Non-linear Neuron Modeling using Padé Approximants with Applications to Single Image Super-Resolution and Image Compression

by

Onur Keleş

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Doctor of Philosophy

in

Electrical and Electronics Engineering

KOÇ ÜNİVERSİTESİ

February 28, 2025

Non-linear Neuron Modeling using Padé Approximants with Applications to Single Image Super-Resolution and Image Compression

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a doctoral dissertation by

Onur Keleş

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Prof. Ahmet Murat TEKALP (Advisor)

_____

Assoc. Prof. İbrahim Aykut ERDEM

_____

Assist. Prof. Zafer DOĞAN

_____

Prof. Mehmet Erkut ERDEM

_____

Prof. Hasan Fehmi ATEŞ

Date: _____

26 yıldan uzun süren öğrencilik hayatımı destekleyen tüm aileme ve destekleriyle ailem gibi yakın hissettiren herkese...

ABSTRACT

Non-linear Neuron Modeling using Padé Approximants with Applications to Single Image Super-Resolution and Image Compression
Onur Keleş
Doctor of Philosophy in Electrical and Electronics Engineering
February 28, 2025

It is fairly recent for artificial neural networks to gain extreme popularity. However, the building blocks of artificial neural networks, the perceptrons, have been in existence for more than eight decades, and convolution operation has been used in networks for more than thirty years. The popularity of neural networks comes not only from their success in solving many problems they are applied to, but also from their theoretically guaranteed convergence to the solutions under certain conditions. The universal approximation theorem tells that for any required mapping with any desired accuracy, there is a neural network that achieves it, provided with sufficient hidden units. This statement is an existence theorem; it does not specify any feature about the network. Therefore, the research community developed different strategies to fulfill the predictions of the theorem such as developing hundreds of non-linear activation functions and proposing more advanced neuron models. In this thesis, we propose a novel, more powerful and inherently non-linear neuron model, called Padé approximant neuron, or *Paon* in short. As the name implies, *Paon* uses the Padé approximant to calculate the rational function approximation on the learned locations of the input features, increasing the representation and learning capacity of the network as well as its non-linear power and receptive field. Moreover, coming in two variants as solutions for the possible singularity problem of rational approximation, *Paon*s are able to replace, and are a super set of, previously proposed neuron models, offering adaptability in various configurations. Experiments on the single image super-resolution and image compression problems demonstrate that *Paon*s surpass their competitors when compared in equal conditions in terms of number of parameters, and are able to bring performance increase even when direct replacement and reduction in number of layers are the cases in point.

ÖZETÇE

Pade Yaklaşımlarını Kullanarak Doğrusal Olmayan Nöron Modelleme ile Tek Görüntü Süper Çözünürlük ve Görüntü Sıkıştırma Uygulamaları

Onur Keleş

Elektrik ve Elektronik Mühendisliği, Doktora

28 Şubat 2025

Yapay sinir ağlarının aşırı popülerlik kazanması oldukça yenidir. Ancak yapay sinir ağlarının yapı taşları olan perseptronlar seksen yıldan uzun süredir varlığını sürdürmektedir ve evrişim işlemi ağlarda otuz yıldan uzun süredir kullanılmaktadır. Sinir ağlarının popülerliği yalnızca uygulandıkları her problemi çözmedeki başarılarından değil, aynı zamanda belirli koşullar altında çözümlere kuramsal olarak güvenceli yakınsamalarından da kaynaklanmaktadır. Evrensel yaklaşım yasası, istenen herhangi bir doğrulukla gerekli herhangi bir eşleme için, yeterli gizli birimlere sahip olduğu sürece bunu başaran bir sinir ağı olduğunu söyler. Bu ifade bir varoluş yasasıdır; ağ hakkında hiçbir özellik belirtmez. Bu nedenle, araştırmacılar yasanın öngörülerini yerine getirmek için doğrusal olmayan yüzlerce etkinleştirme fonksiyonu geliştirmek ve daha gelişmiş nöron modelleri önermek gibi farklı izlemler geliştirmiştir. Bu tezde, Pade yaklaşıklık nöronu veya kısaca *Paon* adı verilen yeni, daha güçlü ve doğası gereği doğrusal olmayan bir nöron modeli öneriyoruz. Adından da anlaşılacağı gibi *Paon*, giriş özelliklerinin öğrenilen konumlarında oransal fonksiyon yaklaşımını hesaplamak için Pade yaklaşımını kullanır ve ağın gösterim ile öğrenme özgücüyle birlikte doğrusal olmayan gücünü ve alım alanını artırır. Dahası, oransal yaklaşımın olası tekillik sorunu için iki farklı çözümle gelen *Paon*lar, daha önce önerilen nöron modellerinin yerini alabilir ve çeşitli yapılandırmalarda uyarlanabilirlik sunan bir süper küme oluşturabilir. Tek görüntü süper çözünürlük ve görüntü sıkıştırma sorunları üzerindeki deneyler, *Paon*ların değişken sayısı açısından eşit koşullarda karşılaştırıldığında rakiplerini geride bıraktığını ve doğrudan değiştirme ile katman sayısında azaltma gibi durumlar söz konusu olduğunda bile başarım artışı sağlayabildiğini göstermektedir.

ACKNOWLEDGMENTS

First of all, I would like to express my deepest gratitude to one of the greatest opportunities in my academic career, if not *the greatest*, my advisor Prof. Ahmet Murat Tekalp. His guidance not only allowed me to enter and master a field I had been passionate about since before my master's studies, but also enabled me to make meaningful contributions to it, culminating in this Ph.D. journey. In fact, this long journey was even impossible to begin without his support, and words cannot fully express my appreciation. I also wish to extend my sincere thanks to my thesis jury members—Assoc. Prof. İbrahim Aykut Erdem, Assist. Prof Zafer Doğan, Prof. Mehmet Erkut Erdem, and Prof. Hasan Fehmi Ateş—for their constructive comments and valuable suggestions that have significantly improved this work.

I am grateful to Ogün Kırmemiş and Gonca Bakar, my close friends since preparatory school, who studied with Prof. Tekalp before I did and, recognizing my keen interest in deep learning, introduced me to him. I would also like to thank the former members of the Multimedia, Vision and Graphics Laboratory (MVGL)—Ege, Buket, Tuğçe, Öykü, and Nasrin—for creating the best laboratory atmosphere I have ever experienced.

I would like to thank Codeway Digital Services not only for encouraging and supporting me to pursue this degree but also for creating an amazing working environment that allowed me to meet incredible people, including (but not limited to) "Sat", Volkan, Mert, Zeynel, Barış, and many many others.

I would like to genuinely thank Mustafa Akın Yılmaz "başkan", who has been my colleague, classmate, teaching assistant, collaborator, and currently (but not finally) my co-worker. Beyond these formal titles, he is a dear friend, and I truly value our conversations on both academic and everyday topics.

My heartfelt thanks, gratitude, appreciation and admiration go to Melis Sucuoğlu, whose supportive, fun and wholehearted presence not only helped me overcome many challenges during this journey, but also enriched my experience throughout this endeavor.

I extend my deepest gratitude to my family, who supported every decision I made. Without their sincere and encouraging support, I would never have found the strength to start, continue, or complete anything I have done so far. I feel indescribably fortunate to have them in my life.

I would like to thank and acknowledge TÜBİTAK for supporting me during my Ph.D. via research projects "*Image and Video Processing with Deep Learning*" (217E033) and "*Explainable Deep Learning Approaches for Image and Video Restoration and Compression*" (120C156).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AI	Artificial Intelligence
BD	Bjøntegaard Delta
BPG	Better Portable Graphics
BPP	Bits-per-Pixel
BSD	Berkeley Segmentation Dataset
CLIC	Challenge on Learned Image Compression
CNN	Convolutional Neural Network
COCO Dataset	Common Objects in Context Dataset
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
codec	coder-decoder
DCN	Deformable Convolutional Network
DF2K Dataset	DIV2K and Flickr2K Datasets
DIV2K Dataset	Diverse 2K Dataset
EDSR	Enhanced Deep Super-Resolution
ELIC	Efficient Learned Image Compression
FLOPS	Floating Point Operations per Second
GDN	Generalized Divisive Normalization
GELU	Gaussian Error Linear Unit
GIS	Greedy Iterative Search
GOP	Generalized Operational Perceptron
GPU	Graphics Processing Unit
ICIP	International Conference on Image Processing
KAN	Kolmogorov-Arnold Network

LPIPS	Learned Perceptual Image Patch Similarity
MAC	Multiply-Accumulate
MLIC	Multi-Reference Entropy Model Learned Image Compression
MLP	Multi-layer Perceptron
MSE	Mean Squared Error
MS-SSIM	Multi-scale SSIM
ONN	Operational Neural Network
PAU	Padé Activation Unit
PSNR	Peak Signal-to-Noise Ratio
RB	Residual Block
RD	Rate-Distortion
ReLU	Rectified Linear Unit
ResNet	Residual Network
RGB	Red-Green-Blue
S4 Models	Structured State Space Sequence Models
SNR	Signal-to-Noise Ratio
SOL	Self-ONN Layer
SOR Block	Self-organized Residual Block
SSIM	Structural Similarity
Self-ONN	Self-organized Operational Neural Network
SuperONN	Super Self-organized Operational Neural Network
VAE	Variational Autoencoder
VGG	Visual Geometry Group
WRB	Wide Residual Block

Chapter 1

INTRODUCTION

1.1 History and Motivation

Artificial neural networks have gained extreme popularity. From serving as an artificial intelligence (AI) assistant that follows the instructions given [Ouyang et al., 2022] to generating text-based [Rombach et al., 2022] or personalized [Ruiz et al., 2023] images, they have become an integral part of daily life. Moreover, neural networks have been applied to more complex use cases. They ended up being a writer of several academic studies [Transformer and Zhavoronkov, 2022], creator of a musical video clip[1] [Brooks et al., 2024], a coding-capable developer[2], a superhuman level Go [Silver et al., 2016] and chess player [Silver et al., 2017], a job recruiter[3], and a highly accurate protein structure predictor [Jumper et al., 2021]. In fact, artificial neural networks have become so widespread and an inseparable part of life that they were the key element in earning the 2024 Nobel Prizes in Physics and Chemistry [NobelPrize.org, 2024].

Although this popularity of neural networks is fairly recent, the building blocks of artificial neural networks have been in existence for more than eight decades. In 1943, Warren McCulloch and Walter Pitts proposed the "logical calculus for nervous activity" [McCulloch and Pitts, 1943] to "treat neural events and the relations among them using propositional logic". They made some assumptions that even

[1] https://www.youtube.com/watch?v=-Nb-M1GAOX8

[2] https://github.com/features/copilot

[3] https://techcrunch.com/2024/10/29/linkedin-launches-its-first-ai-agent-to-take-on-the-role-of-job-recruiters/

the majority of the models today satisfy, such as "the structure of nets does not change in time" [McCulloch and Pitts, 1943]. That paper did not consider "learning" because of the claim that "we can substitute equivalent fictitious nets composed of neurons whose connections and thresholds are unaltered" [McCulloch and Pitts, 1943]. Frank Rosenblatt proposed the perceptron [Rosenblatt, 1957], which is "an electronic or electromechanical system which will learn to recognize similarities or identities between patterns of optical, electrical, or tonal information, in a manner which may be closely analogous to the perceptual processes of a biological brain". He suggested that such a system "depends on probabilistic rather than deterministic principles for its operation, and gains reliability from the properties of statistical measurements obtained from large populations of elements" [Rosenblatt, 1957], which is the main approach for training an artificial neural network today. In one of his later works, Rosenblatt even mentioned the "back-propagating error-correction procedures" [Rosenblatt, 1961], which is the key element for networks to learn, today known as "back-propagation" [Rumelhart et al., 1986], essentially an efficient implementation of the chain rule.

At the end of the 1980s, convolution operation started to enter the neural network literature. The first use was by Homma et al. [Homma et al., 1987] on the speech recognition task with a signal processing approach of a filter, followed by Waibel et al. [Waibel et al., 1989] on phoneme recognition. Later, the influential study by LeCun et al. [LeCun et al., 1989] used convolution and back-propagation on 2D images directly to extract digits from handwritten zip codes. Those works somehow indicated that convolutional layers are more advantageous compared to fully connected networks. Because, for example, to scan the entire input of a 16×16 image, a single fully connected layer requires 256 weights, whereas a convolutional layer with a 5×5 kernel only uses 25. This convolutional approach was commonly used in computer vision, and the seminal work from 2012 [Krizhevsky et al., 2012] using convolutional neural networks (CNNs) triggered the "AI boom".

The popularity of neural networks is fueled not only by their success in solving many problems which they are applied to, but also by their theoretically guaranteed

convergence to the solutions under certain conditions. In 1989, Funahashi stated that "any continuous mapping can be approximately realized by Rumelhart-Hinton-Williams' multilayer neural networks with at least one hidden layer whose output functions are sigmoid functions" [Funahashi, 1989]. In the same year, Hornik et al. claimed that their work "rigorously establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available" [Hornik et al., 1989]. Therefore, they came to the conclusion that "multilayer feedforward networks are a class of universal approximators" [Hornik et al., 1989].

The universal approximation theorem tells us that for any required mapping with any desired accuracy, there is a neural network that achieves it, provided with sufficient hidden units. This statement is an existence theorem; it does not specify the depth of the network or the number of hidden units, not even the architecture. Moreover, it was known that the non-linear power of a network makes it strong because "as long as only linear elements are used, the multilayered network cannot have an ability superior to that of a two-layered network" [Fukushima, 1969], but the theorem requires "squashing functions" or "sigmoid functions" as non-linearity, which is known to be the cause of the problem of vanishing gradient [Hochreiter, 1991] for deeper networks. Therefore, the research community developed different approaches to fulfill the predictions of the theorem.

1.2 Neuron Models

1.2.1 McCulloch-Pitts Neuron Model

The most common neuron model today is the McCulloch-Pitts neuron model [McCulloch and Pitts, 1943], which was inspired by biological neurons. Each of these artificial neurons receives the input from previous neurons generally in the form of real numbers, processes it with an affine transformation, and finally applies a

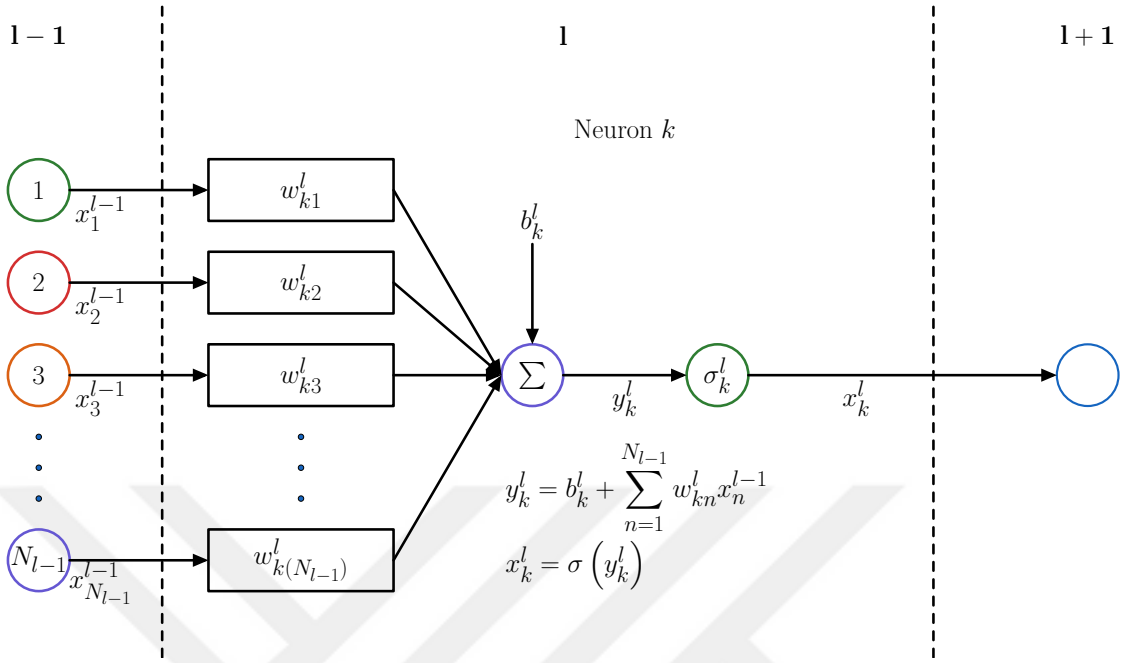


Figure 1.1: McCulloch-Pitts neuron model with the activation function $\sigma_k^l(\cdot)$.

non-linear function. An illustration of the McCulloch-Pitts neuron model is shown in Figure 1.1.

Activation functions are important for introducing non-linearity which can provide derivatives for the networks to be trained. The "all-or-none" threshold function used in the McCulloch-Pitts model [McCulloch and Pitts, 1943] brings non-linear behavior to neural networks, but it produces useless derivatives for back-propagation. Hugh Wilson and Jack Cowan modeled the activation of a neuron with logistic curve as a function of a stimulus [Wilson and Cowan, 1972]. Yann LeCun used a scaled hyperbolic tangent function in his Ph.D. thesis [LeCun, 1987] and later works [LeCun et al., 1989, LeCun et al., 1998]. Although it was used in neural networks as "activity parameter" [Householder, 1941], "analog threshold" [Fukushima, 1969], and "rectification non-linearity" [Hahnloser and Seung, 2000], rectified linear unit activation (ReLU) [Nair and Hinton, 2010] did not become popular until the work by Krizhevsky et al. [Krizhevsky et al., 2012].

There are hundreds of activation functions used in neural networks [Kunc and

Kléma, 2024]. The vast majority of them are hand-crafted activation functions composed of piecewise or continuous mathematical functions, some of which have learnable scale and/or shift parameters. This variety stems from the desire to increase the model power by adding more or different non-linear capabilities and to provide more useful derivatives during back-propagation to prevent vanishing/exploding gradients [Bengio et al., 1994] and dying neuron [Maas et al., 2013] problems.

Mathematical tools are also used as non-linearity in deep learning. One example is Padé activation units (PAU) [Molina et al., 2020]. PAU, as the name implies, uses Padé approximation to actually learn the non-linear mapping of the output features of a layer. It learns the coefficients of two polynomials of the features with given orders via back-propagation, and uses the ratio of polynomials as the activation for the whole layer.

