and Alan Yuille. "Genetic cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 1379–1388.

[Yan+22] Xuecan Yang, Sumanta Chaudhuri, Laurence Likforman, and Lirida Naviner. "MinConvNets: A new class of multiplication-less Neural Networks". In: *2022 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2022, pp. 881–885.

[Ye+12] Peng Ye, Jayant Kumar, Le Kang, and David Doermann. "Unsupervised feature learning framework for no-reference image quality assessment". In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 1098–1105.

[Yin+19] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. "Nas-bench-101: Towards reproducible neural architecture search". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7105–7114.

[Yua+10] Lin Yuanqing, Lv Fengjun, Yang Shenghuo Zhuand Ming, Cour Timothee, Yu Kai, Cao LiangLiang, Li Zhen, Tsai Min-Hsuan, Zhou Xi, Huang Thomas, and Zhang Tong. *ImageNet classification: fast descriptor coding and large-scale SVM training*. `https://image-net.org/static_files/files/ILSVRC2010_NEC-UIUC.pdf` (accessed October 26, 2022). 2010.

[ZB90] Christoph Zetzsche and Erhardt Barth. "Fundamental limits of linear filters in the visual processing of two-dimensional signals". In: *Vision Research* 30 (1990), pp. 1111–1117.

[ZBW93] Christoph Zetzsche, Erhardt Barth, and Bernhard Wegmann. "The importance of intrinsically two-dimensional image features in biological vision and picture coding". In: *Digital Images and Human Vision*. MIT Press, Oct. 1993, pp. 109–38.

[ZF14] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[Zha+18a] Weixia Zhang, Kede Ma, Jia Yan, Dexiang Deng, and Zhou Wang. "Blind image quality assessment using a deep bilinear convolutional neural network". In: *IEEE Transactions on Circuits and Systems for Video Technology* (2018).

[Zha+18b] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. "Shufflenet: An extremely efficient convolutional neural network for mobile devices". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 6848–6856.

[Zho+17] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. “Places: A 10 million image database for scene recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* 40.6 (2017), pp. 1452–1464.

[ZK16] Sergey Zagoruyko and Nikos Komodakis. “Wide Residual Networks”. In: *British Machine Vision Conference 2016*. British Machine Vision Association. 2016.

[ZM20] Guangtao Zhai and Xiongkuo Min. “Perceptual image quality assessment: a survey”. In: *Science China Information Sciences* 63.11 (2020), pp. 1–52.

[Zou+17] Georgios Zoumpourlis, Alexandros Doumanoglou, Nicholas Vretos, and Petros Daras. “Non-linear convolution filters for CNN-based learning”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 4761–4769.

Appendix A

Implementation details

All experiments were conducted using the PyTorch deep learning framework [Pas+19].

A.1 Residual connections

For the residual connections in Equation 3.4 on page 32 some additional computations are needed if the dimensions of $\mathbf{T}_0$ and $\mathbf{T}_3$ differ. In case that d_{out} is greater than d_{in}, zero padding is used to match the dimension of the feature maps. If d_{in} is greater than d_{out}, an additional linear combination is learned to reduce the number of feature maps. If the E-block's stride s is greater than one, $\mathbf{T}_0$ is subsampled by average pooling. For more implementation details regarding residual connections, see Han, Kim, and Kim [HKK17]. We provide more general information about residual connections in Section 2.3.2.

A.2 Cifar-10

A.2.1 Data augmentation

During training, the images were flipped horizontally with a 50% chance. Furthermore, they were padded with four pixels on each side and randomly cropped using a 32×32 window. After scaling each pixel from $[0, 255]$ to $[0, 1]$ (dividing by 255), we normalized each channel of an input image by subtracting the mean μ pixel and dividing by the standard deviation σ. [1] For the E-nets-I and E-nets-R, we used the ImageNet mean $\mu_{imNet}(0.485, 0.456, 0.406)$ and standard deviation $\sigma_{imNet}(0.229, 0.224, 0.225)$ for the three input channels, respectively. For the E-net-M, we used the Cifar-10 mean and standard deviation: $\mu_{cifar} = (0.491, 0.482, 0.447)$ and $\sigma_{cifar} = (0.247, 0.244, 0.262)$. During testing, the images were only normalized. Note that the choice of normalization values did not alter the results significantly.

[1] Having mean-free values with a smaller range is usually good practice when training neural networks: it can prevent exploding gradients and, since the network's initial separation boundary is in most cases initially located near the origin, more gradient information is present if the majority of data points are located near the origin.

A.2.2 Hyperparameters

ResNet We used the following hyperparameters for all ResNet architectures, including the PyrNet, and all E-nets based on those networks. Each model was trained for 200 epochs with a learning rate of 0.1, a weight decay of 0.0001, and using SGD with a momentum of 0.9. The batch size was 128. After 100 and 150 epochs, the learning rate was divided by 10.

DenseNet We trained the DenseNets and E-DenseNets-M for 300 epochs, where the initial learning rate of 0.1 was divided by ten after 150 and 225 epochs. Instead of 128, a batch size of 64 was used. In addition, we used the Nesterov momentum. We trained each network with three different random seeds. All other parameters were equal to the ResNet experiments.

E-net-L Each model was trained for 120 epochs with the Adam optimizer [KB15] and a batch size of 128. For each 40th step, the initial learning rate of 0.001 was divided by 10.

A.3 ImageNet

A.3.1 Data augmentation

During training, each input image was cropped with a random crop size between 8% and 100% of the input size and a random aspect ration in $[\frac{3}{4}, \frac{4}{3}]$ which was subsequently resized to 224×224. Next, the cropped image was flipped horizontally with a 50% chance. Furthermore, color jitter was added. Then, each pixel value was divided by 255 and normalized by subtracting the ImageNet mean μ_{imNet} and dividing it by the ImageNet standard deviation σ_{imNet} (see A.2.1).

During inference, the image was resized to 255×255, and a center crop was extracted. Again the image was normalized as previously described. For more information, see Fastai's repository [How+18] and the ImageNet training example of the PyTorch repository [Pas+].

A.3.2 Hyperparameters

E-ResNet-50-I We trained the model for 100 epochs, starting with a learning rate of 0.1 that was reduced to 0.01 after 33 epochs and to 0.001 after 66 epochs. The batch size was 256, and we used a weight decay of 0.0001. We report the validation error of the last epoch and compare our result to the validation errors of Pytorch's [Pas+19] pre-trained ResNets. We trained the model on four GeForce RTX 2080 for 64 hours. We used the data augmentation scheme described in Section A.3.1

E-MobileNet-V2-R The E-MobileNet-V2-R was trained from scratch with SGD for 150 epochs and with a batch size of 256. The initial learning rate of 0.05 was altered according to a cosine scheduling [LH17], see Li et al.'s repository [LZY21]. For information on data augmentation, see Section A.3.1

E-ResNet-50-R The E-ResNet-50-R was trained for 100 epochs with randomly initialized weights using SGD on 224×224 crops with a batch size of 512. After one-third, and then again after two-thirds of the training time, the initial learning rate of 0.1 was decreased by a factor of ten. The weight decay was 0.0001, and the momentum 0.9.

For data augmentation, we used the code from the 'sequential-imagenet-dataloader' repository [Gra17]: During training, crops of various random sizes were passed to the network ranging from 8% to 100% of the original image size. The aspect ratio was chosen randomly between 3/4 and 4/3. Furthermore, different photometric distortions, e.g., random contrast changes, were applied as described in Szegedy et al. [Sze+15] and the Tensorpack repository [Wu+16].

When computing the test scores, each input image was first resized such that the shortest edge's length was 256. Next, the image was cropped in the center to size 224×224, divided by 255, subtracted with 0.5, and again divided by 0.5.

E-CustomNet-L We trained the network for 100 epochs using the Adam optimizer and a batch size of 256. For every 33 steps, we divided the initial learning rate of 0.001 by 10. Furthermore, we used a weight decay of 0.00001. We used the data augmentation as described in Section A.3.1.

A.4 MNIST

After an exhaustive hyperparameter search, each E-PyrNet-20-M was trained on MNIST with the same parameters: we trained two runs on different seeds for each of the 512 possible E-PyrNets-20-M (including the baseline). Each model was trained for 30 epochs, starting with a learning rate of 0.01, which was decreased to 0.001 after 15 epochs. Additionally, we used a weight decay of 0.01 and a momentum of 0.9 with an SGD optimizer and a batch size of 256. Each E-block-M had an expansion factor of two. Except for dividing the images by 255, subsequently subtracting the mean value 0.1307, and dividing by the standard deviation 0.3081, no additional data augmentation was used.

A.5 E-nets for IQA

A.5.1 Legacy training details

For training, we randomly cropped patches of size 32×32 from the images that were furthermore flipped along the vertical axis with a probability of 50%. Each patch was divided by 255. Subsequently, each color channel was subtracted with μ_{imNet} and divided by σ_{imNet} for RGB (see Section A.2.1). The presented networks were trained for 100 epochs with a learning rate of 0.001, a batch-size of 128, and a weight decay of 0.001, using the Adam optimizer. The training loss function was the absolute error between the sigmoid output of the network and the target score divided by 100.