Some approaches also add non-linearity to the networks not only by the external activation function but also through a network that learns to generate layer weights for McCulloch-Pitts neuron model. Dynamic filter networks [Jia et al., 2016] use this principle to adaptively generate all the layer weights conditioned on the input feature map. Decoupled dynamic filter networks [Zhou et al., 2021] employ the same method to generate depth-wise (spatial dynamic) and point-wise (channel dynamic) filters, reducing the computational cost of the weight generation network. In a similar but different approach, conditionally parameterized convolutions [Yang et al., 2019] and dynamic convolutions [Chen et al., 2020] use the kernel attention mechanism and aggregation on a set of filters to generate the linear kernel.

1.2.2 Non-linear Neuron Models

Researchers note that the non-linearity can be achieved by not only the activation function externally added to the output of the neuron but also more complex operations in the neuron itself. The earliest attempts tried to use "high-order correlations" [Giles and Maxwell, 1987] of inputs and "quadratic function nodes" [Volper and Hampson, 1990] as well as their efficient implementations [Shin and Ghosh, 1991, Cheung and Leung, 1991] in multi-layer perceptrons (MLPs). Those

approaches aimed for fast convergence with shallow networks due to the difficulties of training deep networks at that time [Giles and Maxwell, 1987, Shin and Ghosh, 1991].

Today, these difficulties are overcome by convolutional networks [LeCun et al., 1989] that have fewer parameters compared to their fully connected versions, and the invention of residual blocks [He et al., 2016] that allow gradients to flow more easily, thus, enabling networks with hundreds of layers to be trained. However, the research for more non-linear neurons still continues. One of the notable studies is the generalized operational perceptrons [Kıranyaz et al., 2020]. In this work, the authors suggested replacing the weighted linear combination operation in the classical neuron model with a variety of mathematical functions, such as sine and exponential, chosen by the network itself. By doing so, they claimed that available operations can be realized without multiple linear layers followed by hand-crafted non-linear functions.

Although generalized operational perceptrons provide non-linear functions by default, they are required to be reconstructed even for a smallest change, say, adding one more layer. Together with the high hardware cost of the mathematical operations, their use in neural networks was impractical. Oriented to the function approximation, the researchers proposed generative neurons [Kıranyaz et al., 2021]. Inspired by the Taylor series expansion, they suggested to use the polynomial terms of the input to approximate the necessary non-linear function in a per-layer manner. This approach was also successfully used by the author of the thesis [Keleş et al., 2021a, Yılmaz et al., 2021]. Moreover, to increase the receptive field of a layer, Kıranyaz et al. also proposed another variant, super neurons [Kıranyaz et al., 2024], in which they shift the input features of a generative neuron by a bias learned from training data.

It is important to mention that there are different strategies for modifying neural networks other than changing the activation function or modifying the neuron. The most notable approach of this strategy is to employ transformers [Vaswani et al., 2017] using the attention mechanism [Bahdanau et al., 2015]. Conceptually, from a signal processing perspective, the operations performed in a transformer can be considered as signal reconstruction by the linear combination of some functions or

vectors in a pool. Quite simply speaking, if the pool is obtained from the signal itself, the reconstruction mechanism is called "self-attention", and if the pool of vectors is constructed from different data, it is "cross-attention".

Mathematical tools also play an important role in the development of new neural network construction methods. Inspired by the Kolmogorov-Arnold representation theorem [Kolmogorov, 1957], which states that on a bounded domain a multivariate continuous function can be written as a finite superposition of single-variable continuous functions, Liu et al. presented Kolmogorov-Arnold Networks (KANs) [Liu et al., 2025]. By generalizing the original theorem, which considers constant width and depth, for arbitrary width and depths, they treated every feature element of the input as a variable, and learned a univariate function of it as a linear combination of B-splines and a basis non-linearity. Thus, they bring the inherent non-linear power to a fully connected layer, which only applies linear weights as well as a non-linear activation to the input features. Moreover, since KANs use MLPs for combining the univariate non-linear functions of input elements, they also learn the compositional structure of those functions through fully connected layers. By that, they claim "smaller KANs can achieve comparable or better accuracy than larger MLPs in function fitting task" [Liu et al., 2025].

1.2.3 Other Approaches

Although transformers can handle sequences, their quadratic complexity makes them impractical with increasing sequence length. As an alternative to transformers, state-space models [Voelker and Eliasmith, 2018, Gu et al., 2021], structured state space sequence (S4) models [Gu et al., 2022] and Mamba [Gu and Dao, 2024] were proposed. They use state-space models from the control theory to map the sequence to a latent space, and their linear complexity compared to that of quadratic one in transformers makes them more practical for especially long-sequence modeling.

Hardware logic also affects neural network architectures. Inspired by electronics, Petersen et al. proposed logic gate networks [Petersen et al., 2022, Petersen et al., 2024]. They made logic gate arithmetic differentiable by taking a couple of necessary

steps such as choosing optimal gates using probabilities obtained from the softmax operator and relaxing the logic gate operations by their equivalent real-valued parts. Combined with the efficient execution of the logic gates at the hardware level, they reported extreme speeds of inference time.

1.3 Contributions and Organization

In this thesis, we propose a novel, more powerful and inherently non-linear neuron model, called Padé approximant neuron, or *Paon* in short. As the name implies, *Paon* uses the Padé approximant to calculate the rational function approximation on the learned locations of the input features, increasing the representation and learning capacity of the network as well as its non-linear power. Since Taylor series expansion is a special case of Padé approximants, it presents a more robust and general alternative to other neuron models previously introduced. Moreover, it is equipped with a module that shifts the input features, thus increasing the effective receptive field in a layer. The contributions of this thesis can be summarized as follows:

- We propose a new neuron model called *Paon*, employing the Padé approximant. Equipped with rational function approximation, it improves the learning and non-linear capacity of a layer.

- Its feature-shifter module increases the effective receptive field of the layers, bringing even more representation power.

- We show that the presented neuron model, *Paon*, is a super set of previously known neuron models.

- Our experiments on single image super-resolution and image compression tasks show that the networks containing *Paon*s are superior both when the comparison is done with the same number of layers and when the network with *Paon*s has fewer layers.

The thesis is organized as follows. In Chapter 2, we talk about the previous art on the non-linear neuron models. Chapter 3 includes two works by the author [Keleş et al., 2021a, Yılmaz et al., 2021] using one of the neuron models mentioned in the prior art. Chapter 4 introduces *Paon*s by passing through its motivation, the problems encountered and their solutions during its development, and its superiority. This chapter is based on the paper [Keleş and Tekalp, 2024] of the thesis author. Chapter 5 presents the experiments with *Paon*s on single-image super-resolution and image compression problems. This chapter is also based on the paper [Keleş and Tekalp, 2024], and contains additional improvements and experiments on the aforementioned paper. Chapter 6 concludes the thesis with a discussion and possible future directions for the method.

Chapter 2

PRIOR STATE-OF-THE-ART ON NON-LINEAR NEURON MODELS

In this chapter, previously known non-linear neuron models are examined. Although there have been earlier attempts such as high-order correlations [Giles and Maxwell, 1987] and quadratic function nodes [Volper and Hampson, 1990], this chapter looks closely at more recent and advanced neurons. Specifically, the general operational perceptron is introduced first. Then, generative neurons will be explained. Finally, more recent super neurons are discussed.

2.1 Generalized Operational Perceptrons and Operational Neural Networks

Operational neural networks (ONNs) [Kıranyaz et al., 2020] were introduced with the promise of increasing the heterogeneity and non-linearity of convolutional neural networks. Kıranyaz et al. argued that biological neural systems contain various types of neurons and operations, which is something that must be emulated by computational neurons limited by linear operations. To do that, the authors proposed generalized operational perceptrons (GOPs) [Kıranyaz et al., 2020] as the building block of ONNs.

To understand the working principle of GOPs, a general presentation of the neuron is shown in Figure 2.1. Typically, neurons process information from the previous layer with weights on the connections. Mathematically, the processing function of the neuron k in the layer l connected to the neuron i in layer $l-1$ with weight w_{ki}^{l} can be written as $p_{k}^{l}\left(\cdot, w_{ki}^{l}\right)$. In that case, the gathering function of that neuron can be denoted as g_{k}^{l}. After being collected and adding an additional bias

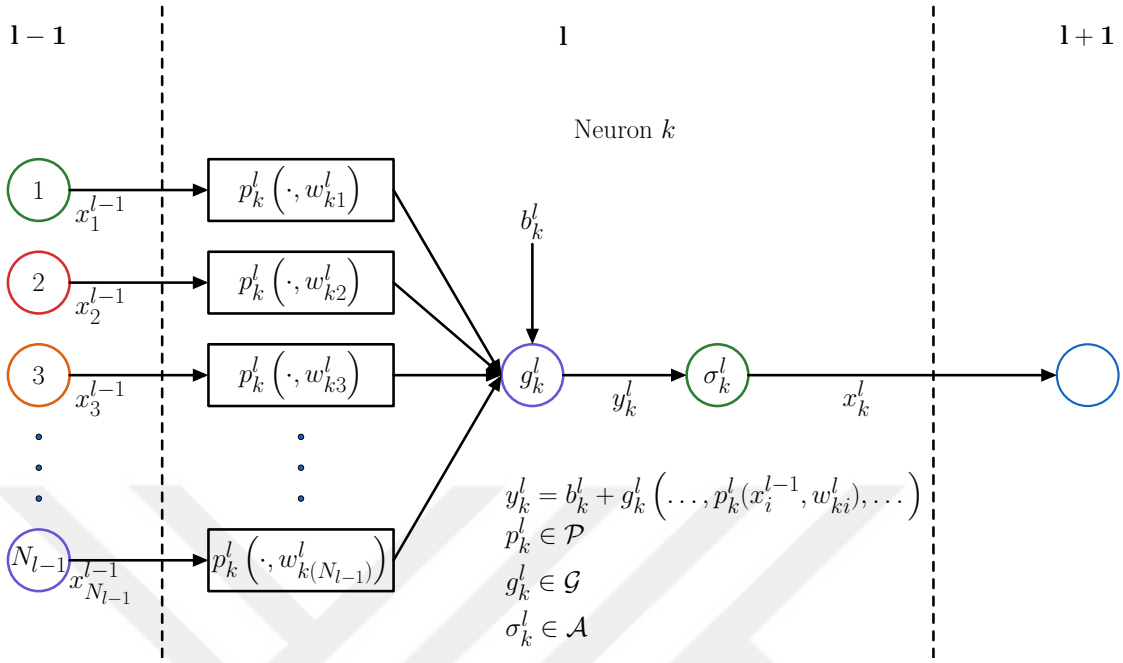


Figure 2.1: General neuron with processing function $p_k^l(\cdot,\cdot)$, gathering function $g_k^l(\cdots)$, and activation function $\sigma_k^l(\cdot)$.

term, the information passes through a non-linear activation function σ_k^l, and the operation of the neuron k is complete.

In a classical neuron, the processing operation p_k^l is the multiplication of input with the weight w_k^l, and the gathering operation g_k^l is the summation of the weighted features. On the other hand, GOPs can use any suitable mathematical function in its set of processing functions $\mathcal{P}$, the set of gathering functions $\mathcal{G}$, and the set of activation functions $\mathcal{A}$. In the study by Kıranyaz et al. [Kıranyaz et al., 2020], the processing and gathering functions are called the "nodal operator" and "pooling operator", respectively. The sets of functions proposed for the processing and gathering functions are given in Table 2.1 and Table 2.2, respectively. It is easy to see that since the operator sets of GOPs contain multiplication and summation, it makes them a super set of classical neurons; they can reduce themselves to the common neural unit when it is the optimum case.

This flexibility and variation of GOPs ensure a fast convergence to the desired

Table 2.1: Processing functions or nodal operators.

Processing Functions	$\mathcal{P} = \{p_k^l\left(x_k^{l-1}, w_{ki}^l\right)\}$
Multiplication	$w_{ki}^l x_k^{l-1}$
Exponential	$e^{w_{ki}^l x_k^{l-1}} - 1$
Second Order	$w_{ki}^l \left(x_k^{l-1}\right)^2$
Third Order	$K w_{ki}^l \left(x_k^{l-1}\right)^3$
Sine	$\sin\left(K w_{ki}^l x_k^{l-1}\right)$
Hyperbolic sine	$\sinh\left(K w_{ki}^l x_k^{l-1}\right)$
Sinc	$\sin\left(K w_{ki}^l x_k^{l-1}\right) / x_k^{l-1}$
Chirp	$\sin\left(K_C w_{ki}^l \left(x_k^{l-1}\right)^2\right)$
Gauss	$w_{ki}^l e^{-w_{ki}^l \left(x_k^{l-1}\right)^2}$
Derivative of Gaussian	$w_{ki}^l x_k^{l-1} e^{-w_{ki}^l \left(x_k^{l-1}\right)^2}$

Table 2.2: Gathering functions or pooling operators.

Gathering Functions	$\mathcal{G} = \{g_k^l\left(\ldots, p_k^l(x_i^{l-1}, w_{ki}^l), \ldots\right)\}$
Summation	$\sum_{k=1}^{N_l} p_k^l(w_{ki}^l, x_k^{l-1})$
1-Correlation	$\sum_{k=2}^{N_l} p_k^l\left(w_{ki}^l, x_k^{l-1}\right) p_k^l\left(w_{(k+1)i}^l, x_k^{l-1}\right)$
2-Correlation	$\sum_{k=3}^{N_l} p_k^l\left(w_{ki}^l, x_k^{l-1}\right) p_k^l\left(w_{(k+1)i}^l, x_k^{l-1}\right) p_k^l\left(w_{(k+2)i}^l, x_k^{l-1}\right)$
Maximum	$\max_k p_k^l(w_{ki}^l, x_k^{l-1})$
Median	$\text{median}_k\, p_k^l(w_{ki}^l, x_k^{l-1})$

function, especially when the target mapping contains one or more functions in processing and/or gathering operations. However, it is impossible to know the target function before reconstruction, and hand-picking those operators may not lead to the desired mapping. To construct ONNs, Kıranyaz et al. [Kıranyaz et al., 2017] proposed the greedy iterative search algorithm (GIS). In GIS, a maximum network depth and a training objective such as the mean squared error (MSE) are determined. Later, starting from the output layer of the network, random sets are assigned to the layer and the training objective is calculated by a few forward and backward passes of the training data. The set of operators that gives the closest objective to the desired one is selected, and the same procedure is repeated for the previous layer. If the target or maximum depth is reached, the first search is stopped. The second GIS is then performed, as the last layer was actually assigned randomly, and the evaluation of the operator set is performed for the output layer while the others are fixed. The process continues until either the objective is satisfied or the maximum determined depth is reached. After the network architecture and operations of each layer are determined, the actual training begins.

2.2 Generative Neurons and Self-Organized Operational Neural Networks

The authors of the study [Kıranyaz et al., 2021] state that even if GOPs bring higher heterogeneity and non-linearity to the networks they are used, they have certain drawbacks. First of all, determining only the network architecture, GIS is a very expensive and sensitive algorithm. If a different training objective and/or maximum depth is chosen as the criterion, the algorithm has to be run again to determine the new architecture. Together with the cost of the GIS algorithm, the expense of operators in the processing and gathering function sets also increases the total training time of ONNs since they are more expensive than multiplication and addition. Moreover, although the network has more variation compared to those built with the classical neuron model, it cannot yet achieve the ultimate heterogeneity because all neurons in a layer have to share a common set. Finally, since the proper

operations cannot be determined for a problem in advance, candidate operator sets may not have the necessary function set at all, making the convergence harder or even impossible.

Inspired by the Taylor series expansion, Kıranyaz et al. proposed generative neurons [Kıranyaz et al., 2021], addressing the aforementioned limitations. Taylor series approximation of an infinitely differentiable function $f(x)$ at a point a can be mathematically written as

$$\begin{aligned} f(x) &= \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \\ &= f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots, \end{aligned} \tag{2.1}$$

where $f^{(n)}(a)$ denotes the n^{th} order derivative of the function f at point a. For many practical purposes, the expansion can be truncated so that it is of some order q at most. The truncated series around the point a can be written as

$$\begin{aligned} f(x) &\approx \sum_{n=0}^{q} \frac{f^{(n)}(a)}{n!}(x-a)^n \\ &= f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(q)}(a)}{q!}(x-a)^q. \end{aligned} \tag{2.2}$$

If we rewrite Equation (2.2),

$$\begin{aligned} f(\mathbf{w}, x) &= \sum_{n=0}^{q} w_n (x-a)^n \\ &= w_0 + w_1(x-a) + w_2(x-a)^2 + \cdots + w_q(x-a)^q. \end{aligned} \tag{2.3}$$

Equation (2.3) forms the basis for the processing function of generative neurons [Kıranyaz et al., 2021]. The terms w_n for $n = 1, 2, \ldots, q$ are the learnable weights of a generative neuron, and w_0 serves as the bias term. They do not assume any particular functional shape, such as sine, exponential, and hyperbolic functions, as GOPs do. Generative neurons do not require a time-consuming operator search step such as GIS as they organize themselves to the required operation via back-propagation during training. Thus, networks having generative neurons are called self-organized operational neural networks, or Self-ONNs for short.

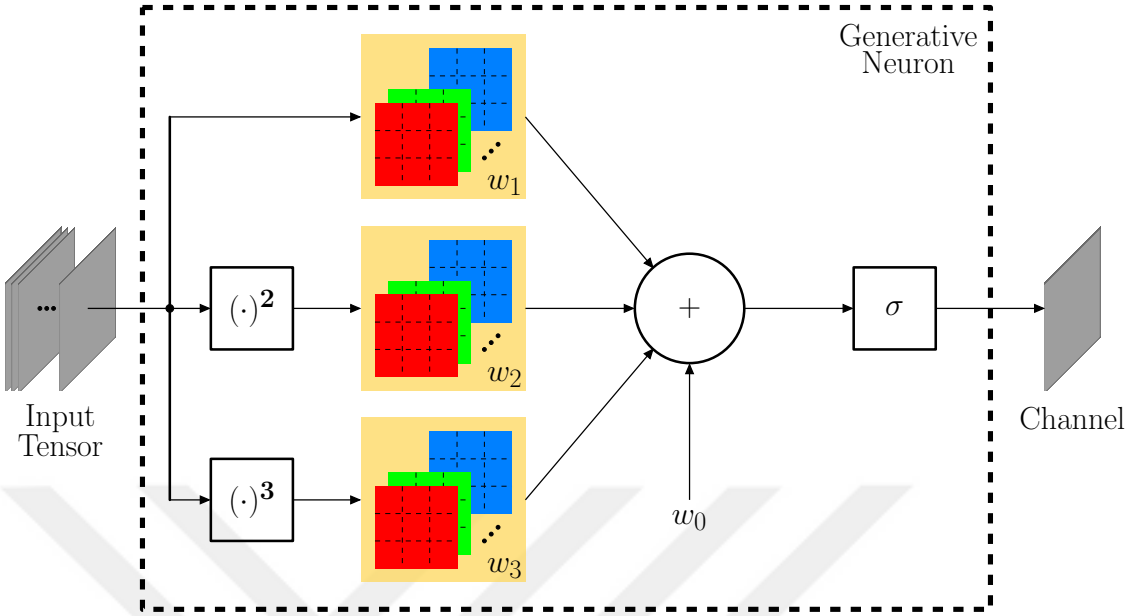


Figure 2.2: Generative neuron with $q = 3$ and σ being the activation function. Directly taken from [Keleş et al., 2021a].