A.5.2 LITW and Kon-IQ training details

We trained the ResNet-50 and E-ResNet-50-I for 100 epochs with the Adam optimizer with a learning rate of 0.0001 and weight decay of 0.001. After the 40th and 80th epochs, the learning rate was divided by 10. For the 2048-dimensional feature vector right before the linear mapping to the quality score, we used a dropout layer with p=0.5. Each training mini-batch contained 32 patches of size 224×224, roughly a quarter of the on average 500×500 dimensional input images. As before, we used the L1-norm as loss function, horizontal flips for data augmentation, and the same normalization as for the Legacy dataset.

A.6 Adversarial attacks and JPEG compression

To compute the fast gradient sign method (FGSM) attacks for each test image, we used the code provided by Torchattacks [Kim20]. The RGB test images were first divided by 255, then the FGSM algorithm was applied, and finally, the image was normalized as described in Section A.2.1. We show a selection of adversarial examples in Figure A.1.

Examples for JPEG-compressed images depending on the quality level are given in Figure A.2. To compute the compression, we used the software provided by the OpenCV library [Bra00].

A.7 Entropy

We analyzed the entropy of all E-neuron-R outputs ($\mathbf{T}_2$) for the E-ResNet-50-R (ImageNet) and the E-PyrNet-56-R (Cifar-10). One hundred randomly sampled images from the respective test set (in the case of ImageNet, the validation set) were passed to each network. For each input, we computed the corresponding feature maps for every E-block-R, one tensor $\mathbf{T}_2$ for every block. We normalized each feature map value $\mathbf{T}_2^m$ from $\mathbb{R}^+$ to $\{0, 1, ..., 255\}$ and computed the entropy of the pixel distribution over the

256 integer values. For the 100 input images, we obtained 100 entropy values for each feature map. We averaged these 100 values resulting in the mean entropy for each feature map, i.e., each E-neuron-R.

We observed that some of the feature maps $\mathbf{T}_1^m$ had all pixel values equal to zero (so-called 'dying ReLUs' [Lu20]). The corresponding E-neurons-R were removed from the analysis. For the E-ResNet-50-R, the percentage of dying ReLUs was 23%, 0.1%, 7%, and 18% for the first, second, third, and fourth E-block-R, respectively. For the E-PyrNet-56-R, only the third E-block-R had 5% dying ReLUs. We tested different weight initializations or alternative non-linearities, such as the leaky ReLU. However, although, using leaky ReLUs stopped the emergence of dying ReLUs, we only noticed a small gain in performance.

A.8 Degree of end-stopping

To measure the degree of end-stopping we used two input images I_0, I_1 , one with a bright and one with a dark square: pixels belonging to the square had a value of +1 or -1, respectively – all other pixels were zero. Each image was normalized to have zero mean and a standard deviation of 1. We computed the feature maps $\mathbf{T}_1(I_0)$, $\mathbf{T}_2(I_0)$, and $\mathbf{T}_3(I_0)$ and squared them to obtain the activation energy. We then normalized each tensor $\mathbf{T}_n$ value from $\mathbb{R}^+$ to $[0, 1]$ by dividing $\mathbf{T}_n$ with the mean plus three times the standard deviation and clipped any values greater than one to make the normalization less susceptible to possible outliers. The percentage of outliers never exceeded 10%. For each feature map, we determined the values $0D$, $1D$, and $2D$ by summing the feature map pixel values (i.e., the activations) over specific regions of interest that were either homogeneous areas, straight edges, or corners in the input image:

$$\psi D(\mathbf{T}_n^m) = \sum_{i,j} \mathbf{T}_n[i, j, m]\mathbf{W}_\psi[i, j]. \tag{A.1}$$

$\mathbf{W}_\psi$ is a binary matrix used to compute the ψD value. All pixels within the region of interest are one, the others are zero. The weighted areas are shown in Figure A.3 in the right panel: the square in the middle is the region of interest for $0D$. The four small squares along the straight edges of the input square measure $1D$, and the four small squares at the corners measure $2D$. Note that the three different regions of interest have the same total area. The left panel shows the input image I_0. The $\psi D(\mathbf{T}_n^m)$ for both input images is the sum $\psi D(\mathbf{T}_n^m) = \psi D(\mathbf{T}_n^m(I_0)) + \psi D(\mathbf{T}_n^m(I_1))$. The degree of end-stopping of a feature map is then defined as:

$$\phi(\mathbf{T}_n^m) = 1 - \frac{1D(\mathbf{T}_n^m)}{2D(\mathbf{T}_n^m) + \epsilon}; \tag{A.2}$$

with $\epsilon = 0.1$. Note that the degree of end-stopping is high (close to one) if the $2D$ activation is high and the $1D$ activation is low. However, two special cases were

considered: (i) a feature map is 'silent', if all values are very small, i.e., $0D+1D+2D < 0.1$. (ii) the feature map is '$0D$' if the $0D$ and $1D$ activations are similar:

$$\mathbf{T}_n^m \; is \; '0D' \Leftrightarrow 1 - \frac{0D(\mathbf{T}_n^m)}{1D(\mathbf{T}_n^m) + \epsilon} < 0.1. \tag{A.3}$$

For these two special cases, Equation A.2 would no longer quantify the degree of end-stopping. Therefore the degree of end-stopping values were not evaluated for silent and $0D$ feature maps. The plots in Figure 4.14 on page 71 show the normalized histograms for the degree of end-stopping. All bars have a bin width of 0.1, and their heights sum up to one.

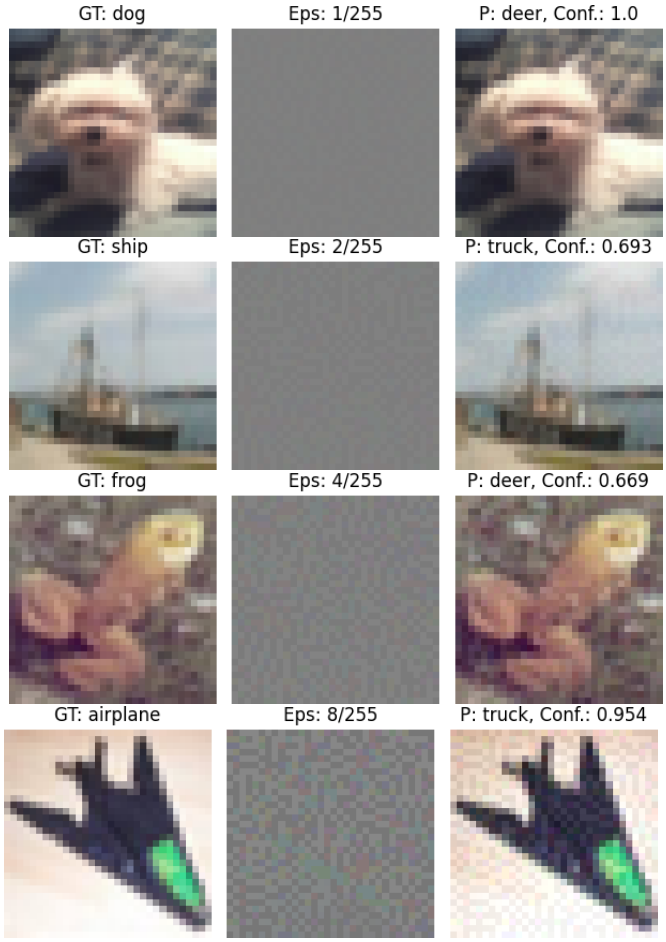


Figure A.1: Examples of Cifar-10 adversarial attacks on the E-ResNet-56-R model. From left to right: original image, perturbation, and resulting adversarial example. Image titles denote the ground truth class (GT), the ϵ value (Eps), the predicted class (P), and the softmax output (Conf.), i.e., the confidence of the network, ranging from 'low'= 0 to 'high'= 1.