The Taylor series expansion is the most accurate near the point a, on which it is differentiable and around which it is calculated. Therefore, in order to keep the approximation accurate, the values entering the generative neuron have to be around a specific point. Moreover, addition of positive integer power of input may cause stability problems by growing the output in an uncontrollable way. In order to prevent this, there are some activation functions ensuring that the values are bounded. If the Taylor series approximation is calculated around $a = 0.5$, then the sigmoid function can be used, and for the point $a = 0$, in the case of which the expansion becomes Maclaurin series approximation, tanh activation can be used. An illustration of a generative neuron is given in Figure 2.2.

Generative neurons are used on several image processing tasks such as image denoising [Malik et al., 2021] and global electrocardiogram classification [Zahid et al., 2023], as well as single image super-resolution [Keleş et al., 2021a] and image compression [Yılmaz et al., 2021] by the thesis author, which are to be inspected in Chapter 3.

2.3 Super Neurons and Super Self-Organized Operational Neural Networks

Generative neurons and Self-ONN solve the linearity of the kernels, and activation being the only powerful non-linear element. However, Kıranyaz et al. [Kıranyaz et al., 2021] claim that the proposed neuron is still limited in terms of its locality, which means that a particular element in the output features is contributed only by the vicinity of that particular element in the previous feature map. They claim that an output value at a given location can be calculated using not only the corresponding region and its vicinity on the input but also the entire activation map.

For this purpose, Kıranyaz et al. upgrade generative neurons and propose super neurons [Kıranyaz et al., 2024]. That neuron model still behaves as a generative neuron in terms of its processing and gathering functions, but the values to be processed on the feature map are sampled from different locations, which are optimized via back-propagation during training for each neuron in each different layer. By that, they claim to ensure not only better construction of the output feature map but also an increase in the receptive field without introducing too much parameter.

Figure 2.3 shows a diagram of the working principle of super neurons, in which 3×3 kernels are used for visualization purposes. Here, the kernels show the location of the processing functions, and the arrows show to which point they contribute. The shades of gray show the classical neurons, which focus on a specific location on the input to find the value at the corresponding place. For the super neuron kernels, shown in color, however, a specific feature element in the output can be contributed by different kernels at the different locations, from which the colorful arrows are directed. The locations are sampled [Jaderberg et al., 2015] at the beginning and then optimized by back-propagation during only training. By this small addition to the number of parameters, the authors of the study [Kıranyaz et al., 2024] claim that they increase the effective receptive field without increasing the model depth or kernel size. The networks employing super neurons are called super self-organized operational neural networks (SuperONN).

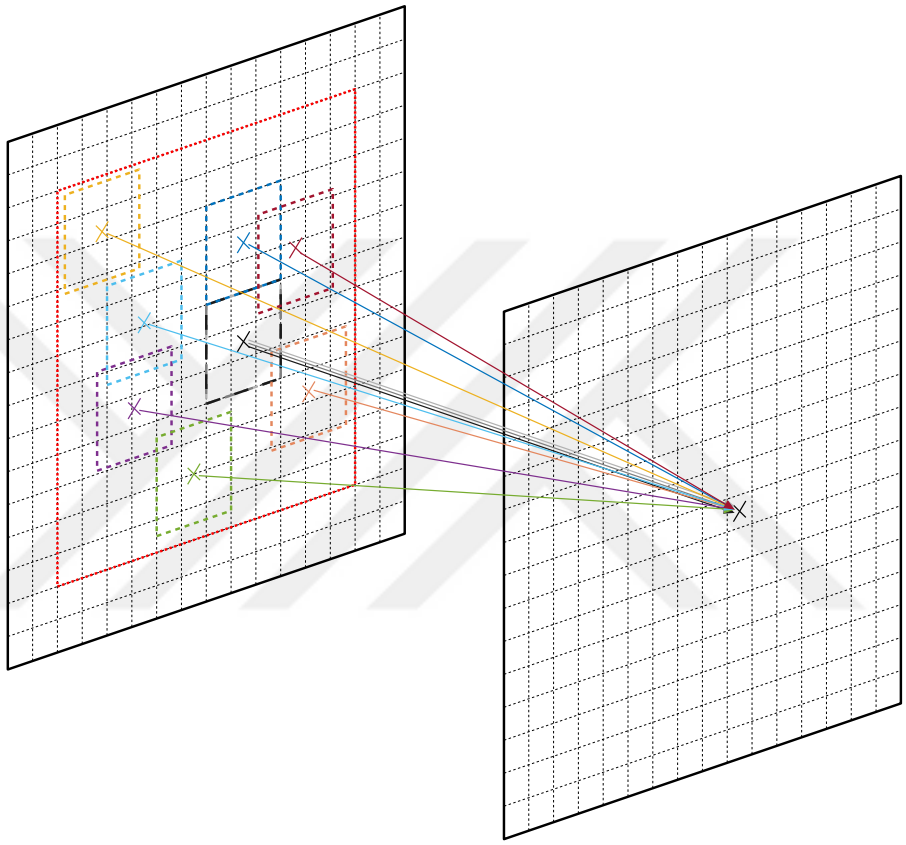


Figure 2.3: Super neuron compared to other neurons. The kernels and arrows with shades of gray show a regular kernel centered at the corresponding place of the output, whereas the colorful kernels and arrows show super neuron operations. Super neuron kernels are free to roam and to sample from sub-pixel locations on the feature map. If desired, they can be limited in terms of their search region, marked with big red square in the figure.

Chapter 3

APPLICATIONS OF SELF-ONNS TO IMAGE SUPER-RESOLUTION AND COMPRESSION

As mentioned before, generative neurons and Self-ONNs were successfully used in single image super-resolution and image compression tasks. The next two sections briefly summarize the International Conference on Image Processing (ICIP) 2021 papers on these tasks, "Self-organized residual blocks for image super-resolution" [Keleş et al., 2021a] and "Self-organized variational autoencoders (Self-VAE) for learned image compression" [Yılmaz et al., 2021], by the thesis author.

3.1 Self-Organized Residual Blocks for Image Super-Resolution

To the best of our knowledge, "Self-organized residual blocks for image super-resolution" [Keleş et al., 2021a] is the first paper to use generative neurons in the single image super-resolution problem. Since Self-ONN layers (SOLs) can replace any convolutional layer in a network, the paper [Keleş et al., 2021a] shows their superiority in solving the problem by employing a well-known architecture instead of proposing a completely new model.

To show the power of generative neurons, the enhanced deep super-resolution (EDSR) model [Lim et al., 2017] is chosen due to its simplicity. The basic architecture of EDSR has 16 residual blocks [He et al., 2016]. It has 64 neurons for each layer, except for the output layer, which has 3 neurons, and the pixel shuffler layer or "sub-pixel convolution layer" [Shi et al., 2016], whose number of neurons depends on the target increase ratio. More information on the sub-pixel convolution layer is given in Appendix A.1.

Although convolutional layers can be easily replaced by SOLs, replacement should

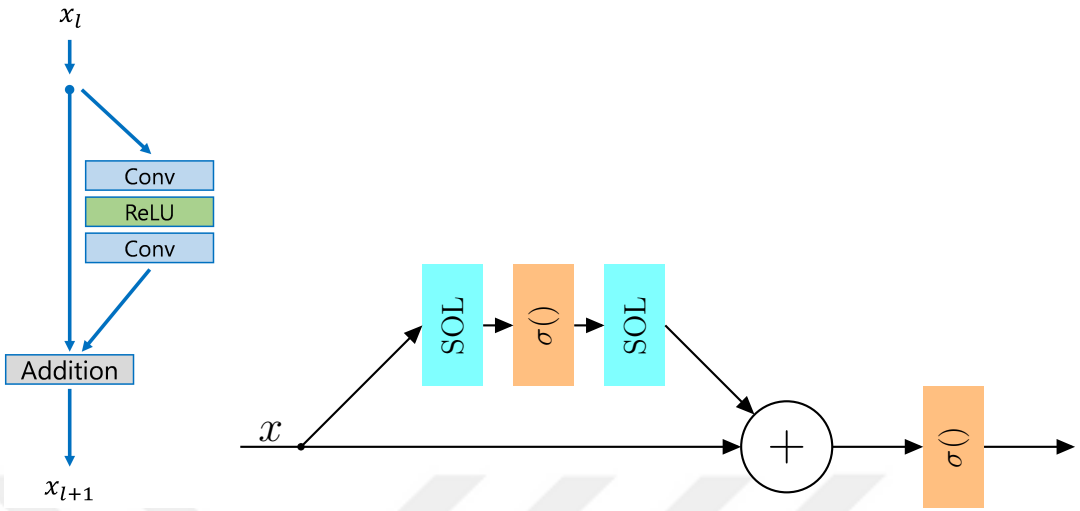


Figure 3.1: Residual blocks. (a) Residual block used in EDSR. (b) Residual block proposed for Self-ONNs with σ being the activation function. Both are directly taken from [Lim et al., 2017] and [Keleş et al., 2021a], respectively.

be done with caution. The residual block used in EDSR architecture has a single non-linear activation layer between two convolutions without any normalization layer. Therefore, after replacement, the second Self-ONN layer should not proceed with a limiting activation function in a residual block. However, summation at the end of the block may cause some values to exceed the limits required for a stable training. Thus, a final limiting activation function follows the residual connection, creating self-organized residual (SOR) block. The residual blocks used in EDSR and the study by Keleş et al. [Keleş et al., 2021a] are shown in Figure 3.1.

In training, the DIV2K dataset containing 800 training images and 100 validation images is used [Agustsson and Timofte, 2017]. During model training, 16 crops with size 48×48 are randomly selected from the low-resolution images and their corresponding high-resolution part. As a data augmentation technique, crops are randomly rotated in integer multiples of 90 degrees. Adam optimizer [Kingma and Ba, 2015] with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ tries to minimize l_1 loss for a total of 300 000 iterations. For the baseline architecture, the learning rate is 10^{-4}

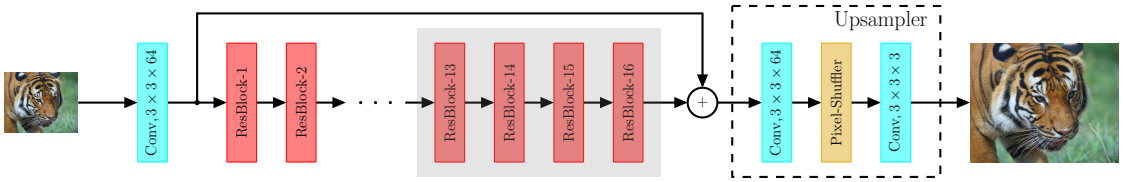


Figure 3.2: The EDSR network architecture. We replace only the residual blocks in the gray box with SOR blocks in the hybrid architecture. We also replace the regular convolutional layers in the upsampler. Directly taken from [Keleş et al., 2021a].

which is halved after 200 000 steps. The architectures containing Self-ONN layers, the starting learning rate is 1.5×10^{-4} and halved after each 100 000 iterations. For evaluation, the peak signal-to-noise ratio (PSNR) [Keleş et al., 2021b], structural similarity (SSIM) [Wang et al., 2004] and learned perceptual image patch similarity (LPIPS) [Zhang et al., 2018] scores are used. For detailed information on SSIM and LPIPS, please refer to Appendices B.1 and B.2, respectively.

The architecture of EDSR is shown in Figure 3.2. The work [Keleş et al., 2021a] proposes two approaches to insert Self-ONN layers into the EDSR architecture. The first is to change every residual block to the SOR block, and every convolutional layer, including the input, output, and those in the upsampler part, to Self-ONN layers. The second alternative is to use a hybrid architecture in terms of layers and residual blocks. Both alternatives have some design choices to investigate.

In the full Self-ONN structure, the layers are known to be superior compared to the classical layer, since the main advantage of generative neurons is to use less layer and/or neuron than normally used. Therefore, it is not necessary to stack as many layers and blocks as EDSR does. To decide on the structure of the full Self-ONN architecture, the experiments are carried out with different numbers of SOR blocks and the layer size. Table 3.1 shows the quantitative results, indicating that the best configuration among the ones tested is with 8 SOR blocks and using 64 generative neurons with $q = 3$ as the maximum order for Taylor series in each layer.

For the hybrid structure, 4 of the 16 residual blocks as well as the layers in the upsampler part are to be changed with their Self-ONN counter parts. To be

Table 3.1: Self-ONN performance by different number of SOR blocks and generative neurons per layer. Directly taken from [Keleş et al., 2021a].

Blocks (Neurons)	4 (32)	4 (64)	8 (32)	8 (64)
PSNR	34.148	34.424	34.377	34.616
SSIM	0.9608	0.9624	0.9621	0.9635
Parameters	365 059	1 448 963	586 499	2 334 311

Table 3.2: Searching for the best hybrid network architecture. Directly taken from [Keleş et al., 2021a].

SOR Blocks	First 4		Last 4	
Upsampler/Out layers	Common	SOL	Common	SOL
PSNR	34.565	34.612	34.631	34.658
SSIM	0.9632	0.9635	0.9636	0.9638

able to understand which features should be processed by stronger layers, several configurations are tested, including the ones in which the initial and final 4 residual blocks are replaced by SOR blocks and whether the upsampler part is modified or remains untouched. The results are shown in Table 3.2, indicating that the best configuration is to change the last four residual blocks with SOR blocks (in other words, change the residual blocks in the gray area in Figure 3.2), and convert every convolutional layer in the upsampler part to SOL using $q = 3$.

The networks are tested for $\times 2$ and $\times 4$ dimension increase. All three models are trained from scratch for the $\times 2$ task. The networks performing $\times 4$ super-resolution are trained in two alternative ways: (i) starting from random initialization and (ii) finetuning using the $\times 2$ version as backbone .

The quantitative results are given in Table 3.3, in which the best scores are given in bold, and the second ones are given with underlined numbers. It can be easily seen that the proposed usage of generative neurons improves the model outputs

Table 3.3: Evaluation of the models using RGB PSNR and LPIPS metrics as well as Y channel SSIM scores. The arrows next to metrics indicate that whether the higher (↑) or lower (↓) scores are better. The **bold** scores indicate the best, and the underlined scores indicate the second best results. Directly taken and modified from [Keleş et al., 2021a].

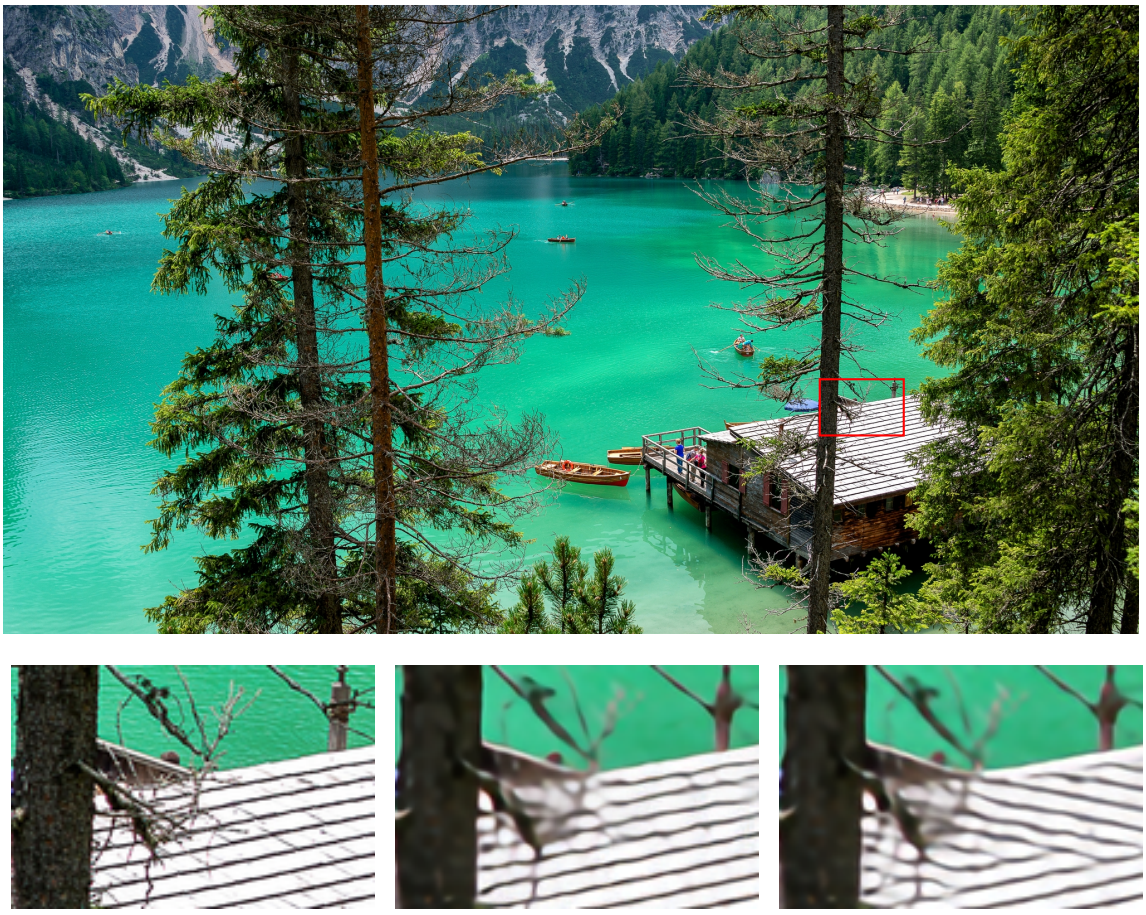
Method / Metric	EDSR			Self-ONN			Hybrid		
	×2	×4		×2	×4		×2	×4	
		Random	Finetune		Random	Finetune		Random	Finetune
PSNR (↑)	34.557	28.924	28.970	34.616	28.983	29.016	**34.658**	**29.018**	**29.064**
SSIM (↑)	0.9632	0.8857	0.8866	0.9635	0.8871	0.8875	**0.9638**	**0.8877**	**0.8886**
LPIPS (↓)	0.0896	0.2775	0.2747	**0.0873**	0.2742	0.2714	0.0890	**0.2726**	**0.2694**

in terms of both fidelity and perceptual quality in both of the architecture types. The scores clearly indicate that Self-ONN layers increase the performance of the networks compared to their baseline models. It can also be seen that the hybrid architecture performs better than the pure Self-ONN structure, possibly indicating that depth is also a crucial element for better performance in neural networks, since it also increases the total number of non-linear functions used in the model.

Quantitative results also reflect perceptual results through the LPIPS metric. An example is shown in Figure 3.3. Here, the output images show the cropped area in the original image indicated with a red box. The lines on the wings of the bird show discontinuity and have a wave-like shape in the baseline output, whereas the ground truth is free of such artifacts. However, the hybrid architecture manages to preserve not only the shape but also the continuity of the lines thanks to its stronger non-linearity which captures more details on the images. Another example is given in Figure 3.4. Again, the region in the red box has parallel lines on the roof. The baseline architecture fails to draw them correctly and confuses the direction of the lines because it lacks the power to dissolve the high frequencies. However, the pure Self-ONN architecture achieves better fidelity by properly continuing the lines in the correct direction. Those results clearly indicate that the model with stronger non-linearity has more capacity to learn the correct mapping function.

(a) (b) (c)

Figure 3.3: Visual comparison of baseline and hybrid network architecture with finetuned pre-trained model, on the image `0853.png` from DIV2K. (a) Ground truth. (b) Hybrid model. (c) Baseline. Directly taken from [Keleş et al., 2021a].



(a) (b) (c)

Figure 3.4: Visual comparison of baseline and Self-ONN trained with random initialization, on the image `0807.png` from DIV2K. (a) Ground truth. (b) Self-ONN. (c) Baseline. Directly taken from [Keleş et al., 2021a].

3.2 Self-Organized Variational Autoencoders (Self-VAE) for Learned Image Compression

To the best of our knowledge, "Self-organized variational autoencoders (Self-VAE) for learned image compression" [Yılmaz et al., 2021] is the first paper to use generative neurons in the image compression problem. Here, the purpose is to use the non-linearity that generative neurons bring to the network instead of the then-common generalized divisive normalization (GDN) layer, which is claimed to produce Gaussian-distributed latent features [Ballé et al., 2016a]. For more information on GDN, see Appendix A.2 and Ballé et al. [Ballé et al., 2016a].

In compression, the image first passes through a forward transform, followed by the quantization step. After entropy encoding, the compressed data are used, transmitted, or stored. On the receiver side, the compressed data follow the reverse path; entropy decoding, dequantization, and inverse transform. In the paper by Yılmaz et al. [Yılmaz et al., 2021], an architecture following the same data flow is chosen for the experiments. The joint autoregressive model by Minnen et al. [Minnen et al., 2018] transforms the input image to a latent representation by an analysis network composed of convolutional and GDN layers. The statistics of the latent are estimated and decoded by hyper-prior networks, which help arithmetic encoder and decoder parts compressing data more effectively. In the end, the decompressed latent features are passed through a synthesis network applying the inverse transform. The architecture is shown in Figure 3.5.

To create the Self-VAE architecture, the authors of the study [Yılmaz et al., 2021] decide to change the first three layers of the analysis and synthesis networks to SOLs having 192 channels with $q = 3$. Also, since Self-ONN layers require the data range to be limited, GDN non-linearities are changed to tanh. For fair comparison; quantization, hyper-prior networks, masked convolutions for context modeling, entropy parameters and arithmetic encoder/decoder parts [Rissanen and Langdon, 1981] remain untouched and used as in the work by Minnen et al. [Minnen et al., 2018], to which the reader is directed for more information.

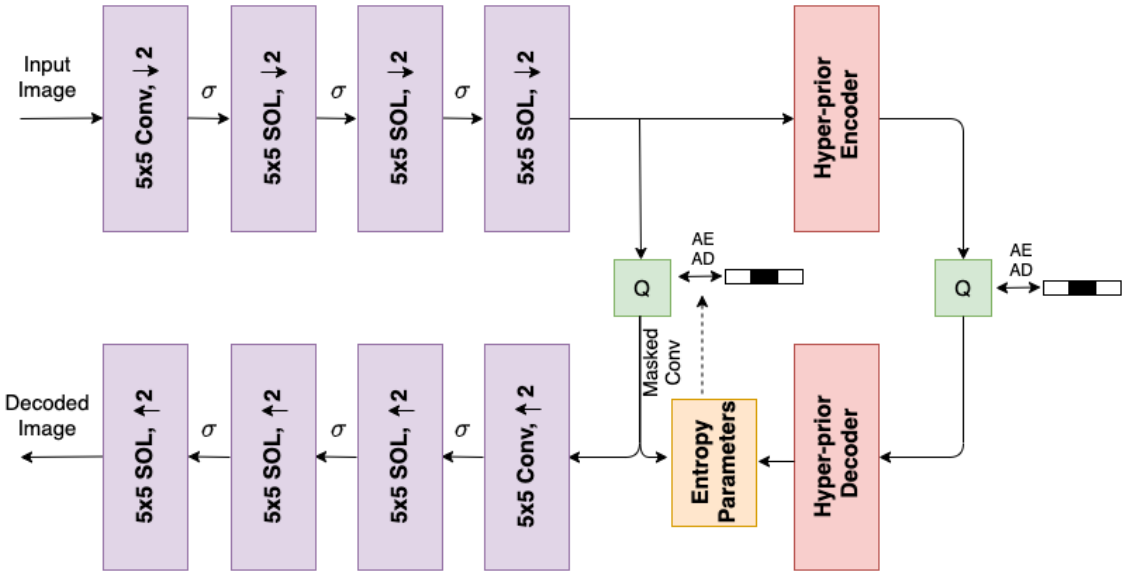


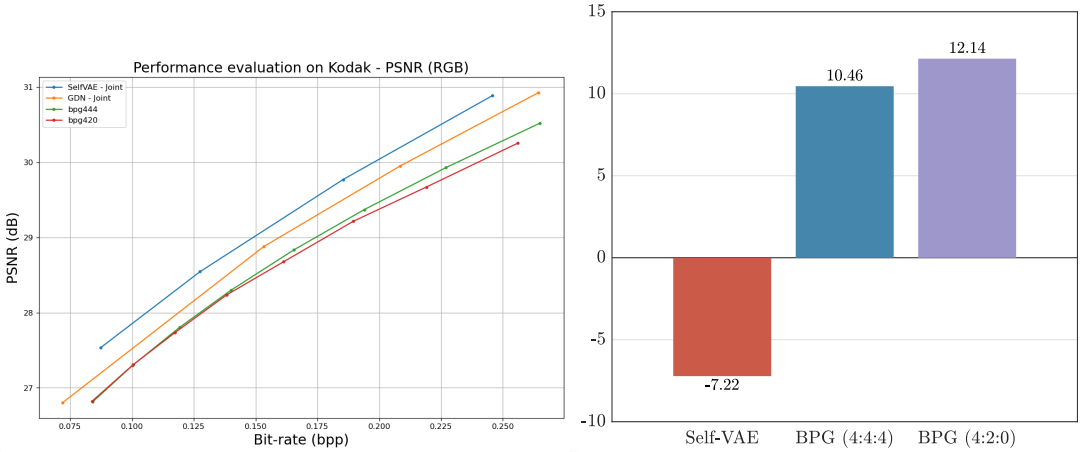
Figure 3.5: Self-VAE network with SOLs for image compression. $\downarrow 2$ and $\uparrow 2$ denote the stride for 5×5 convolution and deconvolution operations, respectively, and σ is tanh non-linearity. "Q", "AE" and "AD" refer to quantization, arithmetic encoding and arithmetic decoding, respectively. Directly taken from [Yılmaz et al., 2021].

Self-VAE architecture is trained on the Vimeo-90K dataset [Xue et al., 2019], from which 256×256 random crops are extracted with batch size 16. Adam optimizer [Kingma and Ba, 2015] is used for 1 000 000 iterations with an initial learning rate 10^{-4}, which is reduced to 10^{-5} after 500 000 iterations. For better training, gradient clipping is applied to the norm of the gradients, the maximum of which is set to 1. Adam optimizer minimizes rate-distortion (RD) loss [Ballé et al., 2018],

$$\mathcal{L} = \mathcal{L}_R + \lambda \mathcal{L}_D, \tag{3.1}$$

where $\mathcal{L}_R$ is rate loss measured in bits-per-pixel (BPP), $\mathcal{L}_D$ is MSE loss, and λ is the Lagrange multiplier chosen as $\lambda = \{0.012, 0.026, 0.042, 0.056\}$, with which the RD curve is drawn. For a brief information on the rate-distortion loss in Equation (3.1), please refer to Appendix C.1.

For evaluation, the anchor model [Minnen et al., 2018], denoted as GDN-Joint,



(a) (b)

Figure 3.6: Graphical comparison of the proposed Self-VAE with benchmarks. (a) Rate-distortion curve, which is directly taken from [Yılmaz et al., 2021]. (b) Average percent BD-rate savings calculated for RGB PSNR values for the anchor GDN-Joint. Adapted from [Yılmaz et al., 2021].

and a classical image coder-decoder (codec) known as better portable graphics (BPG) [Bellard, 2014] are used. Performance is measured on the Kodak dataset [Franzen, 2013] with RGB PSNR and BPP, and LPIPS scores calculated with AlexNet [Krizhevsky et al., 2012] and VGG [Simonyan and Zisserman, 2015] models.

Figure 3.6 shows the performance of the proposed architecture. In Figure 3.6a, rate-distortion curve of the Self-VAE shows its superiority by having higher PSNR when the bit rates are the same and/or by having lower bit rates for the same PSNR. It can also be understood from Figure 3.6b showing the bit rate savings calculated by the Bjøntegaard Delta rate (BD-rate) [Bjøntegaard, 2001]. The plot shows that the proposed Self-VAE consumes 7.22% less bit rate on average than the anchor model and much less than the classical image codec. For more information on BD calculation, please see Appendix B.3.

The superiority also shows itself in the metric scores. Table 3.4 shows PSNR and LPIPS scores for the three selected images, namely `kodim04.png`, `kodim14.png` and `kodim15.png` images in Kodak dataset. The fidelity metric PSNR is higher for

Table 3.4: PSNR and LPIPS scores for selected three Kodak images. Best results are indicated with **bold** font. For `kodim04.png` and `kodim15.png`, bit rate is approximately 0.1 BPP, for `kodim14.png` it is around 0.25 BPP. Directly taken from [Yılmaz et al., 2021].

Image	`kodim04.png`			`kodim14.png`			`kodim15.png`		
Method / Metric	BPG	GDN-Joint	Self-VAE	BPG	GDN-Joint	Self-VAE	BPG	GDN-Joint	Self-VAE
PSNR	30.03	30.51	**30.69**	27.52	27.92	**28.07**	30.34	30.62	**30.79**
LPIPS-AlexNet	0.3413	0.3565	**0.3274**	0.3459	0.3689	**0.3361**	0.2762	0.2955	**0.2636**
LPIPS-VGG	0.4075	0.3976	**0.3830**	0.3923	0.3922	**0.3654**	0.4083	0.4013	**0.3770**

the proposed method than for the rest of the compression methods by a significant margin. Moreover, Self-VAE is much better in the perceptual metric LPIPS when calculated with two different models.

The images used in Table 3.4 are shown in Figure 3.7. For `kodim04.png` image (top), it can be seen that BPG codec creates block artifacts as its compression by-product. However, the output of Self-VAE is free of such artifacts. Moreover, the GDN-Joint model produces more smooth outputs compared to the proposed model. In `kodim14.png` image, the face in the boxed area has block artifacts and color distortions in the output of the classical codec, and the output of the anchor model smooths out nearly all details. However, Self-VAE produces a more detailed face compared to its competitors. Finally, the image `kodim15.png` has compression block artifacts in both of the other approaches, whereas in Self-VAE, such an unwanted effect cannot be seen. Those visual results also indicate that the usage of Self-ONN layers increases fidelity by better reconstruction and fewer artifact generation capabilities in the resulting images.

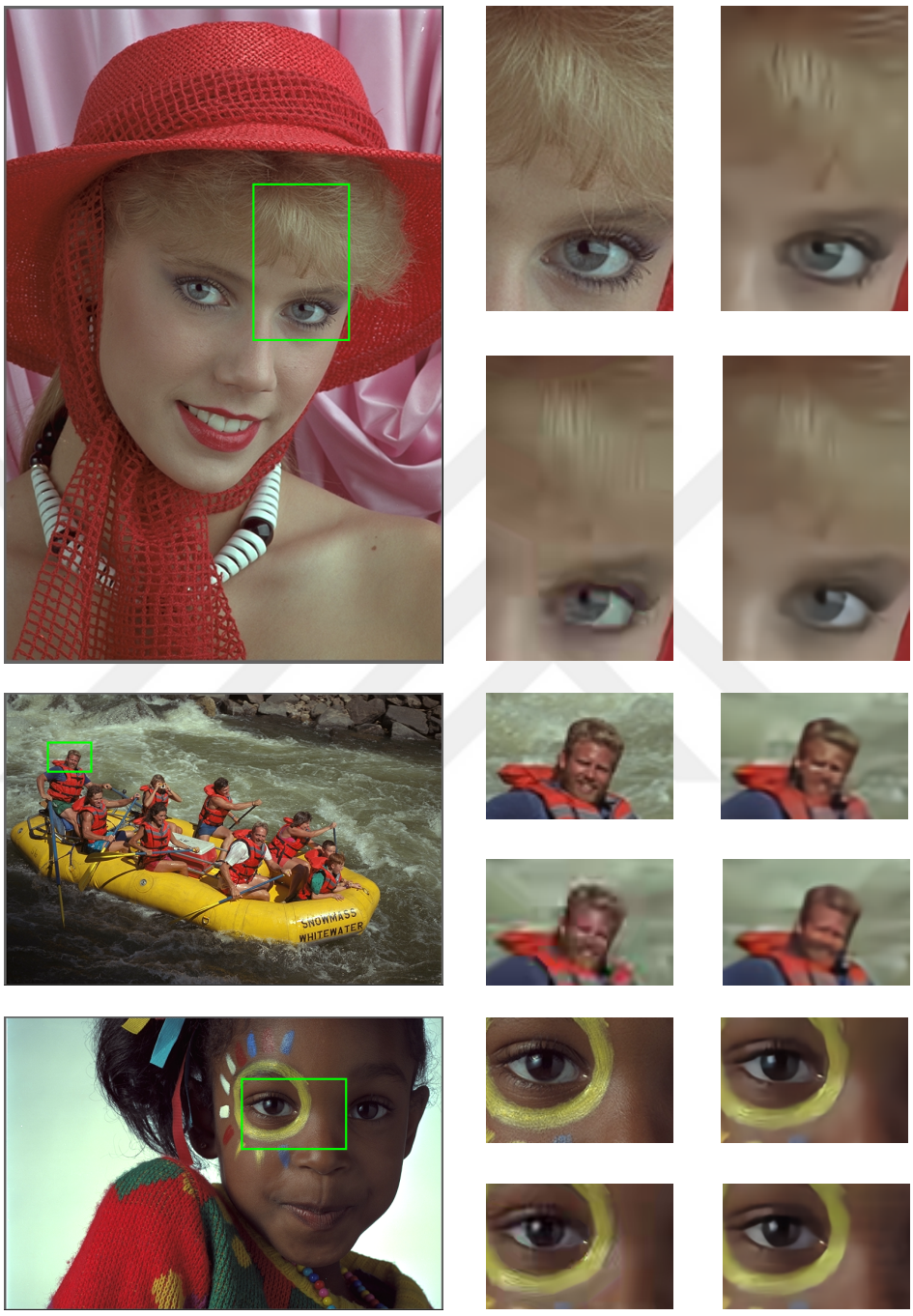


Figure 3.7: Visual evaluation of reconstructed `kodim04.png`, `kodim14.png` and `kodim15.png` images, respectively, in Kodak dataset. Crops are taken from ground truth and Self-VAE output at the top, BPG and GDN outputs at the bottom, respectively. For `kodim04.png` and `kodim15.png`, bit rate is approximately 0.1 BPP, for `kodim14.png` it is around 0.25 BPP. Directly taken from [Yılmaz et al., 2021].

Chapter 4

PADÉ APPROXIMANT NEURONS, LAYERS AND NETWORKS

This chapter introduces a novel and superior neuron model called Padé approximant neurons, or *Paon*s, in short. First, we explore the motivation and idea behind *Paon*. Then, the singularity problem of Padé approximants as a neuron model to be implemented is discussed, and two working solutions are presented. Later, the Shifter module and its integration as an effective approach to increase the receptive field of a *Paon* for convolution operation are explained. Finally, it is shown that the proposed novel neuron model is both a super set and superior to previously explained neurons. This chapter is heavily based on the author's paper [Keleş and Tekalp, 2024] with additional explanations and improvements.

4.1 Motivation and Idea

From a mathematical point of view, artificial neural networks can continuously map any finite-dimensional space to another, as suggested by the universal approximation theorem [Funahashi, 1989, Hornik et al., 1989]. Practically, the required mapping is not known a priori, but learned through multiple examples shown to network and back-propagation algorithm reducing the amount of error that the model produces. In this way, the network learns the best fitting function that approximates the true mapping.

The mapping accuracy depends not only on the number of input/output samples fed to the model but also on the capacity of the network. It generally increases with depth since deeper networks include more non-linear functions that help the model capture the nature of the solution. The more non-linearity that a network has, the

better its function approximation capability becomes.

Looking at the deep learning concept as a function approximation process, it is logical to use mathematical tools in neural networks to help them achieve it with less effort. Equipped with the Taylor series approximation, generative neurons [Kıranyaz et al., 2021] and super neurons [Kıranyaz et al., 2024] adopt this approach. However, that idea comes with requirements that limit their success. Taylor series expansion is most accurate around the point at which it is calculated. To ensure that the summation of input and its positive integer powers remains in the stable computation range and vicinity around that specific point, Kıranyaz et al. [Kıranyaz et al., 2021, Kıranyaz et al., 2024] use the hyperbolic tangent function that serves as both a limiter and an activation function. Although successful, tanh is known to cause the vanishing gradient problem [Hochreiter, 1991] that hinders the training of deep models.