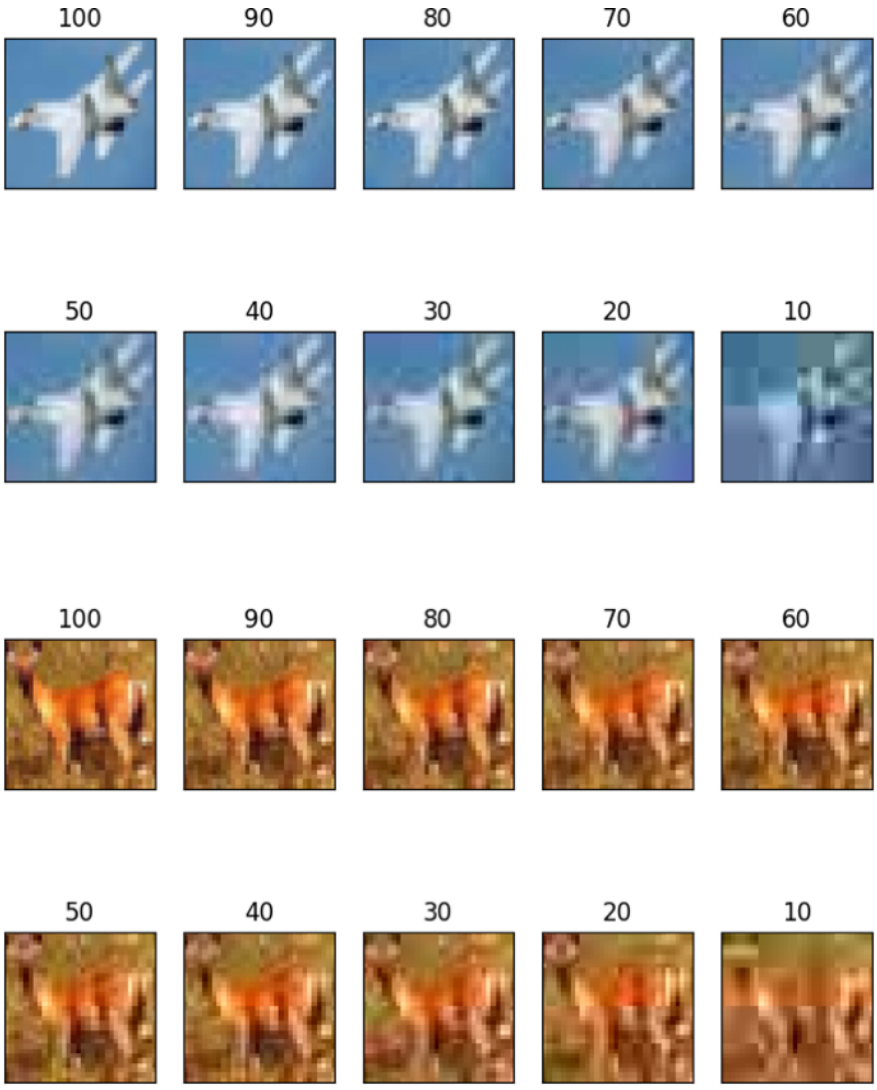
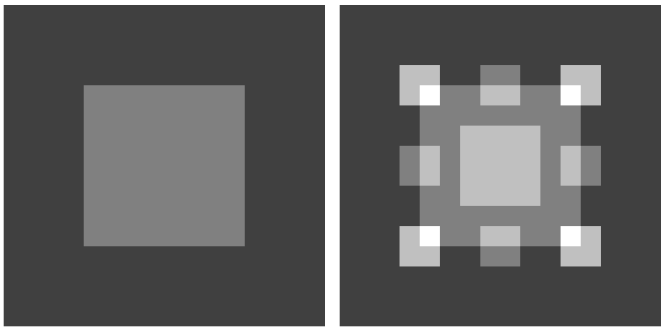


Figure A.2: Cifar-10 JPEG compression examples: the number above each image denotes the image quality $Q \in \{1, 2, ..., 100\}$. $Q = 100$ refers to the original image.



(a) Input **(b)** Input and weights

Figure A.3: Measuring the degree of end-stopping: panel (a) shows the input image presented to the model. Subsequently, we measure the normalized activations $\mathbf{T}_n^m$ of the feature map m of a tensor n. We determine the $0D$-, $1D$-, and $2D$-amount via a weighted sum of the feature map and a binary region of interest image (see Equation A.1). The regions are depicted in panel (b): For $0D$, we measure the activation of the smaller square in the middle. For $1D$, we measure the four small squares located between the corners of the input square, corresponding to the straight lines of the input signal. For $2D$, we measure the four small squares located on the corners of the input square.

Appendix B

Additional results

B.1 E-nets-R

B.1.1 Adversarial attacks

Here, we present additional results for Section 4.4.3.1. The mean robust error values averaged over five runs for different architectures and ϵ-values are given in Table B.1, and the averaged percentage of changed predictions (POCP) values (see Equation 4.5) are given in Table B.2. We observed that, for the basic block networks, the E-ResNet-R consistently outperformed the ResNet for all N's except $N = 3$. For the PyrNets, the larger E-PyrNets-R ($N \in \{7, 9\}$) consistently outperformed the baseline models. Accordingly, especially for larger convolutional neural networks (CNNs), we increased the robustness against adversarial attacks by using E-blocks-R.

B.1.2 JPEG compression

Here, we present additional results for Section 4.4.3.2. The mean robust error values averaged over 5 runs are given in Table B.3, and the averaged POCPs are given in Table B.4. We made the following observations: a decrease in quality by ten was followed by an error increase of roughly $3 - 4$. Analogously, each quality decrease increased the number of changed predictions by $3 - 4$. Deeper networks performed better. Networks using the basic block, the ResNet and the E-ResNet-R, performed better than networks based on the pyramid block. Except for a quality of ten, the E-ResNet-R outperformed the ResNet, and similarly, the E-PyrNet-R outperformed the PyrNet. From this, we concluded that using E-blocks-R in a CNN increased the robustness against noise coming from JPEG compression.

B.2 E-nets-M

B.2.1 Time comparison

To compare the processing time of an E-block-M to an E-block-R, we measured the time a block needed to process a randomly drawn (standard normal distribution) tensor with

Table B.1: Robust error values when using FGSM perturbations for all Cifar-10 models and different ϵ values: we report the mean values averaged over five different training runs.

Model; $\epsilon =$	1/255	2/255	4/255	8/255	16/255
E-PyrNet-20-R	58.460	72.766	82.036	86.180	86.444
E-ResNet-20-R	56.662	73.634	82.296	86.406	89.108
PyrNet-20	60.458	75.566	83.992	88.150	89.762
ResNet-20	57.284	71.912	80.378	84.338	88.016
E-PyrNet-32-R	53.634	70.694	81.088	85.020	88.152
E-ResNet-32-R	51.112	68.510	79.036	84.034	86.830
PyrNet-32	54.620	70.018	79.748	84.392	87.666
ResNet-32	52.434	69.652	79.678	84.166	86.956
E-PyrNet-44-R	51.082	67.468	78.728	83.510	86.814
E-ResNet-44-R	48.360	66.104	77.236	82.486	86.256
PyrNet-44	53.244	69.966	80.764	85.880	88.662
ResNet-44	49.788	67.852	78.590	83.290	86.544
E-PyrNet-56-R	49.226	66.902	78.634	83.524	87.216
E-ResNet-56-R	45.896	64.284	76.076	81.652	85.484
PyrNet-56	52.048	68.442	78.932	84.590	87.620
ResNet-56	47.888	66.056	77.066	81.992	86.410

Table B.2: POCP when using the FGSM perturbations for all Cifar-10 models and different ϵ values.

Model; $\epsilon =$	1/255	2/255	4/255	8/255	16/255
E-PyrNet-20-R	50.766	65.140	74.596	79.320	82.030
E-ResNet-20-R	48.858	65.928	74.906	80.128	85.654
PyrNet-20	52.560	67.818	76.752	82.592	87.562
ResNet-20	49.526	64.228	73.038	78.480	85.148
E-PyrNet-32-R	47.030	64.132	74.858	80.258	86.418
E-ResNet-32-R	44.482	61.960	72.662	78.526	83.700
PyrNet-32	47.808	63.416	73.736	80.064	85.782
ResNet-32	44.996	62.298	72.546	78.148	83.916
E-PyrNet-44-R	44.968	61.452	73.000	78.932	84.772
E-ResNet-44-R	42.102	59.922	71.294	77.402	83.376
PyrNet-44	47.054	63.978	75.342	81.884	86.844
ResNet-44	42.820	60.948	71.898	77.538	83.390
E-PyrNet-56-R	43.536	61.314	73.444	79.506	85.424
E-ResNet-56-R	39.804	58.240	70.196	76.448	82.376
PyrNet-56	45.988	62.592	73.590	80.620	85.988
ResNet-56	41.206	59.426	70.662	76.534	83.808

Table B.3: Error values when using JPEG compression for all Cifar-10 models and different quality (Q) values.

Model; $Q =$	90	80	70	60	50	40	30	20	10
E-PyrNet-20-R	12.412	16.468	19.686	22.624	25.230	28.116	32.920	41.044	57.940
E-ResNet-20-R	12.288	16.338	19.636	22.734	25.306	27.890	32.744	40.138	56.260
PyrNet-20	13.080	17.356	20.676	23.790	27.108	30.324	35.326	42.798	57.336
ResNet-20	12.910	17.322	20.830	24.416	27.308	30.404	35.252	43.356	58.722
E-PyrNet-32-R	11.040	15.154	18.742	22.076	24.676	27.864	32.772	41.240	58.466
E-ResNet-32-R	10.908	14.866	18.142	21.282	23.792	26.760	31.736	40.024	58.488
PyrNet-32	11.918	16.412	20.342	23.898	27.078	30.694	35.888	44.424	60.176
ResNet-32	11.592	15.654	19.114	22.330	24.590	27.462	32.100	40.726	56.878
E-PyrNet-44-R	10.456	14.596	18.214	21.292	24.112	26.720	31.768	40.582	58.022
E-ResNet-44-R	10.624	14.274	17.590	20.384	23.020	26.014	30.646	39.382	56.966
PyrNet-44	11.396	15.928	19.344	23.356	26.518	30.456	35.908	44.706	59.660
ResNet-44	11.338	15.294	18.288	21.254	23.968	27.044	31.534	40.168	56.540
E-PyrNet-56-R	10.118	14.228	18.020	21.116	23.820	26.894	32.382	40.830	57.930
E-ResNet-56-R	10.128	13.984	16.996	19.912	22.116	24.956	29.710	38.096	56.040
PyrNet-56	10.974	15.290	19.088	22.640	25.504	29.244	34.520	43.218	59.616
ResNet-56	10.788	14.830	18.162	21.558	24.200	27.728	32.272	40.942	57.842

Table B.4: POCP when using JPEG compression for all Cifar-10 models and different quality (Q) values.

Model; $Q =$	90	80	70	60	50	40	30	20	10
E-PyrNet-20-R	9.024	14.124	17.676	21.014	23.714	26.688	31.580	40.116	57.502
E-ResNet-20-R	8.868	13.914	17.602	21.064	23.636	26.590	31.540	39.208	55.918
PyrNet-20	10.112	15.086	18.864	22.176	25.670	29.102	34.168	41.842	56.930
ResNet-20	9.996	15.192	19.108	22.826	25.972	29.366	34.340	42.522	58.384
E-PyrNet-32-R	8.374	13.158	17.104	20.644	23.232	26.690	31.766	40.412	58.090
E-ResNet-32-R	8.146	12.820	16.488	19.986	22.540	25.586	30.744	39.296	58.002
PyrNet-32	9.322	14.590	18.746	22.488	25.770	29.656	35.006	43.668	59.840
ResNet-32	8.650	13.584	17.396	20.806	23.168	26.280	31.188	39.802	56.370
E-PyrNet-44-R	8.058	12.868	16.780	19.898	22.830	25.714	30.818	39.848	57.782
E-ResNet-44-R	7.832	12.222	15.932	18.960	21.662	24.748	29.682	38.688	56.742
PyrNet-44	8.804	13.980	17.920	22.114	25.410	29.432	34.964	44.002	59.232
ResNet-44	8.478	13.066	16.464	19.640	22.540	25.884	30.492	39.142	55.986
E-PyrNet-56-R	7.694	12.482	16.704	19.866	22.700	25.886	31.402	40.064	57.774
E-ResNet-56-R	7.632	12.124	15.522	18.564	21.014	23.992	28.952	37.426	55.786
PyrNet-56	8.740	13.498	17.594	21.258	24.294	28.190	33.596	42.560	59.332
ResNet-56	8.180	12.760	16.728	20.056	22.892	26.606	31.298	40.196	57.468

Table B.5: Comparison of the time needed (in milliseconds) for a forward pass: E-block-R versus E-block-M.

	E-block-R	E-block-M
mean	0.743340	0.696191
std	0.044753	0.036152
min	0.662804	0.655890
max	0.940800	0.936270

a batch size of 128, 64 feature maps, and an image height and width of 32. For each block, we repeated this process 1000 times and report the statistics over the acquired values in Table B.5.

B.2.2 DenseNet

Table B.6 shows additional results for the DenseNet models on the Cifar-10 classification task. When regarding minimal errors over all epochs, i.e., the best result with early stopping, we obtained similar results for the DenseNet ($L = 100, k = 12$) as reported in the original paper (4.51) [Hua+17]: the minimum error reached by the baseline DenseNet was 4.56 and 4.36 for the E-DenseNet.

B.2.3 JPEG compression

Figure B.1 shows the Cifar-10 test errors depending on the Q value. Note that these plots only provide the results for $Q = 100, 90, ..., 60$, and only for the largest models of each experiment (ResNet: $N = 9$, DenseNet: $k = 12, L = 100$). The results for $Q = 100, 90, ..., 10$ and for all models are given in the Tables B.7, B.8, B.9, and B.10. Table B.7 shows the test error values for the DenseNet experiment and Table B.8 shows the POCP. The test error and POCP for the ResNet experiments are given in Table B.9 and Table B.10, respectively. Note that the POCP does not only focus on inputs that were classified correctly before compression and classified wrongly after compression. Additionally, inputs that change from one incorrect class to another are counted. Some example images compressed with different Q-values are shown in Figure A.2 on page 148.

B.3 Finding the optimal E-net design rule

We report the genetic algorithm results for different numbers of blocks per stack: see Table B.11 for five blocks ($N = 5$), Table B.12 for seven blocks, and Table B.13 for nine blocks.

Table B.6: Classification results for the DenseNet experiment.

Model	k	Num. parameters	L	After 300 epochs				Min. over all epochs			
				mean	std	min	max	mean	std	min	max
E-DenseNet	12	57040	22	9.22	0.17	9.03	9.33	8.78	0.10	8.72	8.89
DenseNet	12	71686	22	9.76	0.19	9.57	9.94	9.39	0.13	9.27	9.52
E-DenseNet	12	298564	58	5.67	0.07	5.62	5.75	5.51	0.07	5.46	5.59
DenseNet	12	314470	58	5.90	0.30	5.72	6.25	5.65	0.14	5.51	5.78
E-DenseNet	12	751786	100	4.55	0.14	4.39	4.64	4.51	0.13	4.36	4.59
DenseNet	12	769162	100	4.79	0.06	4.74	4.86	4.62	0.06	4.56	4.66

Table B.7: DenseNet: mean test error on Cifar-10 for JPEG-compressed test sets $\mathcal{S}_Q$.

Model	L	k	Q=100	90	80	70	60	50	40	30	20	10
E-DenseNet	22	12	9.2	14.9	19.5	23.2	26.4	29.5	32.5	37.5	44.9	60.3
DenseNet	22	12	9.8	15.4	20.0	24.0	27.6	30.8	34.6	39.5	47.8	64.6
E-DenseNet	58	12	5.7	10.4	15.0	19.0	22.6	25.4	28.6	34.0	43.1	62.5
DenseNet	58	12	5.9	10.9	15.7	20.0	24.3	27.3	31.3	36.4	45.9	63.0
E-DenseNet	100	12	4.6	8.8	13.6	17.2	20.9	23.8	27.6	33.1	42.5	61.4
DenseNet	100	12	4.8	9.9	14.6	19.1	22.8	25.9	30.1	35.2	44.5	62.6

Table B.8: DenseNet: mean POCP on Cifar-10 for JPEG-compressed test sets $\mathcal{S}_Q$.

Model	L	k	Q=90	80	70	60	50	40	30	20	10
E-DenseNet	22	12	11.7	16.9	21.4	24.8	28.0	31.2	36.3	43.9	60.0
DenseNet	22	12	12.1	17.6	21.9	25.8	29.3	33.4	38.3	46.8	64.2
E-DenseNet	58	12	8.3	13.7	17.8	21.6	24.6	27.9	33.5	42.5	62.4
DenseNet	58	12	9.2	14.7	19.3	23.5	26.4	30.5	35.7	45.3	62.7
E-DenseNet	100	12	7.3	12.4	16.3	20.1	23.2	27.1	32.6	42.1	61.4
DenseNet	100	12	8.1	13.5	18.2	21.9	25.1	29.3	34.8	44.1	62.6

Table B.9: ResNet: mean test error on Cifar-10 for JPEG-compressed test sets $\mathcal{S}_Q$.

Model	N	Q=100	90	80	70	60	50	40	30	20	10
E-ResNet	3	8.0	12.5	16.7	20.2	23.5	26.2	29.3	34.1	42.5	59.1
ResNet	3	8.3	13.2	17.4	21.0	24.5	27.6	31.1	36.0	44.1	59.8
E-ResNet	5	6.9	11.3	15.1	18.7	22.0	24.9	27.9	33.4	42.3	59.2
ResNet	5	7.2	11.8	16.3	20.0	23.6	26.6	30.0	35.4	44.0	59.6
E-ResNet	7	6.2	10.6	14.6	17.9	21.5	24.5	27.7	32.8	42.2	59.9
ResNet	7	6.8	11.4	15.9	19.2	22.9	26.3	29.7	35.0	43.7	60.1
E-ResNet	9	6.0	10.1	14.4	18.0	21.4	24.2	27.4	32.7	41.7	58.8
ResNet	9	6.3	10.9	15.3	19.0	22.4	25.6	29.2	34.5	43.4	59.1

Table B.10: ResNet: mean POCP on Cifar-10 for JPEG-compressed test sets $\mathcal{S}_Q$.