This work seeks to employ a better function approximation tool which brings more non-linearity, thus capacity, without creating the aforementioned instability and vanishing gradient problems. The idea is to use Padé approximants in the basic element of neural networks.

Proposed at the end of 1800s, Padé approximant is the best approximation of a transcendental function near a specific point by a ratio of two polynomials with given orders [Padé, 1892]. Let $f_{[M/N]}(x)$ be a rational polynomial approximation for the real continuous function $f(x)$ using the M-degree polynomial $P_M(x)$ and the N-degree polynomial $Q_N(x)$ in the numerator and denominator, respectively. The Padé approximant for the target function can be written as

$$f(x) \approx f_{[M/N]}(x) = \frac{P_M(x)}{Q_N(x)} = \frac{\sum_{i=0}^{M} a_i x^i}{\sum_{j=0}^{N} b_j x^j}, \tag{4.1}$$

where a_is and b_js are the coefficients of polynomials in numerator and denominator, respectively. In this representation, there are $M+1$ unknown coefficients for $P_M(x)$ and $N+1$ coefficients for $Q_N(x)$. In order for the approximation $f_{[M/N]}(x)$ to be a

continuous approximation of a continuous function, the condition $Q_N(x) \neq 0$ should hold for all x, which means $Q_N(0) \neq 0$, thus $b_0 \neq 0$. This requirement helps the coefficients be scaled such that $b_0 = 1$. Accordingly, if we rewrite the approximation, we have

$$f_{[M/N]}(x) = \frac{P_M(x)}{Q_N(x)} = \frac{\sum_{i=0}^{M} a_i x^i}{1 + \sum_{j=1}^{N} b_j x^j} = \frac{a_0 + \sum_{i=1}^{M} a_i x^i}{1 + \sum_{j=1}^{N} b_j x^j}, \tag{4.2}$$

where there is one coefficient less to compute compared to Equation (4.1). Normally, the coefficients of the Padé approximant are found in two different ways: (i) If the target function is infinitely differentiable or is in polynomial form, then the Maclaurin series of the target function up to the order $M + N$ is equalized to Equation (4.2) and the system is solved. (ii) For the rest of the cases, the least-squares method is used. However, for the Padé approximant to be employed effectively for deep learning applications, those coefficients must be learnable.

As mentioned before, the Padé approximant is the "best" approximation of the univariate function $f(x)$ by a rational function of given order, as shown by the Montessus de Ballore theorem, which establishes uniform convergence of Padé approximants on compact subsets excluding the poles [de Montessus de Ballore, 1902]. Although there have been several works to extend this result to multivariate functions [Cuyt, 1999, Guillaume and Huard, 2000], they address several challenges for proofs of convergence under certain assumptions. However, keeping this in mind, if we treat the coefficients in front of the polynomial terms as neuron weights and the constant term on the numerator as the bias term, then we can formulate a new neuron model as follows:

$$\boxed{f_{[M/N]}(x) = \frac{P_M(x)}{Q_N(x)} = \frac{w_0 + \sum_{i=1}^{M} w_{mi} \circledast x^i}{1 + \sum_{j=1}^{N} w_{nj} \circledast x^j}} \tag{4.3}$$

where x is the input tensor, $\circledast$ denotes convolution or multiplication depending on the chosen layer being either a convolutional or a fully connected layer, w_{mi} and w_{nj}

are the neuron weights of the numerator and denominator for the input of order i and j, respectively, and w_0 is the bias term for the numerator.

This novel neuron model is called the Padé approximant neuron, or *Paon* for short. The basic mathematical definition in Equation (4.3) can be considered as a ratio of two different polynomials $P_M(x)$ and $Q_N(x)$, each of which is a summation of some power series expansion calculated at each input element separately. *Paon* incorporates integer power of the feature map in both the numerator and the denominator, bringing higher non-linearity than the classical or generative neuron.

4.2 Possible Singularity Problem

One critical aspect of a neuron model is to ensure continuous mapping. However, due to the rational nature of the Padé approximants, there is a potential for the denominator to become equal to or very close to zero. Although all the weights can be initialized such that this issue can be prevented at the beginning, gradient descent and back-propagation do not guarantee that the denominator part will remain nonzero large enough to maintain stability throughout training. To mathematically ensure that the divisor is always nonzero, in other words, the denominator does not have real roots, we propose two variants of the Padé approximant neurons.

The first variant involves the most straightforward way to eliminate real roots in the expression: taking the absolute value of each individual power in the denominator, which is also used in PAU [Molina et al., 2020] over Equation (4.2) for implementation. This makes the summation of positive integer powers strictly non-negative, and the first term on the denominator guarantees that each element in the numerator is divided by a number greater than or equal to 1. The final expression for the Padé approximant neuron with absolute value, denoted as *Paon–A*, becomes:

$$Paon\text{–}A \coloneqq f_{[M/N]}(x) = \frac{w_0 + \sum_{i=1}^{M} w_{mi} \circledast x^i}{1 + \sum_{j=1}^{N} \left| w_{nj} \circledast x^j \right|} \tag{4.4}$$

Although this modification ensures that the denominator remains positive and sufficiently large to prevent computational instability arising from division by values close to zero, it may not be a mathematically sound solution to benefit from the properties of the Padé approximants. Thus, another version is proposed to handle the singularity problem.

The second variant of the Padé approximant neuron is inspired by the work of Beckermann and Kalyagin [Beckermann and Kalyagin, 1997] introducing the smoothed Padé approximants. Although this method was initially proposed for diagonal Padé approximants, where $M = N$, we have empirically observed that it can be applied effectively in cases where $|M - N| = \{0, 1\}$ without any further modifications. This smoothed variant of the Padé approximant neuron, denoted as *Paon–S*, can be expressed as:

$$\boxed{Paon\text{–}S \coloneqq f_{[M/N]}(x) = \frac{Q_N(x)P_M(x) + Q_{N-1}(x)P_{M-1}(x)}{Q_N^2(x) + Q_{N-1}^2(x)}} \tag{4.5}$$

where $P_M(x)$ and $Q_N(x)$ are defined as in Equation (4.3). In this formulation, $Q_{N-1}(x)$ and $P_{M-1}(x)$ represent polynomials of one degree lower than $Q_N(x)$ and $P_M(x)$, respectively. For this variant, considering Equation (4.3), when the polynomials are of zero degree, the numerator simplifies to the bias term, $P_0(x) = w_0$, and for the denominator polynomial, $Q_0(x) = 1$, aligning the *Paon*s with aforementioned neurons when the higher order terms are absent. This variant ensures smoother behavior and stability by preventing the denominator from approaching zero, thus maintaining robust computational properties throughout the training process.

This modification inherently possesses stronger non-linearity. A closer examination of Equation (4.3) reveals that an $[M/N]$ Padé approximant agrees with a Taylor series expansion of order $M + N$, and requires $M + N$ "layer operations" in total, where layer operation refers to all operations required either for a convolutional layer when $\circledast$ is convolution or for a fully connected layer when $\circledast$ is multiplication. However, the approximation in Equation (4.5) corresponds to an $[(M+N)/2N]$ Padé approximant, which achieves the total polynomial order of $M + 3N$. Remarkably, this higher-order approximation is achieved with only $M + N$ "layer operations".

Consequently, this method allows us to achieve a higher degree of non-linearity compared to what is provided by Equation (4.3).

Paon introduces more non-linearity to the model by incorporating higher-order features into both the numerator and the denominator. Furthermore, due to its structure as a ratio of polynomials, *Paon* exhibits greater stability for higher-order approximations by limiting the summation of the positive integer powers of the input to go beyond the stable range through division. As a result, it does not require the output to be limited by bounded activation functions, such as tanh, thus avoiding issues such as the problem of vanishing gradients.

4.3 Shifter Module

In terms of non-linearity, *Paon* matches and surpasses its competitors. However, it still operates on the local neighborhood of a feature element to build up the corresponding one on the output activation map when the layer operates with convolution. To increase the receptive field of the neuron, we propose the Shifter module. The Shifter is designed to move the input features, thereby increasing the receptive field of the neuron. There are two different implementations for the Shifter module in *Paon*.

4.3.1 Shifter-I: Shifting via Small Network

In the first and original implementation, which was also used in the author's introductory paper [Keleş and Tekalp, 2024], the behavior of the Shifter module varies based on the shift parameter b:

- When the shift parameter is negative, the module is deactivated.
- When b is a positive integer, the module performs gradient-based optimization to find the optimal shifts within the range $[-b, b]$, just as the super neuron [Kıranyaz et al., 2024].
- When $b = 0$, the module computes the best shift for each channel without

restrictions. It consists of an averaging operation in channel dimension, a 1×1 convolution, and a parametric ReLU layer [He et al., 2015] as the non-linear activation function, to calculate a different shift for each input channel, along with some reshaping operation to maintain shape consistency.

- If the shifts are desired to be constant, then the sampling locations are initialized at the beginning and used throughout training and testing.

This shifting strategy respects the integrity of the kernel and is mainly in agreement with the one suggested by Kıranyaz et al. [Kıranyaz et al., 2024]. The shifting is either an integer, achieved by rounding the real number shifts, or a real number, in which case bilinear interpolation is used to sample features on non-integer grid points, which benefits from the same methodology used in spatial transformer networks [Jaderberg et al., 2015]. Moreover, it operates on either a limited range or the entire feature map. Our addition to this shifting strategy is that Shifter learns the amount of shift from the input data itself through a one-layer network; it manipulates the input features to extract the learned kernel offsets, whereas in Kıranyaz et al. [Kıranyaz et al., 2024] they are optimized by back-propagation and used for every test sample uniformly. Another difference is the padding strategy, which is necessary to define the behavior of resampling when the kernels go out of the feature borders. In that case, super neurons apply zero padding, meaning that out-of-boundary features are set to zero. On the other hand, our Shifter module pads the input via one of the border, reflection, or zero padding methods before bilinear interpolation. Following the work [Woods et al., 1985], we recommend and prefer to use border padding for Shifter, and when *Paon*s are used in the convolution kernel form, we suggest to pad the shifted input features via replication padding among the options being zero, reflection, replication, and circular padding, which are also available as implementation.

4.3.2 Shifter-II: Shifting via Deformable Kernels

Another strategy to achieve input feature shifts for the convolutional case is to use deformable kernels [Dai et al., 2017]. This method is more powerful because it can calculate the input feature shifts for each convolution kernel element individually. Deformable convolution allows the network to adaptively adjust the size and shape of the receptive field for each spatial location, leading to better modeling of complex geometric transformations and spatial dependencies in the data. However, excessive shifting in deformable convolution may cause the kernel to operate in regions inside or outside the input feature map, potentially leading to unstable training. Therefore, it is crucial to limit the amount of shift to ensure stability.

The amount of shifts (offsets) is learned through a single convolutional layer in this module. Also, as in the first variant, the behavior of this version of Shifter depends on the shift parameter b in terms of this limitation.

- When it is not positive, the limit for the maximum allowable shift m is $\max(h, w)/4$, where h and w are the height and width of the input feature map, respectively.

- For a positive integer b, $m = b$.

The limitation of the shift in deformable convolution is achieved using the tanh operation in the form of $m \cdot \tanh(x/m)$, where x is the offset map. In our opinion, the division of x by m might be crucial, although often overlooked in the literature [Chan et al., 2022], as it prevents small values from being pulled towards the upper limit. Without this division, the tanh function could start to saturate even for relatively small input values, leading to unintended shifts. Thus, this method ensures that the shifts remain within a controlled range, maintaining stable and effective training. By incorporating deformable convolution together with the aforementioned adjustments, the Shifter module can further enhance its ability to capture more useful features and improve the overall performance of the model when *Paon*s are used in the convolution kernels.

To ensure that the module only learns the amount of shift when it does not negatively impact performance, the convolution weights and bias in the Shifter module are initialized to zero. This initialization strategy allows the module to start in a neutral state, learning the necessary kernel offsets only when they improve the model performance.

For this shifting strategy, handling locations outside the original feature map is also crucial. The common practice in the community for both shifting the feature map and applying convolution is to pad the feature map with zeros. Although this approach is practical and often the default, it leads to the kernel operating in regions that contain no information. To address this issue, during both shifting and convolution, we pad the input using its border values, as in the first variant and also as suggested in [Woods et al., 1985]. This method ensures that the kernel always operates on more meaningful and nonzero data, even at the edges of the feature map.

Compared to the first alternative, this shifting method is superior. If necessary, each kernel element can be shifted and looks at the different locations on the feature map independent of each other. Moreover, the shifting offset can be found so that it also respects the integrity of the kernels. The only downside of this approach is that deforming the input features for each weight separately is efficiently implemented for regular convolution and, for instance, could be very expensive to use in transposed convolutions. Therefore, in such cases, the first shifting approach would be preferable, while in the other cases the deformable strategy should be chosen.

4.4 Paons as a Super Set of Proposed Neuron Models

An illustration of the proposed *Paon* with numerator degree 2 and denominator degree 3 is shown in Figure 4.1. First, the Shifter module, if necessary, shifts the input features so that each weight can operate at the optimum locations. Then the shifted features are subjected to exponentiation and then to the operation of the neuron. The final element-wise division applies either *Paon–A* approach in Equation (4.4) or *Paon–S* version with Equation (4.5).

As mentioned earlier, this newly proposed neuron model is called *Paon*. In

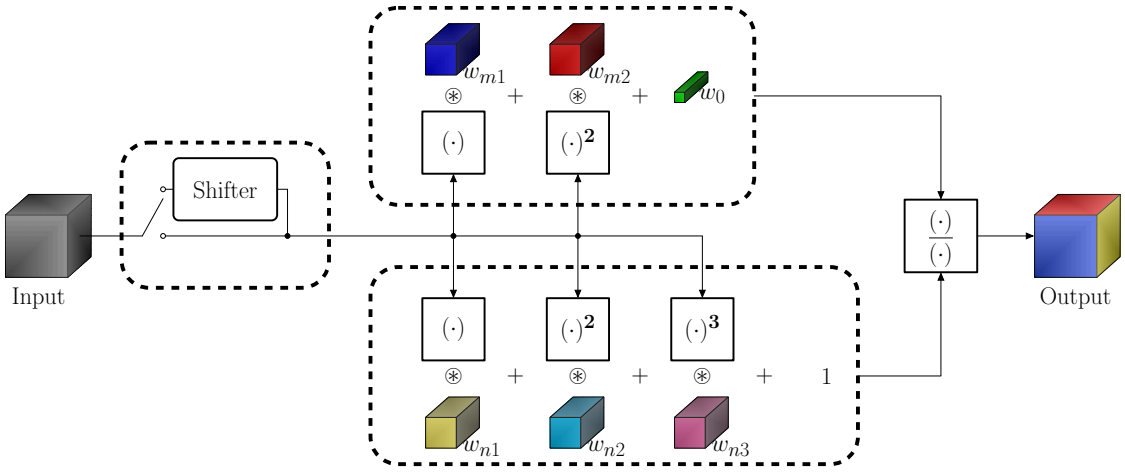


Figure 4.1: Illustration of a Padé approximant neuron (*Paon*) for $[M/N] = [2/3]$, where w_0 is bias for numerator, $(\cdot)^k$ takes k^{th} power of the input in element-wise manner, $\frac{(\cdot)}{(\cdot)}$ implements either *Paon–A* with Equation (4.4) or *Paon–S* with Equation (4.5). The Shifter module shifts the input features.

a network, a layer composed of Padé approximant neurons is called the Padé approximant layer, or *PaLa* in short. Finally, any network containing a Padé approximant layer is called the Padé approximant network or PadéNet for short. For the remainder of the thesis, these names will be used when necessary.

Since the Padé approximant is a better approximation compared to Taylor series expansion, which is also encapsulated by the Padé approximant, in terms of fidelity, stability, error reducing, and broader convergence [Baker and Graves-Morris, 1996], *Paon*s are a super set of the aforementioned neuron models, offering adaptability in various configurations:

- **Ordinary Neuron:** For $M = 1$, $N = 0$, and with the Shifter deactivated, the *Paon* reduces to an ordinary neuron. This is because, in the Padé approximant, when $N = 0$, the denominator becomes 1.

- **Generative Neuron:** For $M \geq 2$ and $N = 0$, the *Paon* behaves as a generative neuron, approximating complex functions through Taylor series expansion using higher-order terms.

- **Super Neuron:** When the Shifter module is activated in a generative neuron setting, the *Paon* operates as a super neuron, learning effective shifts from the data, thus improving performance through increased receptive fields.

- **Deformable Convolution:** The *Paon* also encapsulates deformable convolution capabilities by adapting to independent kernel shifts.

In a network containing *PaLa*s, *Paon*s are initialized so that only the bias term w_0 and the weights corresponding to $i = 1$ in Equation (4.3) are not 0, effectively starting *Paon*s from their simpler versions without any higher degrees and denominator terms as well as any shifting ability. This strategy ensures that Padé approximant neurons do not start from a suboptimal and/or a local minimum point at which they might be stuck during optimization, and learns the weights of other degrees and Shifter only if their contribution increases the performance for both finetuning and training from scratch. Given these properties, *Paon*s can seamlessly replace any neuron model in a neural network, providing robust and improved model performance. Moreover, this replacement is valid for all layers built by McCulloch-Pitts (ordinary) neurons, including common convolutions or dynamic convolutions [Jia et al., 2016, Yang et al., 2019], and fully connected layers in transformers [Vaswani et al., 2017] or vision transformers [Dosovitskiy et al., 2021].

Since *Paon*s are a super set of other neuron models, their training does not require any different trick that is not used in the literature; in other words, their training is as simple and straightforward as the training of other networks. As shown in the following chapter, the training of PadéNets can benefit from different optimizers, learning rate schedulers, loss functions, and so on. Moreover, this flexibility also helps PadéNets adopt different strategies such as dropout [Srivastava et al., 2014], weight normalization [Salimans and Kingma, 2016], and depth-wise separable kernels [Chollet, 2017] for the cases of convolutional layers.

The PyTorch [Paszke et al., 2019] implementation is open source and can be found at `https://github.com/onur-keles/Paon`.