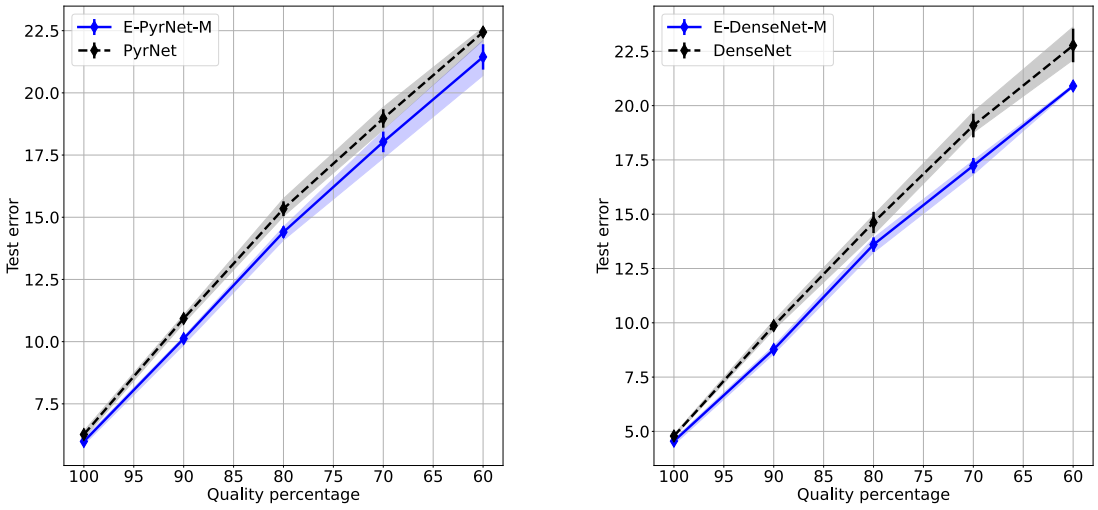
Model	N	Q=90	80	70	60	50	40	30	20	10
E-ResNet	3	9.5	14.5	18.4	21.8	24.7	28.0	33.1	41.8	58.8
ResNet	3	10.1	15.1	19.3	22.9	26.2	29.9	34.8	43.3	59.3
E-ResNet	5	8.5	13.2	17.1	20.6	23.7	26.8	32.5	41.6	58.8
ResNet	5	9.4	14.4	18.6	22.3	25.5	29.1	34.5	43.2	59.3
E-ResNet	7	8.1	12.8	16.7	20.3	23.3	26.6	32.0	41.4	59.7
ResNet	7	8.9	14.0	17.9	21.8	25.1	28.8	34.2	43.1	59.8
E-ResNet	9	7.8	12.6	16.6	20.2	23.0	26.3	31.7	40.9	58.5
ResNet	9	8.5	13.6	17.6	21.1	24.5	28.2	33.6	42.7	58.7

Table B.11: Ten best E-PyrNets-32-M ($N = 5$) found by the genetic algorithm: we do not provide results for the baseline and the default-model for this particular experiment (i.e., the particular seeds). However, based on previous results (see e.g., Figure 4.20), the baseline has approximately a mean test error of 7.2 and the default-model a mean test error of 6.9

Rank	Bitstring	Cifar-10 test error		
		mean	min	max
1	00011 01000 00010	6.433	6.352	6.514
2	00010 00010 00010	6.456	6.368	6.544
3	00011 00000 00010	6.467	6.398	6.536
4	10100 01000 01000	6.548	6.538	6.558
5	00111 01000 00011	6.554	6.288	6.820
6	00101 00010 00000	6.572	6.456	6.688
7	01011 01000 00010	6.587	6.582	6.592
8	00010 00001 00000	6.602	6.526	6.678
9	00011 00000 01010	6.603	6.444	6.762
10	10011 01100 00110	6.607	6.542	6.672

Table B.12: Ten best E-PyrNets-44-M ($N = 7$) found by the genetic algorithm. In addition, we report the default-model results and baseline results.

Rank	Bitstring	Cifar-10 test error		
		mean	min	max
1	0000011 0100000 0000001	5.932	5.864	6.000
2	0000011 0100000 0000000	5.973	5.804	6.142
3	0000011 0101000 0000000	6.005	5.842	6.168
4	0000011 1100000 0001000	6.027	5.752	6.302
5	0001101 0001000 0000000	6.037	5.890	6.184
6	0000011 0101000 0100000	6.037	5.814	6.260
7	0101111 0001000 0010000	6.037	6.026	6.048
8	0000101 0100000 0000010	6.059	5.976	6.142
9	0000011 0100000 0100001	6.077	5.970	6.184
10	0001011 1001000 0000000	6.096	5.982	6.210
48	1000000 1000000 1000000	6.304	6.270	6.338
67	0000000 0000000 0000000	6.970	6.938	7.002



(a) PyrNet (b) DenseNet

Figure B.1: Cifar-10 test error depending on the compression quality.

Table B.13: Ten best E-PyrNets-56-M ($N = 9$) found by the genetic algorithm. In addition, we report the baseline results. Considering the results from Figure 4.20, the default-model has an test error of 6.

Rank	Bitstring	Cifar-10 test error		
		mean	min	max
1	000110010 111000000 000000000	5.548	5.430	5.666
2	001110011 001000000 000000000	5.600	5.564	5.636
3	001110010 001010000 001000000	5.627	5.436	5.818
4	001110010 011010000 001000000	5.655	5.448	5.862
5	001110010 010001000 001000000	5.657	5.600	5.714
6	001110010 011001000 001000000	5.668	5.602	5.734
7	000110010 001000000 000000000	5.676	5.514	5.838
8	001100000 010001010 001000000	5.684	5.608	5.760
9	001110010 011000000 000000000	5.723	5.618	5.828
10	001110000 011001000 001000000	5.728	5.696	5.760
159	000000000 000000000 000000000	6.549	6.450	6.648

Table B.14: Cifar-10 test results averaged over five runs: for $N \in \{5, 7, 9\}$, we used the three best models (E-PyrNets-M with a specific design rule) found with the genetic algorithm (Table B.11, Table B.12, and Table B.13). For $N = 3$, we used the two models with the lowest test error on Cifar-10 (see Table 4.15 on page 98) and the model with the lowest max-rank on Cifar-10 and MNIST (see Table 4.16 on page 99). Furthermore, for each number of blocks, we evaluated the baseline and the default model.

Bitstring	N	num. param. (k)	Cifar-10 test error			
			mean	std	min	max
001110010 001010000 001000000	9	761	5.75	0.18	5.51	5.92
000110010 111000000 000000000	9	809	5.90	0.16	5.65	6.10
001110011 001000000 000000000	9	826	6.01	0.17	5.78	6.22
100000000 100000000 100000000	9	801	6.08	0.27	5.73	6.38
000000000 000000000 000000000	9	854	6.29	0.08	6.16	6.35
0000011 0100000 0000000	7	640	6.08	0.12	5.87	6.17
0000011 0101000 0000000	7	627	6.13	0.18	5.82	6.29
0000011 0100000 0000001	7	585	6.23	0.25	5.92	6.51
1000000 1000000 1000000	7	606	6.47	0.18	6.22	6.69
0000000 0000000 0000000	7	660	6.73	0.18	6.53	6.96
00011 01000 00010	5	390	6.62	0.27	6.41	7.07
00011 00000 00010	5	404	6.74	0.25	6.46	7.10
00010 00010 00010	5	393	6.88	0.40	6.43	7.45
10000 10000 10000	5	411	6.97	0.15	6.81	7.13
00000 00000 00000	5	465	7.25	0.26	6.92	7.54
100 100 000	3	257	7.93	0.26	7.62	8.25
011 010 100	3	210	7.96	0.26	7.67	8.26
100 100 100	3	217	8.14	0.16	7.95	8.33
000 100 000	3	260	8.19	0.19	7.93	8.39
000 000 000	3	270	8.28	0.25	7.83	8.45

Appendix C

E-nets for medical image segmentation

This section gives additional information about the evaluation metrics, datasets, and hyperparameters of Section 4.8. Here, we present excerpts from Tilman Wegener's Master's thesis [Weg21], which we supervised.

C.1 Evaluation metrics

In binary segmentation, there are two classes, the object of interest and the background. By applying the sigmoid function to a CNN's output, a probability (between zero and one) for each pixel is returned. The predicted segmentation can be obtained by thresholding the probabilities with a selected threshold value. If not stated otherwise, a value of 0.5 is used in this work.

Using the ground truth label and the prediction, the number of true positives (TPs), true negatives (TNs), false positives (FPs), and false negatives (FNs) can be calculated. In image segmentation, the FPs, for example, correspond to the number of pixels that were classified as foreground but actually belonged to the background.

The Matthews correlation coefficient (MCC) [CJ20] is a robust measure of the performance of binary classifiers and is defined as

$$\mathrm{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}}. \tag{C.1}$$

An MCC of plus one indicates perfect performance, a value of zero random prediction, and a value of minus one corresponds to exactly predicting the opposite of the true label.

The MCC is insensitive to imbalanced class distributions (e.g., when the background is more dominant than the foreground) and to which class is chosen as positive. It is therefore seen as one of the best measures to describe the performance of a binary classifier in a single number [CJ20].

Two often-used metrics for medical image segmentation algorithms are the Dice coefficient and the Jaccard index (also known as F1-score and intersection over union (IoU), respectively) [Min+22]. They are defined as:

$$\text{Dice} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}, \qquad \text{Jaccard} = \frac{TP}{TP + FP + FN}, \tag{C.2}$$

and can take values between zero and one, where one indicates perfect segmentation. Other metrics we will provide include accuracy, sensitivity, and specificity [Min+22]:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}, \tag{C.3}$$

$$\text{Sensitivity} = \frac{TP}{TP + FN}, \qquad \text{Specificity} = \frac{TN}{TN + FP}. \tag{C.4}$$

Again, a value of one indicates better performance, and a value of zero indicates poor performance.

However, these last five metrics face several problems that make them unsuitable for evaluating binary classifiers [CJ20]. The Dice score, for example, is independent of the number of TNs. The MCC does not face these problems and should therefore be preferred when evaluating binary classifiers [CJ20].

C.2 Digital retinal images for vessel extraction dataset

The digital retinal images for vessel extraction (DRIVE) dataset [Nie+04] is a publicly available dataset for binary segmentation of blood vessels in retinal images. The vascular structure of the eye is of clinical importance for the diagnosis and screening of various cardiovascular and opthalmological diseases [Nie+04]. Changes in the morphological characteristics of retinal blood vessels, such as width, can be indicators of diseases, for example, diabetes, hypertension, or arteriosclerosis [Fra+12]. Retinal blood vessels can be examined easily and non-invasively, which allows the screening of large populations. Segmentation of the blood vessels is a crucial first step in this analysis [Guo+21].

However, manual segmentation requires expert knowledge and is prone to errors. In addition, ophthalmologists often lack the time to segment and analyze the images themselves. Therefore, an automated segmentation method is required as a first step in developing computer-assisted diagnostic systems for ophthalmic disorders [Fra+12].

The DRIVE dataset [Nie+04] consists of 20 training and 20 test images in RGB format with a resolution of 565×584 pixels. The images were acquired from the Dutch Diabetic Retinopathy Screening Program using a Canon CR5 non-mydriatic 3-CCD fundus camera [Fra+12]. Training images have one manually annotated blood vessel

segmentation, while the test set was segmented twice. The annotators were trained by an experienced ophthalmologist. Images of healthy subjects and images with visible pathological abnormalities, such as background diabetic retinopathy, were included in the dataset [Fra+12].

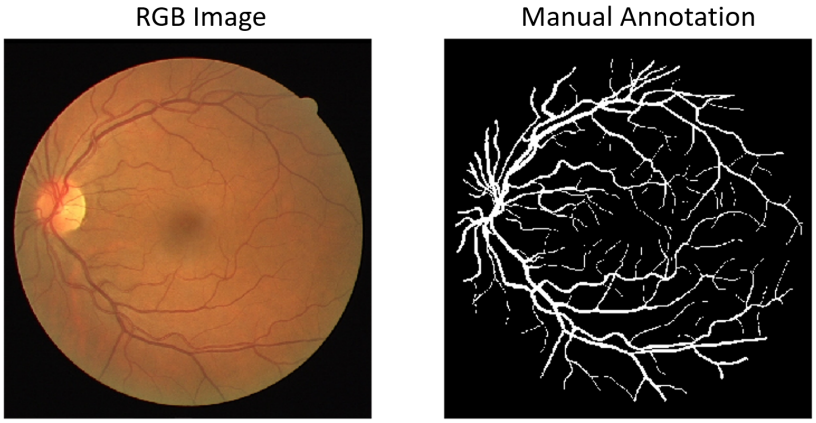


Figure C.1: Example input RGB image (left) and corresponding ground truth blood vessel segmentation (right) from the DRIVE dataset [Nie+04]: retinal vessel segmentation is a challenging task due to the small size of the blood vessels, varying lighting conditions, and presence of pathological abnormalities in the dataset.

C.2.1 Pre-processing and augmentation

To ensure comparability, all models trained on DRIVE used the same pre-processing steps and augmentation. After some initial tests to augment the data randomly during training, the data from the SA-U-net [Guo+21] GitHub repository was used. Guo et al. [Guo+21] augmented the training data offline (before training) from 20 to 234 images using horizontal and vertical flipping, color jittering, and adding Gaussian noise. Images were scaled to the range of [0, 1] and normalized with the ImageNet mean (see μ_{imNet} in Section A.2.1 on page 141). No contrast enhancement was performed, and RGB images were used as by Guo et al. [Guo+21]. Images were padded to a size of 592×592 pixels and cropped back to their original size before calculating the metrics. Performance was evaluated on the 20 test images from the original DRIVE dataset without augmentation, using the segmentation of the first annotator as ground truth.

C.2.2 Hyperparameters

The training procedure and hyperparameters used by Guo et al. [Guo+21] were employed for training the SA-U-net and the corresponding E-nets (see Section 4.8.1.1). The models were trained from scratch on the DRIVE dataset for 150 epochs using the Adam optimizer [KB15], a batch size of four, and a learning rate of 0.001 that was

reduced by a factor of ten after 100 epochs. The loss function was the weighted binary cross-entropy (BCE) loss (weights = 0.65/0.35 for blood vessels/background). The DropBlock size was set to 7×7 and the drop probability to 0.18. The weighted BCE loss is often used with imbalanced class distributions [Jad20]. It weighs the positive and negative part of the loss function with a constant $\beta \in (0, 1)$ and is defined as:

$$L_{W-BCE}(y, \hat{y}) = -\beta \cdot y \cdot log(\hat{y}) - (1 - \beta) \cdot (1 - y) \cdot log(1 - \hat{y}). \tag{C.5}$$

C.2.3 Producing the baseline results

We attempted to reproduce the results of Guo et al. [Guo+21]. Using ten different random seeds, we calculated the mean and standard deviation values of the Dice score, accuracy, and MCC. Table C.1 shows the results for the PyTorch re-implementation. In contrast to the SA-U-net paper, where augmentation was calculated once prior to training, here, the training data was augmented on-the-fly. However, the same random augmentations were used as described by Guo et al. [Guo+21] (flipping, rotation, color jittering, and adding Gaussian noise).

The reproduced metrics were not able to match the values reported in the SA-U-net paper. Extensive tests were performed to match the performance (e.g., changing the augmentation strength and hyperparameter tuning), but no improvement was observed.

Table C.1: Reported and reproduced metrics for the SA-U-net when using augmentation on-the-fly. The same augmentation techniques were used as described by Guo et al. [Guo+21]. For the reproduced metrics, the mean, standard deviation, and maximum metrics are given. A higher metric corresponds to better performance.

SA-U-net implementation	Dice	Accuracy	MCC
Reported results [Guo+21]	0.8263	0.9698	0.8097
Reproduced with PyTorch re-implementation	0.8170 ± 0.0046 Max: 0.8220	0.9681 ± 0.0039 Max: 0.9691	0.8008 ± 0.0014 Max: 0.8051

Therefore, the offline augmented training data that was made publicly available in the SA-U-net GitHub repository [Guo+21] was used for training. Using these data, both the PyTorch re-implementation and the published Keras implementation were evaluated. The mean, standard deviation, and maximum metrics are shown in Table C.2. The performance of the PyTorch re-implementation increased significantly on the original DRIVE test data. The reason for this is unclear; however, it was investigated whether there was any leaking of test data into the training data, which was not the case. The augmented images also looked visually very similar to the images in the official DRIVE

training dataset. Due to better performance, the offline augmented training data was used for training all models on DRIVE, while the official DRIVE test data was used for testing.

Despite the increase in performance, the mean reproduced metrics in Table C.2 were lower than the reported metrics [Guo+21]. This is surprising since the same hyperparameters, training data, and pre-processing steps were used. One explanation would be that Guo et al. reported the maximum metrics over several runs (they did not comment on this). When using maximum metrics for comparison, the reproduced PyTorch model performed best in both accuracy and MCC, and even better than the published Keras implementation. Therefore, the PyTorch re-implementation of the SA-U-net was used as the baseline for all corresponding E-nets.

Table C.2: Reported and reproduced metrics for the SA-U-net when using the offline augmented training data that was made publicly available in the SA-U-net GitHub repository. For the reproduced metrics, the mean, standard deviation, and maximum metrics are given.

SA-U-net implementation	Dice	Accuracy	MCC
Reported results [Guo+21]	0.8263	0.9698	0.8097
Reproduced with PyTorch	0.8230 ± 0.0016 Max: 0.8248	0.9694 ± 0.0005 Max: 0.9700	0.8079 ± 0.0014 Max: 0.8100
Reproduced with published Keras code	0.8245 ± 0.0008 Max: 0.8257	0.9692 ± 0.0003 Max: 0.9698	0.8077 ± 0.0007 Max: 0.809

C.3 Skin lesion segmentation

Skin cancer is the most common type of cancer globally, accounting for more than 40 % of all cases [CAC12]. It is usually caused by over-exposure to ultraviolet light from the sun or tanning beds and can be divided into non-melanoma, and melanoma skin cancer [CAC12]. Of these types, melanoma skin cancer is the most aggressive and deadly form, but it is usually curable if detected at an early stage [CAC12].

Dermoscopy was developed as a non-invasive imaging technique for screening conspicuous skin lesions. It uses magnification lenses and a powerful illumination system to examine the exact morphology and structure of the lesion [Men+13]. However, visual examination of dermoscopic images can be time-consuming, and accuracy in detecting malignant melanoma depends mainly on the dermatologist's experience [Men+13]. Therefore, computer-aided detection systems have been designed that often consist of three stages: image segmentation, feature extraction, and lesion classification [ÖÖ20].

Skin lesion segmentation as a first step makes the decision of a classification algorithm more explainable. However, later stages depend heavily on segmentation performance requiring high accuracy at this stage.