4.5 Complexity Analysis of Paon and Shifter Types

*Paon*s have two different variants using Equation (4.4) and Equation (4.5), each of which can be used with Shifter-I or Shifter-II. Although the proposed neuron is a super set of previously known models, it is important to know the complexity of each separate choice. That being said, this analysis only focuses on the number of multiplication and division operations, as they are algorithmically more expensive compared to addition and subtraction, in a convolutional layer setting. Furthermore, we only consider the complexity of forward propagation as back-propagation is, presumably, implemented automatically and efficiently by the PyTorch [Paszke et al., 2019] framework.

The analysis is performed assuming that the input shape is $H \times W \times C_i$, where H is height, W is width, and C_i is the number of input channels. Furthermore, we assume odd kernel sizes k and k_s for convolution operations with stride 1 in *Paon* and Shifter, respectively.

Let C_o denote the number of *Paon*s in the current layer, all of which have $k \times k$ kernels. If the input is padded so that both the height and width of the input become $(H+2\cdot\lfloor k/2 \rfloor)$ and $(W+2\cdot\lfloor k/2 \rfloor)$, respectively, where $\lfloor\cdot\rfloor$ indicates the floor operation, the output shape becomes $H \times W \times C_o$. In that case, both *Paon* versions, *Paon–A* and *Paon–S*, with degrees $[M/N]$ perform $(M+N) \times C_i \times k \times k \times H \times W \times C_o$ multiplications to compute convolution and $H \times W \times C_o$ individual divisions for the calculation of the final ratio. In addition to these numbers, *Paon–S* has extra tensor multiplications for the calculations of $Q_N(x)P_M(x)$, $Q_{N-1}(x)P_{M-1}(x)$, $Q_N^2(x)$ and $Q_{N-1}^2(x)$, adding $4 \times H \times W \times C_o$ more operations.

The aim of the Shifter module is to slide the input features as a group or in an element-wise manner in horizontal or vertical directions so that the layer performing the convolution can extract more representative features compared to the case where the input is not shifted. That means that the dimensions of the output of Shifter should remain the same as the input; i.e., it must be $H \times W \times C_i$. Consequently, when shifting is necessary, there is a fixed number of operations $4 \times H \times W \times C_i$ as

the algorithmic cost, as Shifter performing bilinear interpolation should calculate a weighted sum of 4 feature "pixels" for each feature element.

For the first Shifter type, if the shift parameter b is negative, there is no operation on the resampling of the input. Otherwise, if the shifts are to be learned from the input data, then a convolutional layer calculates the amount of shift in the x and y directions for each channel. This means that particular layer has $2C_i \times k_s \times k_s \times C_i$ parameters. To obtain the shifts as a tensor with dimensions $H \times W \times 2C_i$, the input is padded so that the height and width become $(H + 2 \cdot \lfloor k_s/2 \rfloor)$ and $(W + 2 \cdot \lfloor k_s/2 \rfloor)$, respectively. In these circumstances, the total number of multiplications required to perform convolution is $2C_i \times k_s \times k_s \times H \times W \times C_i$.

For Shifter-II, we have to find an offset for each element of a $k \times k$ convolution operation kernel. Thus, the layer that extracts these offsets must have $2k^2 \times k_s \times k_s \times C_i$ parameters that operate on the padded input feature vector. This setup requires $2k^2 \times k_s \times k_s \times H \times W \times C_i$ multiplications for convolution, where k^2 occurs as the number of elements in a $k \times k$ kernel, and 2 is for the x and y directions. Here, please note that every kernel shares the calculated offsets. If we want to deform the features so that every input channel has its own unique offset values, then $2k^2$ becomes $2k^2C_i$.

Although the number of operations and parameters can be deduced from the analysis above, it would be more complete and understandable to give numerical examples for the complexity of implemented layers. The number of multiply-accumulation operations (MAC) [Ludgate, 1909], where each multiplication followed by a summation is counted as one operation, and/or floating point operations per second (FLOPS) [Kuck, 1974], in which each addition, subtraction, multiplication or division operation of two floating point numbers is counted as one operation, active cache and allocated memory in terms of mebibytes (MiB), and processing times in the central processing unit (CPU) and the graphics processing unit (GPU) are considered. For this pur-

Table 4.1: The number of MACs and FLOPS reported by different libraries. For convolutional and transposed convolutional versions of *PaLa*s, the conditions in the parantheses indicate the Shifter-I modes. Each number is the required MACs for the corresponding layer, except for `flop_counter` which gives the number of FLOPS.

Layer \ Library	`fvcore`	`ptflops`	`thop`	`torchprofile`	`flop_counter`
Fully Connected (Linear)	16,777,216	16,781,312	16,777,216	16,777,216	33,554,432
*PaLa*Linear	33,550,000	33,554,432	0	4,096	67,108,864
Convolution	14,745,600	14,942,208	14,745,600	14,745,600	29,491,200
*PaLa*Conv($b < 0$)	29,491,200	29,491,200	29,491,200	29,687,808	58,982,400
*PaLa*Conv($b = 0$)	30,277,650	29,491,236/29,491,224	29,491,224	29,687,829	58,982,436
*PaLa*Conv($b > 0$)	30,277,632	29,491,200	29,491,200	29,687,808	58,982,400
Transposed Convolution	14,745,600	59,768,832/15,532,032	58,982,400	58,982,400	29,491,200
*PaLa*TransposedConv($b < 0$)	29,491,200	117,964,800/29,491,200	117,964,800	118,751,232	58,982,400
*PaLa*TransposedConv($b = 0$)	30,277,650	117,964,836/29,491,224	117,964,824	118,751,253	58,982,436
*PaLa*TransposedConv($b > 0$)	30,277,632	117,964,800/29,491,200	117,964,800	118,751,232	58,982,400
Deformable Convolution	9,830,400	13,107,200	9,830,400	9,830,400	19,660,800
*PaLa*DeformableConv	9,830,400	121,241,600/13,107,200	9,830,400	13,303,808	19,660,800

pose, four different libraries, namely `fvcore`[1], `ptflops`[2], `thop`[3], and `torchprofile`[4] together with native PyTorch methods such as `torch.utils.flop_counter` and `torch.profiler` are used. For linear layers, the assumed input is $x_l \in \mathbb{R}^{1\times 4096}$, and for common, transposed, and deformable convolutions, it is $x_c \in \mathbb{R}^{1\times 3\times 256\times 256}$. Each layer gives the output with the same shape as the input, except for the transposed convolution, which doubles the height and width of the input. For convolutions, the kernel sizes are 5×5, and for each *PaLa*, the degrees are [1/1] with *Paon–S*. The codes for calculating the number of operations and the memory complexity are given in Appendix D.

The numbers of operations detected by different libraries are given in Table 4.1.

[1] `https://github.com/facebookresearch/fvcore`, accessed: February, 2025.

[2] `https://github.com/sovrasov/flops-counter.pytorch`, accessed: February, 2025.

[3] `https://github.com/Lyken17/pytorch-OpCounter`, accessed: February, 2025.

[4] `https://github.com/zhijian-liu/torchprofile`, accessed: February, 2025.

First, please note that all libraries give the number of MACs, except for the native PyTorch method `torch.utils.flop_counter`, depicted as `flop_counter`, which reports the number of FLOPS. Interestingly, the results in Table 4.1 indicate that there are some inconsistencies both among the libraries and within them for each layer type.

- **Fully Connected:** `fvcore`, `thop` and `torchprofile` libraries give $4096 \cdot 4096$ multiplication and addition for the common linear layer, disregarding the bias since its operation does not include multiplication. `ptflops` seems to take the bias addition into account, but that operation is not a "MAC". Although `flop_counter` claims to count the number of FLOPS, it only seems to consider $4096 \cdot 4096$ multiplication and accumulation, again not considering the addition of bias term. For *PaLa* version, `thop` and `torchprofile` give counterintuitive results, most probably because those libraries take only specific modules and functions into account and the linear version of *PaLa* is implemented without the native PyTorch layer. `flop_counter` method gives a result exactly twice of the one it gives for the common linear layer, which indicates that it ignores not only the addition of the bias but also the operations needed for *Paon–S*. In fact, the multiplication, addition and division of polynomials required for *Paon–S*, even the addition of 1 to the denominator polynomial, seem not to be considered at all by any library.

- **Convolution:** For the common implementation, only `ptflops` library considers the bias addition to each output pixel. However, for *PaLa* without a Shifter ($b < 0$), `torchprofile` seems to add the bias term operations to the number of MACs reported. When Shifter that learns the optimal shifts with only back-propagation ($b > 0$) is included, all but the `fvcore` library ignore the bilinear interpolation required to shift the input features. For the case where $b = 0$ for Shifter, `fvcore` and `flop_counter` seem to consider the layer that learns the amount of shifts. `ptflops` gives two different MAC values, the first of which is for the PyTorch backend and the second for the ATen library

that uses the C++ framework in PyTorch.

- **Transposed Convolution:** Since transposed convolution can be considered as fractionally-strided convolution, it is logical to see bigger numbers for MAC and FLOPS requirements. However, `fvcore` and `flop_counter` give the exact same values as they give for convolutional layers. In case of the *PaLa* version for transposed convolution, `ptflops` library reports more reasonable results for the PyTorch backend, but gives a smaller number for the ATen backend.

- **Deformable Convolution:** Without any offset to deform the kernels and offset calculating layer, the deformable convolution should be at least equal to the common convolutional layer in terms of the required operations. However, none of the libraries gives results parallel to this expectation; in fact, the numbers indicate that the deformable convolution is cheaper than the common convolution. Considering an additional layer to find the offset values and bilinear interpolation of the input feature for each kernel element separately, the deformable convolution should cost more than the common convolution. This result can only be seen in PyTorch backend results in case of *PaLa*DeformableConv.

The number of parameters, maximum active and maximum active cached memory, and required CPU and GPU total times for the layers given in Table 4.1 are presented in Table 4.2. Memory requirements are measured by native PyTorch functionality, `torch.cuda.memory._record_memory_history()`, and time is measured via another native method, `torch.profiler.profile`, on a computer with Intel Core i7-10750H @ 2.60GHz CPU and NVIDIA Quadro RTX 3000 GPU. As expected, *PaLa* versions of all layers have a higher number of parameters, memory requirements, and forward times. Although the number of parameters for a *PaLa* version of a common layer with degrees [1/1] is close to double of its simpler version, the memory requirements do not seem so. In contrast, the total times in both CPU and GPU seem to be higher than expected. The most basic explanation for these results is that *Paon* is implemented

Table 4.2: Memory and time complexity.

Layer \ Criterion	Parameters	Memory	Time
Fully Connected (Linear)	$16,781,312$	73.2 MiB / 86.0 MiB	32μs / 212μs
*PaLa*Linear	$33,558,528$	200.2 MiB / 200.3 MiB	160μs / 549μs
Convolution	228	201.7 MiB / 214.0 MiB	51μs / 88μs
*PaLa*Conv($b < 0$)	453	208.5 MiB / 224.0 MiB	1.087ms / 253μs
*PaLa*Conv($b = 0$)	483	212.2 MiB / 226.0 MiB	1.106ms / 510μs
*PaLa*Conv($b > 0$)	459	212.2 MiB / 226.0 MiB	1.544ms / 592μs
Transposed Convolution	228	203.9 MiB / 214.0 MiB	723μs / 606μs
*PaLa*TransposedConv($b < 0$)	453	226.4 MiB / 236.0 MiB	1.296ms / 1.345ms
*PaLa*TransposedConv($b = 0$)	483	228.7 MiB / 238.0 MiB	1.144ms / 1.645ms
*PaLa*TransposedConv($b > 0$)	459	228.7 MiB / 238.0 MiB	1.715ms / 1.940ms
Deformable Convolution	428	341.4 MiB / 358.0 MiB	2.129ms / 740μs
*PaLa*DeformableConv	653	342.2 MiB / 368.0 MiB	2.539ms / 2.795ms

in the PyTorch framework without using any GPU programming, causing some inefficient implementations. For instance, in the case of deformable convolution, the *PaLa* version is implemented via torchvision's native class `DeformConv2d`. Although the offsets are calculated once, the input has to be deformed twice, one for the numerator polynomial and one for the denominator polynomial. Such inefficiencies can be prevented by using GPU-friendly frameworks, which is left as a long-term future work.

Chapter 5

APPLICATIONS OF PAONS TO IMAGE SUPER-RESOLUTION AND COMPRESSION

This chapter presents experimental evidence for the superiority of *Paon*s in convolutional networks. In the first part, experiments and results are shown in the single image super-resolution problem. This section has results from the initial *Paon* settings, which was also introduced in the ICIP paper by Keleş and Tekalp [Keleş and Tekalp, 2024], and results from its enhanced version. In the second part, *Paon*s are employed in two different image compression frameworks, in one of which the number of layers is kept constant and replaced by *PaLa*s while in the second the number of layers is also reduced after replacement.

5.1 Single Image Super-Resolution

As a machine learning task, single image super-resolution is an ill-posed problem in which the aim is to obtain a high-resolution image from its low-resolution counterpart by recovering and retaining as much detail as possible. In this section, *Paon*s with two versions of Shifter module are tested against their alternative neuron models in a simple setting for the single image super-resolution problem. The first subsection covers the experiments with the initial proposal of *Paon* while the next subsection presents the experiments with *Paon*s equipped with the enhanced Shifter module.

5.1.1 Single Image Super-Resolution with Original Shifter Module

In the 2024 ICIP paper where *Paon*s are first introduced [Keleş and Tekalp, 2024], we show the performance of PadéNets with shifting via small network on the single-image super-resolution problem. The basic architecture chosen for this task is a

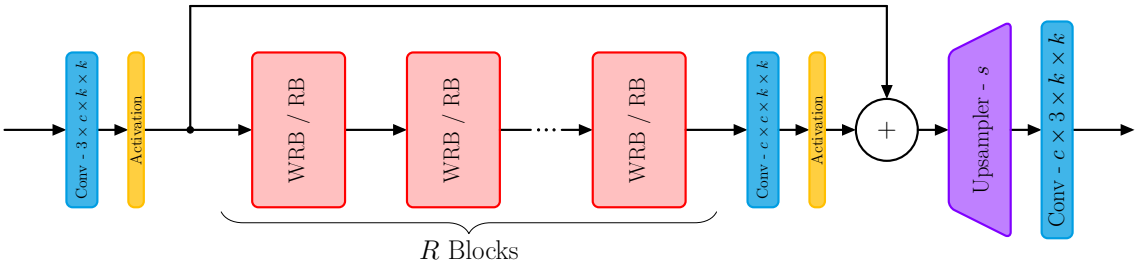


Figure 5.1: The architecture for the super-resolution experiments. Directly taken from [Keleş and Tekalp, 2024].

widely used one since the seminal paper by Ledig et al. [Ledig et al., 2017]. In this architecture, a single convolutional layer extracts features from the image. Then it is followed by a series of blocks for residual feature refinement. The initial features are added back to the refined residual features. The sum is processed by a feature upsampler module, which contains a convolutional layer, an activation, and a pixel shuffler layer [Shi et al., 2016]. The employed architecture is shown in Figure 5.1.

For their simplicity, we chose residual blocks [He et al., 2016] and wide residual blocks [Zagoruyko and Komodakis, 2016] with scaled residuals [Szegedy et al., 2017] as our feature refinement blocks. For a fair comparison in terms of the number of parameters, we use wide residual blocks in the models composed of common neurons. For the layers operating on not only the input features but also its positive integer powers, we use residual blocks. The learnable scalar layer for each output channel is initialized from 0.1. Figure 5.2 shows the structure of a residual and wide residual block in which case w is bigger than 1.

We check the performance of PadéNets by comparing various architectures: a wide residual network composed of convolutional layers and Gaussian error linear unit (GELU) activation [Hendrycks and Gimpel, 2016] (called ResNet), a network with convolutional layer and PAU activation (called PAU-Net), a Self-ONN, a SuperONN, and PadéNet. In all models, we keep the convolutions in the initial feature extractor, at the end of feature refinement, and the final image constructor part (upsampler and final layer) the same degrees as [1/0] to be able to keep track of the number

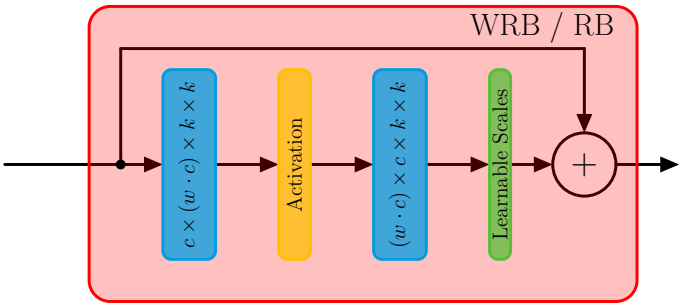


Figure 5.2: Normal and wide residual block structure. For wide residual block (WRB), w is bigger than 1 while for normal residual block (RB), it is 1. Directly taken from [Keleş and Tekalp, 2024].

Table 5.1: PSNR scores on DIV2K ×2 validation set. "FL", "LL" and "AL" denote first layer, last layer and all layers are *PaLa*, respectively. All values are calculated using *Paon–S*, except for the *Paon–A* column. Better results are indicated with **bold** font. Directly taken from [Keleş and Tekalp, 2024].

No Shift	Shift	*Paon–A*	*Paon–S*	FL	LL	AL
34.32	**34.36**	34.35	**34.36**	34.29	34.30	**34.36**

of parameters. For more information on GELU, please see Appendix A.3 or its introductory work [Hendrycks and Gimpel, 2016].

Before the final comparison, we investigate the performance of PadéNet in different setups. We check which *Paon* is better, whether or not Shifter is helpful, and which layers of RB to be converted to *PaLa*. The results are shown in Table 5.1. Based on these metrics, we continue with *Paon-S* with Shifter activated in both layers of the residual block.

For the final comparison, the architecture details are given in Table 5.2. Note that Self-ONN and SuperONN have tanh activations. In experiments with GELU, these two networks became unstable, so we used tanh in residual blocks as proposed in their studies.

For training, we use the DF2K dataset [Lim et al., 2017] as it has more images

Table 5.2: Model configurations. Degrees denote the degree of numerator/denominator polynomials $[M/N]$. The degree for PAU-Net is the degree of the PAU activation. RB and WRB denote residual block and wide residual block, respectively. Directly taken from [Keleş and Tekalp, 2024].

Attribute \ Model	ResNet	PAU-Net	Self-ONN/SuperONN	PadéNet
Degrees	[1/0]	[7/6]	[3/0]	[2/1]
Activation	GELU	PAU	tanh	GELU
Block Type (w)	WRB (3)	WRB (3)	RB (1)	RB (1)
Blocks, R	3	3	3	3
Channels	48	48	48	48

compared to DIV2K [Agustsson and Timofte, 2017]. The models are trained on 64×64 patches scaled to the $[-1, 1]$ range with 25 batch size for 500 000 iterations to perform on $\times 2$ and $\times 4$ super-resolution. The data are augmented with random rotation, horizontal and vertical flip, and color channel shuffling. In addition, in the experiments, we observed that adding a small amount of Gaussian noise during training improves the validation score of the network. Therefore, we add Gaussian noise with signal-to-noise ratio (SNR) 40 dB to the cropped patches. At each iteration, the models minimize the loss function proposed by Barron [Barron, 2019], with $\alpha = 1.5$ and $c = 2$. We use the Adan optimizer [Xie et al., 2024] with 10^{-3} as the initial learning rate and the cosine annealing scheduler [Loshchilov and Hutter, 2017] until the learning rate becomes 10^{-6} at the end of the training. The best models are saved with respect to their validation PSNR in the DIV2K validation set. Those training settings are all common for all the models compared. For more information on the loss function, please see Appendix C.2.

For comparison, the standard sets in super-resolution that are BSD100 [Martin et al., 2001], Manga109 [Matsui et al., 2017], Set5 [Bevilacqua et al., 2012], Set14 [Zeyde et al., 2012], and Urban100 [Huang et al., 2015] are used. For all compared

models, PSNR, SSIM and LPIPS metrics are reported to compare performance. PSNR is calculated from RGB images [Keleş et al., 2021b]. For SSIM [Wang et al., 2004], Y channel of YCbCR image is used. LPIPS [Zhang et al., 2018] results are reported for both AlexNet [Krizhevsky et al., 2012] and VGG [Simonyan and Zisserman, 2015] as the feature extractor networks.

The quantitative results are shown in Table 5.3. It can be seen that PadéNet surpasses all the models in every fidelity metric, PSNR and SSIM, for nearly all the datasets. The PSNR difference with wide ResNet can be as much as 0.15 dB. This indicates that although the number of parameters is close to each other, a model equipped with function approximation capability can serve better in the image reconstruction. Moreover, comparison with PAU-Net shows that increasing the approximation capability of the network by introducing the Padé approximants to the input features instead of the activations only increases the expressive capability of the models. Finally, using Padé approximant rather than Taylor series expansion in the neurons proves to be a better strategy, thanks to the more accurate approximation capability of former over the latter. This shows that the networks built with *Paon* can take advantage of the most input information and express the desired function that a neural network tries to approximate better.

Figure 5.3 and Figure 5.4 show the qualitative results of the models. Since the $\times 2$ super-resolution is a relatively easier problem than the $\times 4$ super-resolution, the nuances are harder to detect; however, it is still possible to find evidence for PadéNet's superiority. In Figure 5.3, it can be seen that the cropped region has some high frequency content. Inspecting the images, it can be claimed that the most successful model to reconstruct these details is PadéNet. The other models either cause aliasing or fail to reconstruct a straight line (in the case of Self-ONN). The difference is clearer in Figure 5.4, where the images are the results of the models performing $\times 4$ super-resolution. The table cover normally has square patterns that all models fail to reconstruct. But PadéNet is again the most successful method in reconstructing the perpendicular lines, indicating the better image reconstruction and detail-capturing capabilities of PadéNets.

Table 5.3: Quantitative comparison. The top two scores in each cell are PSNR(↑) and SSIM(↑), and the bottom two are LPIPS(↓) calculated via AlexNet and VGG, respectively. The best and second best scores for each dataset are shown in **blue** and red, respectively. Directly taken and rearranged from [Keleş and Tekalp, 2024].

Scale	Model \ Dataset	BSD100	Manga109	Set5	Set14	Urban100
×2	ResNet	30.64/0.8903 0.1513/0.1580	35.60/0.9724 0.0242/0.0568	35.50/0.9548 0.0591/0.0964	30.99/0.9074 0.0985/0.1462	29.76/0.9148 0.0763/0.1198
	PAU-Net	30.63/0.8903 0.1516/0.1576	35.60/0.9723 0.0241/**0.0561**	35.51/0.9548 0.0591/**0.0955**	30.97/0.9073 0.0994/0.1457	29.75/0.9147 0.0767/0.1198
	Self-ONN	30.65/0.8904 0.1508/0.1584	35.62/0.9724 0.0235/0.0566	**35.57**/0.9550 0.0585/0.0957	31.11/0.9074 0.0985/0.1466	29.82/0.9153 0.0757/0.1193
	SuperONN	30.65/0.8905 **0.1501**/0.1578	35.63/0.9724 0.0236/0.0567	35.54/0.9549 0.0582/0.0964	31.12/0.9073 0.0984/0.1462	29.80/0.9153 0.0750/0.1191
	PadéNet	**30.68**/**0.8909** 0.1516/**0.1574**	**35.70**/**0.9727** **0.0234**/0.0569	35.54/**0.9551** **0.0580**/0.0958	**31.14**/**0.9080** **0.0981**/**0.1456**	**29.90**/**0.9165** **0.0742**/**0.1183**
×4	ResNet	26.13/0.7133 **0.3877**/0.3434	28.16/0.8924 0.1166/0.1720	29.78/0.8775 0.1808/0.2184	26.29/0.7594 **0.2947**/0.3090	24.27/0.7598 0.2540/0.2944
	PAU-Net	26.11/0.7127 0.3892/0.3434	28.11/0.8912 0.1172/0.1696	29.81/0.8771 0.1807/0.2172	26.23/0.7594 0.2952/0.3080	24.23/0.7582 0.2559/0.2946
	Self-ONN	26.12/0.7125 0.3892/0.3432	28.14/0.8908 0.1168/**0.1685**	29.85/0.8774 **0.1799**/**0.2156**	26.32/0.7591 0.2951/0.3085	24.26/0.7585 0.2559/**0.2940**
	SuperONN	26.13/0.7131 0.3893/0.3444	28.16/0.8922 **0.1156**/0.1728	29.88/0.8776 0.1812/0.2185	26.35/0.7598 0.2956/0.3092	24.30/0.7608 0.2517/0.2942
	PadéNet	**26.16**/**0.7138** 0.3884/0.3439	**28.28**/**0.8941** 0.1161/0.1726	**29.90**/**0.8788** 0.1812/0.2192	**26.37**/**0.7604** 0.2958/**0.3078**	**24.35**/**0.7624** **0.2515**/**0.2940**

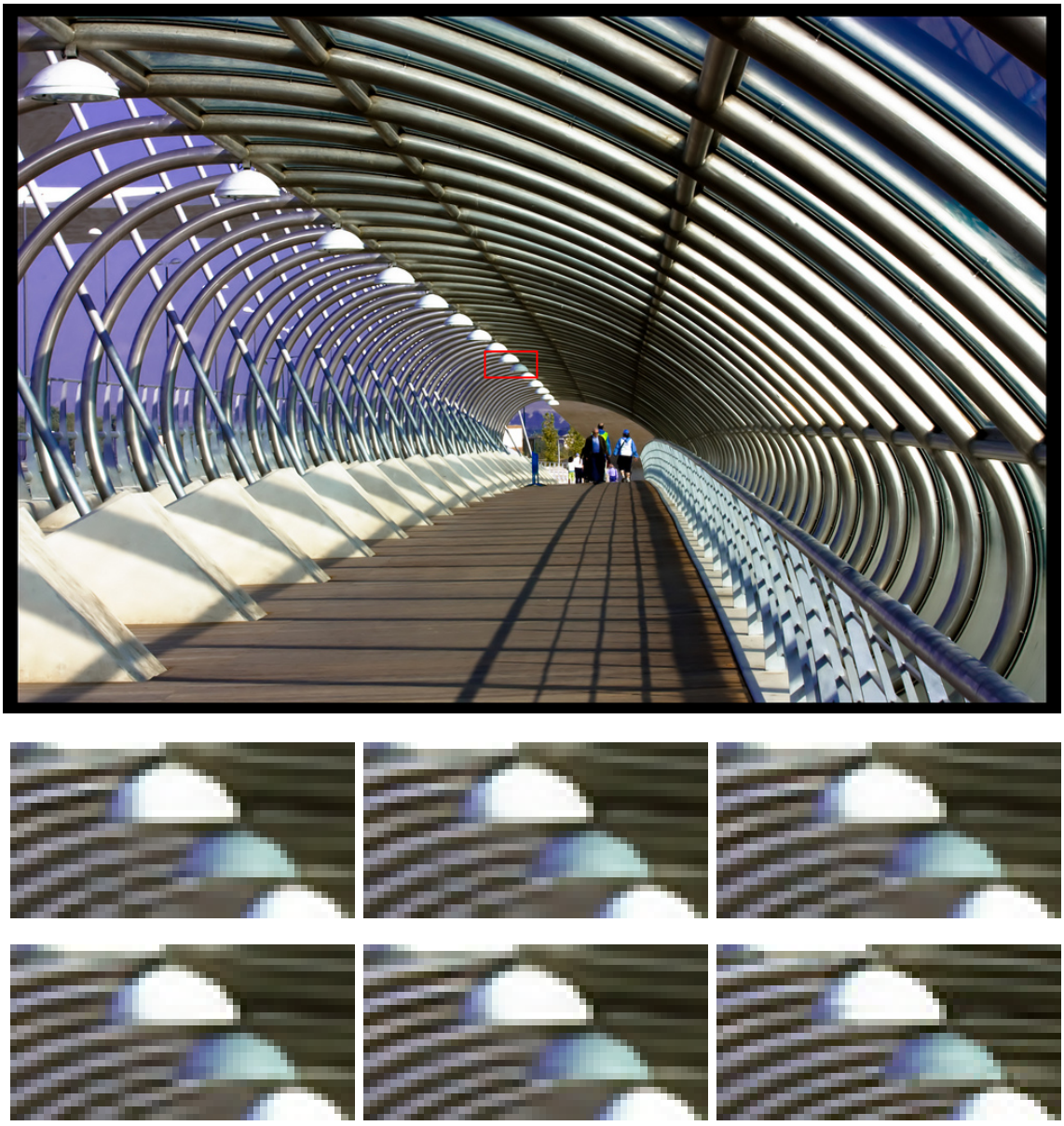


Figure 5.3: Visual comparison for $\times 2$ SR on `img_058.png` from Urban100 dataset. Crops from left to right are outputs of ResNet, PAU-Net, Self-ONN, SuperONN, PadéNet and ground truth. Directly taken from [Keleş and Tekalp, 2024].

Figure 5.4: Visual comparison for $\times 4$ SR on `barbara.png` from Set14 dataset. Crops from left to right are outputs of ResNet, PAU-Net, Self-ONN, SuperONN, PadéNet and ground truth. Directly taken from [Keleş and Tekalp, 2024].

5.1.2 Single Image Super-Resolution with Enhanced Shifter Module

In these experiments, we show the performance of PadéNets using the input shifting via deformable convolution. The training settings are exactly the same as in Section 5.1.1 and the ICIP paper [Keleş and Tekalp, 2024]. The architecture is also the same as in Figure 5.1 using the residual block in Figure 5.2 employing *Paon-S* in Equation (4.5). However, since the current work extends and improves the previous work [Keleş and Tekalp, 2024], we need to address some design choices made in the earlier study. One of the key considerations is to determine whether the order [2/1] is necessary or if it can be simplified in favor of a less complex model. To investigate this, we evaluate different orders, including [1/1], [2/0] and [2/1] using the same Shifter setting as in Keleş and Tekalp [Keleş and Tekalp, 2024].

The results of this evaluation are presented in Table 5.4. They show that although the best model is with degrees [2/1], the second best model with [1/1] demonstrates very similar performance while having more than 25% fewer parameters. The table also indicates that, despite having the same number of parameters, the degree [1/1] outperforms the degree [2/0]. We hypothesize that this difference in performance is due to the effective degree of approximation achieved by Equation (4.5). For the [1/1] configuration, the effective degree is 4, while for [2/0], it remains 2. Consequently, we choose to proceed with order [1/1] for further experiments and evaluations.

Table 5.4: The Padé degree experiments. On the top row, RGB PSNR and Y channel SSIM are reported on DIV2K validation dataset. The bottom row is the approximate number of parameters.

$[M/N]$ / Criteria	[1/1]	[2/0]	[2/1]
PSNR / SSIM	28.82/0.8179	28.79/0.8170	28.85/0.8187
Number of parameters	$\approx$ 469K	$\approx$ 469K	$\approx$ 593K

We also have to consider the Shifter type to be used. The previous study used kernel-wise shifts, whereas in this study, we introduce a deformable strategy that

Table 5.5: Full deformable convolution kernels and offset kernel experiments with degrees [1/1]. On the top row, RGB PSNR and Y channel SSIM are reported on DIV2K validation dataset. The bottom row is the approximate number of parameters.

Full Deformable \ Offset Kernel	1×1	3×3
False	28.81/0.8177 $\approx$ 441K	28.80/0.8173 $\approx$ 445K
True	28.84/0.8182 $\approx$ 445K	28.83/0.8179 $\approx$ 487K

can be used for both element-wise and kernel-wise shifts via offsets applied to the input features. Here, the offset values are determined through convolutional layers. Furthermore, to keep the parameter count as low as possible, we also evaluate the performance of Shifter with the offset calculation kernels 1×1 and 3×3.

The results are presented in Table 5.5. Here, "full deformable" indicates whether the offsets are applied kernel-wise (False) or element-wise (True). The results show that when the shifts are applied element-wise, the performance is better, as expected. Moreover, it indicates that the use of a 1×1 offset convolution does not affect the overall performance. Finally, according to Table 5.5, the usage of deformable convolution with a 1×1 offset kernel decreases the number of parameters compared to the previous shifting strategy.

As a result, we have chosen to proceed with the degrees [1/1] and deformable shifting with a 1×1 offset kernel. This configuration balances performance and parameter efficiency, ensuring that the model remains both effective and computationally feasible.

For comparison, we again use standard datasets in super-resolution, which include BSD100 [Martin et al., 2001], Manga109 [Matsui et al., 2017], Set5 [Bevilacqua et al., 2012], Set14 [Zeyde et al., 2012], and Urban100 [Huang et al., 2015], as well as the DIV2K validation dataset for ablation. The performance of all compared models

Table 5.6: Model configurations. Degrees denote the degree of numerator/denominator polynomials $[M/N]$. The degree for PAU-Net is the degree of the PAU activation. "RB" and "WRB" denote residual block and wide residual block, respectively. "Deform." means deformable, and "KW" is kernel-wise shift.

Attribute \ Model	ResNet	DCN $k_s \times k_s$	PAU-Net	Self-ONN	SuperONN	PadéNet
Degrees	[1/0]	[1/0]	[7/6]	[2/0]	[2/0]	[1/1]
Activation	GELU	GELU	PAU	GELU	GELU	GELU
Blocks, R	3	3	3	3	3	3
Type (w)	WRB (2)	WRB (2)	WRB (2)	RB (1)	RB (1)	RB (1)
Channels	48	48	48	48	48	48
Strategy	–	Deform.	–	–	KW	Deform.
Shift Kernel	–	$k_s \times k_s$	–	–	–	1×1

is evaluated using PSNR, SSIM, and LPIPS metrics. PSNR [Keleş et al., 2021b] is calculated from RGB images. SSIM [Wang et al., 2004] is calculated using the Y channel of the YCbCR image. LPIPS [Zhang et al., 2018] results are reported from both the AlexNet [Krizhevsky et al., 2012] and VGG [Simonyan and Zisserman, 2015] networks.

We evaluate the performance of PadéNets by comparing various architectures, including the wide residual network (ResNet), which is composed of convolutional layers with GELU activation, PAU-Net with convolutional layers and PAU activation, deformable convolution networks using GELU non-linearity (DCN $k_s \times k_s$) having various offset convolution kernel sizes, a Self-ONN, a SuperONN, and PadéNet. In all models, we maintain the same degrees of convolution in the initial feature extractor, the feature refinement endpoint, and the final image constructor (including the upsampler and final layer) as [1/0]. Only feature refinement parts containing residual blocks are modified to ensure a similar number of parameters across different architectures. The architecture details for all the models compared are presented in Table 5.6.

Table 5.7: Quantitative comparison. The top two scores in each cell are PSNR(↑) and SSIM(↑), and the bottom two are LPIPS(↓) calculated via AlexNet and VGGNet, respectively. Under the names of each model is the approximate number of parameters. The best and second best scores for each dataset are shown in **blue** and red, respectively.