C.3.1 ISIC 2017 dataset

The international skin imaging collaboration (ISIC) 2017 dataset [Cod+18] is a dataset for skin lesion analysis that was published as a challenge at the IEEE ISBI 2017. The challenge consisted of three parts: lesion segmentation, feature detection, and disease classification. For this work, only the lesion segmentation task is relevant.

The lesion segmentation dataset consists of 2 000 training, 150 validation, and 600 test images. Only training and test images were used in this work since the validation images were unavailable on the official website. The images are in RGB format and have varying sizes ranging from 540×722 up to $4\,499 \times 6\,748$ pixels.

Each image has a corresponding binary segmentation of the skin lesion that an expert annotated. Figure C.2 shows some example images from the test dataset that illustrate the challenges of lesion segmentation, such as hairs and markers in the image, uneven lighting conditions, and varying lesion sizes [Has+20].

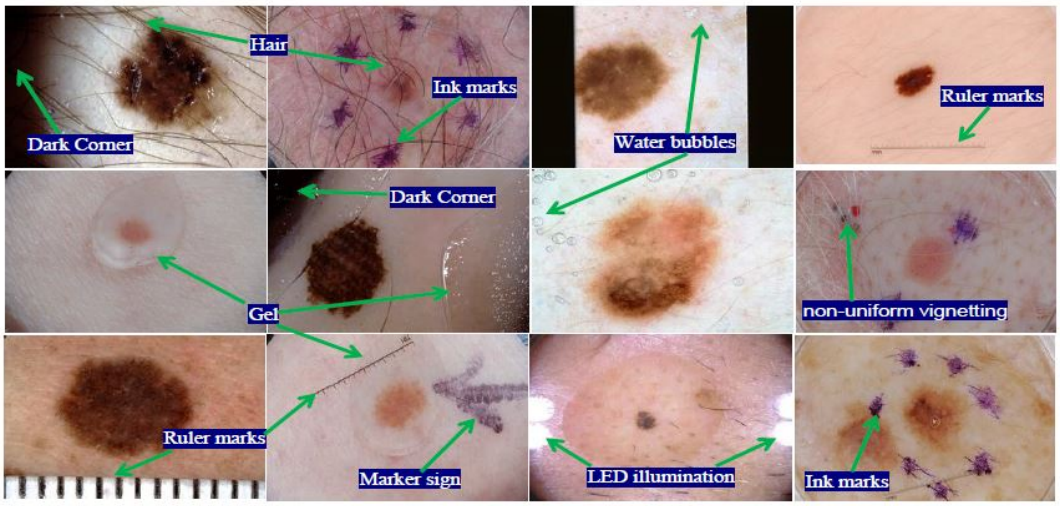


Figure C.2: Example images from the ISIC 2017 test dataset and some image features that make skin lesion segmentation a challenging task. Figure adopted from Hasan et al. [Has+20].

C.3.2 PH2 dataset

The PH2 dataset [Men+13] was introduced by the Dermatology Service of the Hospital Pedro Hispano in Matosinhos, Portugal, for both image segmentation and classification.

For segmentation, the dataset consists of 200 RGB images with a size of 768×560 pixels. A domain expert annotated the corresponding binary segmentation masks.

In this work, the PH2 dataset was only used to test the learned lesion segmentation on the ISIC 2017 dataset, as done by Hasan et al. [Has+20].

C.3.3 Pre-processing and augmentation

Since most images have an aspect ratio of 3:4, the images and masks of the PH2 and ISIC 2017 dataset were resized to 192×288 pixels using interpolation (bilinear for the images, nearest-neighbor for segmentation masks). A similar approach was used by Hasan et al. [Has+20] for the DSNet. Images were scaled to [0, 1] and normalized with the ImageNet mean and standard deviation (see Section A.2.1 on page 141).

The following random augmentation techniques were used during training: adding Gaussian noise, affine transformation (rotation, scaling, translation), and color jittering. For testing, the pixel-wise probability map produced by the network was upsampled to the original image size using bilinear interpolation, and metrics were calculated on this size.

C.3.4 Hyperparameters

All models were trained for 150 epochs using the Adam optimizer, a batch size of 16, and a learning rate of 0.001, which was reduced to 0.0001 after 100 epochs. The training data was augmented as described in Section C.3.1.

C.3.5 Producing the baseline results

We compared the results of a baseline ResNet-50 (pre-trained on ImageNet, dubbed Res-DSNet) to the original DSNet (using a pre-trained DenseNet-121). Hasan et al. [Has+20] proposed a combined loss function consisting of the sum of IoU and BCE loss and reported that this loss outperformed both BCE and IoU loss individually. For the Res-DSNet, experiments with the following loss functions were performed: BCE, weighted BCE (weights 0.65/0.35), IoU + BCE, IoU, and Focal loss (using the Kornia library [Rib+20]). Since the weighted BCE loss for the Res-DSNet performed best in most metrics, this loss function was used to train all other Res-DSNet models and the corresponding E-nets.

Hasan et al. [Has+20] used two post-processing steps for determining the final segmentation mask. First, they used ISODATA [Vel79] to determine an optimal threshold to separate the foreground and background pixels in the output probability map. Second, they chose the largest area of connected components in the binary mask as the final segmentation. For this work, a visual inspection of the output segmentations indicated that only keeping the largest area in the binary mask would not improve performance significantly. Furthermore, the observed performance improvement due to

ISODATA thresholding was insignificant, and the evaluation time increased. Therefore, ISODATA post-processing was not used in further experiments. Instead, the threshold of 0.5 was used.

Glossary

1×1 convolution A 1×1 convolution (also called point-wise convolution) is a convolution layer with a kernel size $k = 1$. Thus, instead of applying image filters ($k > 1$), a 1×1 convolution only results in a linear recombination of the incoming feature maps. 17, 31

2D patches 2D patches (2D regions, 2D signals) are patches in an image with an iD of two. In natural images, these patches contain a lot of information. Furthermore, knowing the 2D patches of a grayscale image suffices for reconstructing the entire image. 2D patches are extracted by end-stopped cells in the visual cortex. 9

adversarial attack Adversarial attacks (sometimes called adversarial examples) are small perturbations – for images, noise patterns that are barely visible to the human eye – that are added to an input aiming to change the decision of a classifier. These attacks are surprisingly effective with neural networks and therefore pose an important challenge in deep learning research. 72

architecture An architecture is a set that contains models which are similar in a specific aspect. E.g., the set of E-nets where each particular model contains at least one E-block. 28

baseline In the context of E-nets, a baseline is a convolutional neural network (CNN) that is used as a starting point to create an E-net. The baseline is transformed by replacing some of its blocks with E-blocks. In the result sections, baseline models are usually established approaches that we aim to outperform with our E-nets. 27

basic block A basic block is a specific sequence of two convolutions, BN layers, and ReLUs first presented in shallower ResNets which are often used in small-image datasets, such as Cifar-10. 22

block In a CNN, a block is a specific sequence of layers, for example, the E-block. 15, 27

batch normalization (BN) BN is a normalization layer usually applied after each convolution layer to re-scale the convolution's output. BN facilitates training deeper networks as it smooths the optimization landscape and reduces the chance of exploding or vanishing (gradient) values. 20

bottleneck block A bottleneck block is an alternative to a standard convolution layer. The block uses a 1×1 convolution to reduce the number of incoming feature maps, applies a 3×3 convolution to this smaller tensor, and increases the number of feature maps again with another 1×1 convolution. Given the right feature dimensions, a bottleneck block needs fewer parameters than a standard convolution layer. 23

capacity A machine learning algorithm has a high capacity if it can formulate a large variety of hypotheses to fit data. A high capacity is necessary to obtain a small training error on complex datasets; however, a highly capable machine learning algorithm may overfit the training data leading to a poor generalization of unseen test data. 3

clipped iterative gradient ascent (CIGA) The CIGA is a white box adversarial attack technique. Similar to the FGSM, it aims to change a network's output for an image by adding a noise pattern that does not change each image's pixel value by more than ϵ. In contrast to the FGSM, CIGA does not use the sign function but rather adds the clipped version of the gradient to an image. 73

dense layer A dense layer denotes a matrix multiplication of an input $\vec{x}$ with a weight matrix $\mathbf{W}$. In contrast to convolution layers, which contain many zero entries and redundancies in their weight matrices, i.e., many of the shared weights are sparse, each entry of a dense layer's weight matrix can hold a unique value. 16

DenseNet DenseNets present an alternative to ResNets. Instead of residual connections, which enable the reuse of shallower features in deeper layers by adding a layer's input to its output, DenseBlocks explicitly concatenate shallower features to pass them to a higher layer. 22

design rule To create an E-net, the convolution blocks of a baseline CNN are substituted with E-blocks. A design rule defines which blocks should be exchanged. 30, 94

depthwise (DW) When using DW convolution layers, each feature map m is filtered with a single kernel $\mathbf{K}_{DWS}^{m} \in \mathbb{R}^{k \times k}$. Hence, the number of parameters is reduced when compared to a standard convolution, where, for an output feature map m, each kernel also sums over all d input feature maps $\mathbf{K}_{conv}^{m} \in \mathbb{R}^{k \times k \times d}$. However, when applying DW convolutions, no information exchange happens between feature maps. Accordingly, 1×1 convolutions usually follow DW convolutions (this sequence is also called a 'depthwise separable convolution'). 17

efficient An efficient CNN can provide performance comparable to state-of-the-art (SOTA) models while needing fewer parameters, fewer operations, or needing less time to compute the results. 47

end-stopping When first discovered, the behavior of end-stopped cells of the visual cortex was described as responding to a line segment with a particular orientation. If that line segment exceeded a certain length, the response was inhibited. However, in this work, we use the definition that an end-stopped neuron reacts only to signals with an iD of two. 2, 9

exo-origin iso-response contour In a two-dimensional projection of a neuron's output, an iso-response contour denotes a path along which the output value does not change. For neurons that are not hyperselective, the iso-response contours are straight lines. For hyperselective neurons, some of the iso-response contours bend away from the origin of the projection (exo-origin), indicating that these neurons are tuned more specifically to their optimal stimulus. 38

fully convolutional network (FCN) An FCN is a CNN created for image segmentation, consisting of an encoder-part and a decoder-part. FCNs do not contain dense layers (fully convolutional). 105

feature map With the m-th feature map, we denote the channel of a tensor $\mathbf{T}[:,:,m]$. After operations such as convolution, $\mathbf{T}[:,:,m]$ is the result of the input processed by a specific neuron m. Thus, $\mathbf{T}[:,:,m]$ shows the features extracted by the neuron. 15

fast gradient sign method (FGSM) The FGSM is a white box (i.e., the method has access to the model parameters) adversarial attack technique. In this work, it is used to change the output of a network for an image by adding a particular noise pattern to the image. The noise pattern is computed by applying the sign function to the gradient of the network output for the original image and multiplying the result with a value ϵ. The resulting noise pattern is added to the input image. If the attack is successful, the network's output is altered, e.g., instead of a dog, the network now 'sees' a car. ϵ is usually chosen to be very small so that for a human, the image and the image with the added noise pattern look the same. 73

hyperselectivity For a hyperselective neuron, deviating from the optimal stimulus (along a certain direction) reduces the output of the neuron. 37

instance normalization (IN) Like BN, IN is applied after convolution layers to facilitate network training. However, each instance is normalized instead of computing normalization values over an entire batch. IN is especially used in E-blocks-R for image classification. 20

intrinsic dimensionality (iD) Based on differential geometry, small patches of images can be separated into classes that contain different amounts of information. The iD counts the degrees of freedom (from 0 to 2) along which the image patch

changes in its neighborhood. Opposed to 0D or 1D patches that are redundant in their close vicinity (i.e., have a high auto-correlation), 2D patches are unique in their neighborhood. 9

layer In a CNN, a layer denotes the smallest building block, such as convolution layers or non-linearity functions. 27

log-space A signal $\mathbf{x}$ is transformed into log-space by taking the logarithm of said signal. Adding entries of the signal in log-space (e.g., by applying convolution) and transforming the signal back into the original space results in a product of the entries. 83

max-rank The max-rank is the maximum rank value of a model over several experiments, e.g., the rank values over several datasets. We use this metric to investigate which design rule performs best on Cifar-10 and MNIST. For each model (defined by a specific design rule), we take the maximum of the Cifar-10 rank and the MNIST rank. The model with the smallest maximum value performs best on both datasets. 96

mobile inverted bottleneck The mobile inverted bottleneck is a special version of the bottleneck block. Instead of using a standard 3×3 convolution on a reduced number of feature maps, a 3×3 DW convolution is applied to a tensor with an increased number of feature maps. Since DW convolutions need fewer parameters than standard convolutions, mobile inverted bottlenecks can be very parameter efficient – despite the increased number of feature maps, which is done to reduce potential information loss due to using ReLUs. 23

network depth The depth of a network denotes the number of layers applied in sequence by a neural network. Using convolution layers with non-linearities can drastically increase the capacity of a network. Interestingly, deep CNNs do not tend to overfit but often generalize better with increased depth. Using better hardware to enable the training of deeper networks with more data created the starting point for today's rediscovery and widespread adoption of neural networks in machine learning – nowadays often called deep learning. 21

neuron In a block or layer, a neuron produces a single feature map. When referring to the visual cortex, a neuron is a cell of the visual cortex. 28

optimal stimulus The optimal stimulus of a neuron is an input signal that produces the neuron's maximal output; i.e., the neuron is maximally excited. 37

orientation-selective A neuron of the visual cortex (e.g., a simple cell) is said to be orientation-selective when it is strongly excited when presented with a particular shape at a specific angle in the neuron's visual field. Similarly, neurons of a CNN

(oriented filters) can produce a large output to stimuli with a specific orientation while not reacting to other orientations. Certain Sobel filters, which CNNs can learn, for example, strongly react to horizontal- but not to vertical edges and lines. 6, 85

oriented filters Oriented filters model neurons that are orientation selective. For example, Sobel filters. 6

pyramid block A pyramid block is an improved version of the basic block with only one ReLU (instead of two) and three BN layers instead of two. 22

PyrNet A PyrNet is an architecture based on the Cifar-10 ResNets, with a pyramid block as the main building element. The pyramid block was proposed to improve the original ResNet's basic block and contains more BN layers and fewer ReLUs. 22

receptive field In the visual cortex, a neuron (e.g., a simple cell) only reacts to a small area of the entire visual field: this area is called the receptive field. For the neuron of a CNN, the receptive field denotes the effective patch size of an input image a filter kernel can process. E.g., a 3×3 filter kernel applied directly to an image has a receptive field of 3×3. However, another 3×3 filter kernel applied to the output of the first kernel processes a 5×5 window of the input image. Increasing the depth of a network increases the receptive field linearly. Using max-pooling or strided convolutions can quickly expand the receptive field. 6, 86, 106

ReLU A rectified linear unit (ReLU) is a one-dimensional function that sets any incoming negative values to zero and does not alter positive values (one-sided clipping). For CNNs, ReLU is the default activation function. It played an important role in the resurgence and now widespread application of neural networks in deep learning. In contrast to other typical activation functions, such as the hyperbolic tangent or the sigmoid function, a ReLU does not saturate for positive values, leading to better gradient information which facilitates the training of deep networks. 5

residual connection A residual connection adds the input of a layer f to its output $f(x)$. This way, low-level features from shallower layers can be passed to high-level features in deeper layers. Furthermore, it is assumed that learning is facilitated because only a residual mapping needs to be learned by the network. Moreover, the use of residual connections in a deep network essentially transforms this deep network into an ensemble of many shallower networks. 21

ResNet A ResNet is a widely adopted CNN architecture utilizing residual connections. Shallower ResNets, designed for datasets with smaller images, such as Cifar-10,

consist of several sequences of basic blocks. Deeper networks, such as the ResNet-50, employ bottleneck blocks. The training accuracy of other deep networks used before 2015 (the year the ResNet paper was published) often dropped after a certain number of convolution layers. In contrast, ResNets can be trained with an arbitrarily large depth without a decrease in training accuracy. 21

robustness In this work, a network is said to be robust when its output does not change when the input is slightly altered, e.g., by an adversarial attack. 74

simple cell A simple cell is a specific neuron of the visual cortex that only reacts to edges or lines with a specific orientation within the cell's receptive field. 5

stack Stacks are sequences of blocks. Usually, they are separated by subsampling operations, either by blocks with strided convolutions or strided pooling layers. 27

stage Similar to stacks in image classification CNNs, stages are sequences of blocks – or single blocks – of FCNs. 106

stem In a CNN, the first sequence of convolution layers and subsequent non-linearities is called stem. 27

vectorize Vectorization transforms an multi-dimensional input tensor $\mathbf{T} \in \mathbb{R}^{h \times w \times d}$ into a one-dimensional vector $\vec{x} = vect(\mathbf{T}) \in \mathbb{R}^{h \cdot w \cdot d}$. 18

Philipp Grüning

Wissenschaftlicher Lebenslauf

Studium

10/2009 - 09/2012	**Bachelor Medizinische Ingenieurswissenschaft**, *Universität zu Lübeck*, Lübeck, Deutschland, Abschlussnote: 1.6
10/2012 - 06/2016	**Master Informatik (Robotik und Automation)**, *Universität zu Lübeck*, Lübeck, Deutschland, Abschlussnote: 1.3
04/2017 - heute	**Doktorand**, *Universität zu Lübeck, Institut für Neuro- und Bioinformatik*, Lübeck, Deutschland

Berufserfahrung

11/2016 - 03/2017	**Softwareentwickler**, *Fraunhofer-Einrichtung für Marine Biotechnologie und Zelltechnik EMB*, Lübeck, Deutschland
02/2017 - 12/2021	**Softwareentwickler**, *Pattern Recognition Company GmbH*, Lübeck, Deutschland
04/2017 - heute	**Wissenschaftlicher Mitarbeiter**, *Universität zu Lübeck, Institut für Neuro- un Bioinformatik*, Lübeck, Deutschland

Schulausbildung

08/1994 - 06/1998	**Grundschule**, *Grundschule Bokel*, Bokel, Deutschland
08/1998 - 06/2007	**Gymnasium**, *Marienschule*, Lippstadt, Deutschland

Lübeck, 30.12.2022, Philipp Grüning