Scale	Model \ Dataset	BSD100	Manga109	Set5	Set14	Urban100
×2	ResNet	30.61/0.8899	35.52/0.9722	35.44/0.9546	30.93/0.9071	29.68/0.9137
	≈ 357K	0.1521/0.1587	0.0243/0.0570	0.0592/0.0973	0.0992/0.1469	0.0781/0.1210
	DCN 1 × 1	30.66/0.8904	35.56/0.9724	35.53/0.9548	31.11/**0.9076**	29.82/0.9153
	≈ 363K	0.1507/0.1579	0.0236/0.0574	0.0584/0.0965	0.0978/0.1454	0.0753/0.1196
	DCN 3 × 3	30.65/0.8903	35.61/0.9725	35.54/**0.9549**	31.13/**0.9076**	29.80/0.9148
	≈ 426K	0.1513/0.1583	0.0238/0.0569	0.0583/0.0962	0.0983/0.1457	0.0766/0.1200
	PAU-Net	30.59/0.8897	35.48/0.9720	35.42/0.9544	30.92/0.9068	29.60/0.9128
	≈ 357K	0.1514/0.1589	0.0243/0.0571	0.0596/0.0977	0.0996/0.1479	0.0794/0.1221
	Self-ONN	30.62/0.8899	35.49/0.9721	35.51/0.9547	31.06/0.9069	29.69/0.9134
	≈ 357K	0.1514/0.1592	0.0239/0.0572	0.0583/**0.0960**	0.0990/0.1468	0.0780/0.1207
	SuperONN	30.62/0.8900	35.49/0.9721	35.48/0.9546	31.04/0.9069	29.69/0.9137
	≈ 357K	0.1508/0.1586	0.0239/**0.0568**	0.0590/0.0969	0.0994/0.1470	0.0771/0.1206
	PadéNet	**30.68**/**0.8909**	**35.69**/**0.9726**	**35.55**/**0.9549**	**31.14**/**0.9076**	**29.93**/**0.9168**
	≈ 362K	**0.1500**/**0.1571**	**0.0233**/0.0570	**0.0580**/0.0968	**0.0965**/**0.1449**	**0.0723**/**0.1186**
×4	ResNet	26.10/0.7123	28.02/0.8904	29.76/0.8768	26.21/0.7585	24.19/0.7567
	≈ 440K	0.3892/0.3447	0.1186/0.1739	0.1815/0.2190	0.2956/0.3105	0.2572/0.2978
	DCN 1 × 1	26.14/0.7133	28.16/0.8923	29.90/0.8781	26.34/0.7595	24.30/0.7604
	≈ 447K	0.3870/**0.3434**	0.1160/0.1719	0.1803/0.2171	0.2950/**0.3072**	0.2522/0.2942
	DCN 3 × 3	26.14/0.7132	28.13/0.8917	29.87/0.8779	26.33/0.7593	24.27/0.7594
	≈ 509K	**0.3864**/0.3435	0.1162/0.1724	0.1802/0.2174	0.2950/0.3083	0.2535/0.2952
	PAU-Net	26.08/0.7116	27.98/0.8894	29.73/0.8758	26.20/0.7576	24.15/0.7551
	≈ 440K	0.3908/0.3435	0.1188/**0.1711**	0.1805/0.2167	0.2971/0.3091	0.2600/0.2977
	Self-ONN	26.11/0.7123	28.08/0.8908	29.80/0.8770	26.33/0.7586	24.22/0.7573
	≈ 440K	0.3896/0.3452	0.1178/0.1717	0.1804/0.2181	0.2964/0.3090	0.2567/0.2956
	SuperONN	26.11/0.7122	28.05/0.8902	29.81/0.8768	26.30/0.7583	24.22/0.7571
	≈ 440K	0.3892/0.3449	0.1174/0.1720	0.1808/0.2190	0.2962/0.3096	0.2562/0.2956
	PadéNet	**26.15**/**0.7139**	**28.25**/**0.8935**	**29.96**/**0.8792**	**26.37**/**0.7604**	**24.35**/**0.7622**
	≈ 445K	0.3868/**0.3434**	**0.1146**/0.1717	**0.1797**/**0.2163**	**0.2943**/0.3073	**0.2503**/**0.2933**

Quantitative results, including fidelity metrics (PSNR and SSIM) and perceptual metrics (LPIPS), are shown in Table 5.7. These results demonstrate that PadéNet consistently achieves the best fidelity performance across all datasets and scaling factors. Furthermore, PadéNet generally shows superior perceptual metric performance for both scaling factors. Comparison with Self-ONN and SuperONN clearly demonstrates the superiority of *Paon*s in terms of the mathematically accurate function approximation ability. Moreover, compared to SuperONN, the Shifter module proves to be a more effective strategy for feature shift. Table 5.7 also validates the design choice for the offset calculation kernel 1×1. Even within deformable convolution networks, the 1×1 offset kernel performs closer to or better than the 3×3 offset calculation kernel.

The qualitative results are presented in Figure 5.5. These results clearly indicate that *Paon* exhibits superior performance compared to its competitors in terms of fidelity. For example, the high-frequency patch on the top image (`img_024.png`) is reconstructed best by PadéNet, nearly without aliasing, while other methods introduce aliasing artifacts. This superior performance is also observed in other images. In the middle image (`img_073.png`), the shown crop has the least amount of aliasing artifacts in the output of PadéNet. For the bottom image (`img_076.png`), the building stripes are mostly correctly oriented in the output of PadéNet, demonstrating the effectiveness of *Paon*s in maintaining structural integrity. These qualitative results confirm the quantitative findings, demonstrating that PadéNet offers superior performance in preserving high-frequency details and structures in the image. This effectiveness is attributed to the superior approximation capabilities of the Padé approximant employed in *Paon*.

Motivated by these results, we also decide to conduct further experiments. To see whether *Paon*s are also effective and competitive in both perceptual and fidelity metrics *when they do not receive help from an outside activation function*, we train a PadéNet configured as in Table 5.6 *without* GELU activation for each of the target resolution increase; in other words, we use identity activation in residual blocks depicted in Figure 5.2 in PadéNet architecture. Moreover, to investigate the

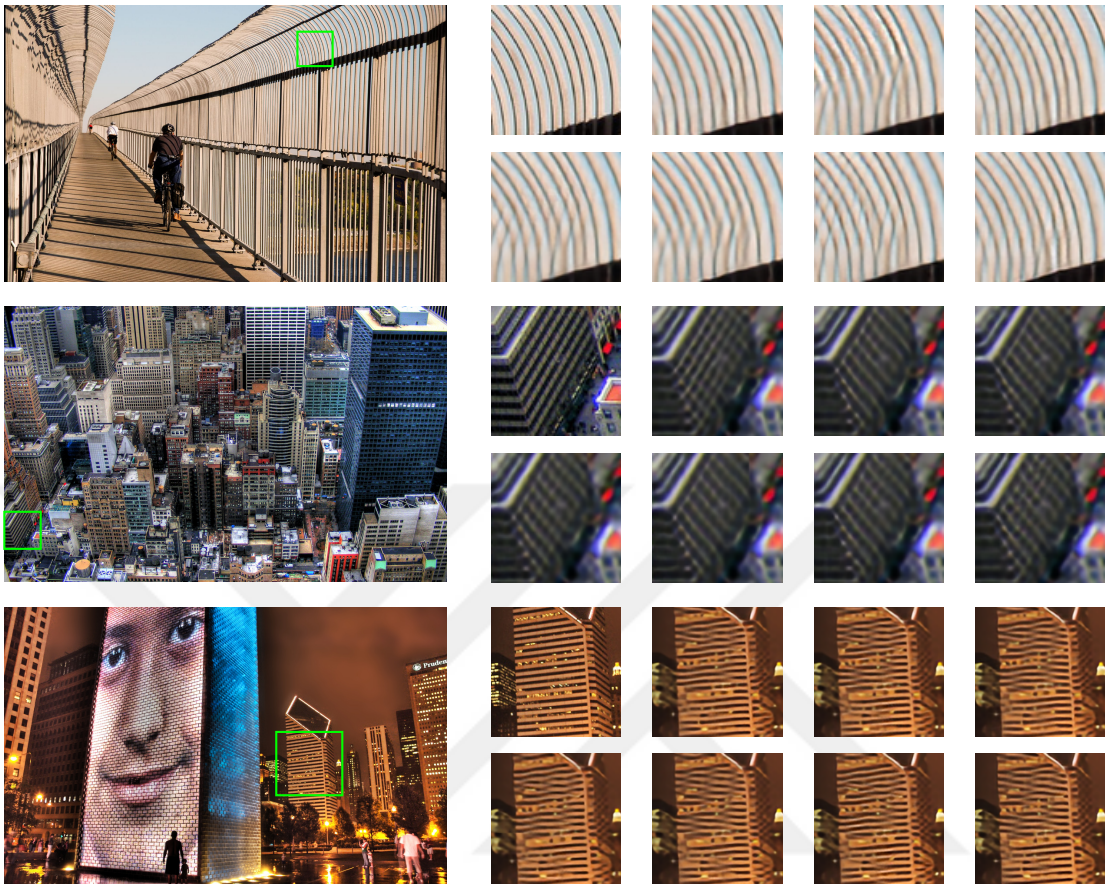


Figure 5.5: Visual comparisons on `img_024.png`, `img_073.png` and `img_076.png` images from Urban100 dataset for $\times 4$ super-resolution. From left to right, top row crops are from ground truth, PadéNet, Self-ONN and SuperONN outputs, and the bottom crops are from ResNet, PAU-Net, DCN 1×1 and DCN 3×3 outputs.

potential of PadéNets, we decrease the number of channels in the residual feature refinement part. As the first shallow feature extraction part still has the original number of channels, we use two *PaLa*s using 3×3 kernels, which are placed in the network before the first residual block and just after the final residual block, to match the number of feature channels. The resultant architecture for different number of channels in the residual blocks is shown in Figure 5.6. The configurations of PadéNets without any activation (PadéNet-ID), and PadéNets with less number of channels in the residual blocks containing 24 (PadéNet-24) and 32 (PadéNet-32) channels are given in Table 5.8.

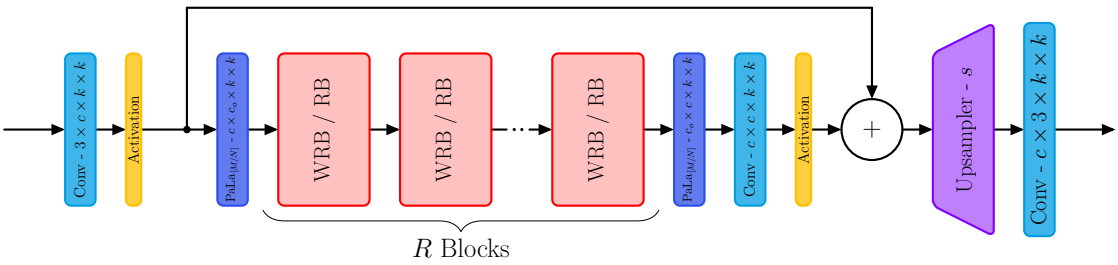


Figure 5.6: The architecture for different number of channels in the residual block part. Here, two *PaLa*s are placed to adjust the number of channels suitable for the residual blocks.

Table 5.8: Model configurations. Degrees denote the degree of numerator/denominator polynomials $[M/N]$. "Deform." means deformable shift.

Attribute \ Model	PadéNet-ID	PadéNet-24	PadéNet-32
Degrees	[1/1]	[1/1]	[1/1]
Activation	Identity	GELU	GELU
Blocks, R	3	3	3
Type (w)	RB (1)	RB (1)	RB (1)
Channels	48	24	32
Strategy	Deform.	Deform.	Deform.
Shift Kernel	1×1	1×1	1×1

Table 5.9: Quantitative comparison of PadéNet without an external activation, and PadéNets with less number of channels. The top two scores in each cell are PSNR(↑) and SSIM(↑), and the bottom two are LPIPS(↓) calculated via AlexNet and VGGNet, respectively. Under the names of each model is the approximate number of parameters. The best and second best scores for each dataset are shown in **blue** and red, respectively.

Scale	Model \ Dataset	BSD100	Manga109	Set5	Set14	Urban100
×2	PadéNet	30.68/0.8909	35.69/0.9726	**35.55**/0.9549	31.14/0.9076	29.93/0.9168
	≈ 362K	0.1500/**0.1571**	0.0233/0.0570	0.0580/0.0968	0.0965/0.1449	0.0723/0.1186
	PadéNet-24	30.64/0.8903	35.49/0.9720	35.53/0.9548	31.07/0.9069	29.72/0.9145
	≈ 214K	**0.1490**/0.1580	0.0234/0.0581	0.0582/0.0966	0.0986/0.1462	0.0756/0.1208
	PadéNet-32	30.67/0.8907	35.64/0.9724	35.53/**0.9550**	31.12/0.9076	29.85/0.9162
	≈ 277K	0.1492/0.1575	**0.0232**/**0.0568**	0.0583/0.0962	0.0976/0.1449	0.0734/0.1193
	PadéNet-ID	**30.69**/**0.8910**	**35.71**/**0.9727**	**35.55**/**0.9550**	**31.15**/**0.9078**	**29.94**/**0.9171**
	≈ 362K	0.1508/0.1575	**0.0232**/0.0571	**0.0579**/**0.0959**	**0.0964**/**0.1446**	**0.0716**/**0.1183**
×4	PadéNet	26.15/0.7139	28.25/0.8935	**29.96**/0.8792	26.37/0.7604	24.35/0.7622
	≈ 445K	0.3868/0.3434	0.1146/**0.1717**	**0.1797**/**0.2163**	**0.2943**/0.3073	0.2503/0.2933
	PadéNet-24	26.10/0.7121	28.06/0.8904	29.86/0.8779	26.30/0.7579	24.20/0.7572
	≈ 297K	0.3889/0.3451	0.1166/0.1743	0.1798/0.2185	0.2970/0.3095	0.2554/0.2978
	PadéNet-32	26.13/0.7133	28.15/0.8922	29.91/0.8785	26.35/0.7591	24.30/0.7605
	≈ 361K	0.3867/0.3437	0.1149/0.1738	**0.1797**/0.2186	0.2955/0.3083	0.2512/0.2952
	PadéNet-ID	**26.17**/**0.7144**	**28.28**/**0.8940**	**29.96**/**0.8793**	**26.39**/**0.7607**	**24.38**/**0.7636**
	≈ 445K	**0.3864**/**0.3428**	**0.1136**/**0.1717**	0.1799/0.2171	0.2946/**0.3064**	**0.2476**/0.2924

The quantitative comparison between PadéNet in Table 5.6 and those configured as in Table 5.8 is given in Table 5.9. It is clear that PadéNet without any external activation performs at least equal to, but generally better than, the PadéNet using GELU activation in the fidelity scores (PSNR and SSIM) for all dimension increase tasks. Moreover, PadéNet benefiting only from its inherent non-linearity generally improves the perceptual metrics over PadéNet using an external non-linear function. Furthermore, it has even the potential to outperform rival models by using fewer channels. These results indicate the strong learning capacity of the inherently non-linear *Paon*s.

In order to explore whether we can increase the performance of PadéNet within

Table 5.10: Quantitative comparison around 360K parameters with shared pixel shuffler for the ×4 super-resolution task. The top two scores in each cell are PSNR(↑) and SSIM(↑), and the bottom two are LPIPS(↓) calculated via AlexNet and VGGNet, respectively. Under the names of each model is the approximate number of parameters. The best and second best scores for each dataset are shown in **blue** and red, respectively.

Model \ Dataset	BSD100	Manga109	Set5	Set14	Urban100
ResNet	26.08/0.7114	27.93/0.8887	29.70/0.8757	26.17/0.7579	24.13/0.7546
≈ 357K	0.3914/0.3455	0.1202/0.1737	0.1827/0.2189	0.2977/0.3103	0.2618/0.2996
DCN 1 × 1	26.13/0.7127	28.08/0.8910	29.81/0.8769	26.33/0.7592	24.25/0.7584
≈ 363K	0.3890/0.3444	0.1177/0.1727	0.1807/0.2189	0.2963/0.3081	0.2563/0.2968
DCN 3 × 3	26.13/0.7126	28.09/0.8910	**29.89**/0.8778	**26.34**/0.7592	24.26/0.7585
≈ 426K	0.3889/0.3443	0.1175/0.1723	**0.1805**/0.2178	0.2954/0.3083	0.2559/0.2963
Self-ONN	26.10/0.7118	28.03/0.8898	29.84/0.8772	26.29/0.7585	24.18/0.7557
≈ 357K	0.3916/0.3457	0.1193/**0.1714**	**0.1805**/**0.2171**	0.2983/0.3103	0.2608/0.2977
SuperONN	26.10/0.7117	27.96/0.8887	29.78/0.8760	26.27/0.7577	24.17/0.7555
≈ 357K	0.3918/0.3451	0.1192/0.1721	0.1819/0.2180	0.2978/0.3103	0.2598/0.2972
PadéNet-ID	**26.15**/**0.7131**	**28.21**/**0.8930**	29.86/**0.8779**	**26.34**/**0.7599**	**24.32**/**0.7615**
≈ 362K	**0.3868**/**0.3437**	**0.1156**/0.1716	0.1809/0.2178	**0.2946**/**0.3078**	**0.2524**/**0.2942**

the same parameter budget as in Table 5.7, we decide to use two ×2 pixel shuffler layers with shared weights instead of two independent ×2 pixel shuffler layers, and then add one more residual block to the competing models. Therefore, in the following experiments, the results are only presented for ×4 super-resolution task. Moreover, inspired by the quantitative results presented in Table 5.9, we omit the external non-linear activation function between *PaLa*s in the residual blocks of the PadéNet architecture, as that configuration quantitatively shows better performance. Finally, due to the evident lower metric scores, we decide to exclude the PAU-Net architecture from the conducted experiments.

For the first step in which we drop one of the pixel shuffler layers, the quantitative results are presented in Table 5.10. Comparison of the metric scores in it with Table

Table 5.11: Quantitative comparison around 450K parameters with shared pixel shuffler and 4 residual blocks for $\times 4$ super-resolution task. The top two scores in each cell are PSNR(↑) and SSIM(↑), and the bottom two are LPIPS(↓) calculated via AlexNet and VGGNet, respectively. Under the names of each model is the approximate number of parameters. The best and second best scores for each dataset are shown in **blue** and red, respectively.

Model \ Dataset	BSD100	Manga109	Set5	Set14	Urban100
ResNet	26.13/0.7132	28.13/0.8919	29.79/0.8777	26.26/0.7595	24.26/0.7594
≈ 440K	0.3894/0.3434	0.1176/0.1722	0.1809/0.2181	0.2957/0.3091	0.2559/0.2958
DCN 1×1	26.16/0.7143	28.26/0.8934	29.95/0.8789	26.39/0.7609	24.35/0.7622
≈ 449K	0.3873/**0.3428**	0.1155/**0.1698**	0.1793/**0.2163**	0.2940/0.3070	0.2522/**0.2925**
DCN 3×3	26.16/0.7140	28.22/0.8930	29.92/0.8788	26.36/0.7605	24.35/0.7622
≈ 532K	0.3875/0.3439	0.1160/0.1718	0.1799/0.2172	0.2955/0.3078	0.2524/0.2940
Self-ONN	26.14/0.7132	28.15/0.8918	29.83/0.8776	26.35/0.7596	24.29/0.7600
≈ 440K	0.3902/0.3445	0.1174/0.1704	0.1816/0.2176	0.2966/0.3087	0.2553/0.2939
SuperONN	26.13/0.7128	28.09/0.8910	29.84/0.8775	26.32/0.7591	24.27/0.7595
≈ 440K	0.3902/0.3451	0.1172/0.1719	0.1823/0.2180	0.2968/0.3094	0.2552/0.2945
PadéNet-ID	**26.18**/**0.7149**	**28.32**/**0.8944**	**29.99**/**0.8799**	**26.41**/**0.7611**	**24.41**/**0.7645**
≈ 447K	**0.3857**/0.3430	**0.1145**/0.1708	**0.1791**/0.2176	**0.2933**/**0.3064**	**0.2488**/0.2927

5.7 shows that there is a small performance loss due to the use of shared $\times 2$ pixel shuffler layer for the $\times 4$ super-resolution task, as expected. Later, we add one more residual block to each model using the parameters saved by using shared weights in the two pixel shuffler layers. The results are shown in Table 5.11. The fidelity and perceptual metric scores indicate that we gain more than what we lose in all models within the same approximately 450K parameter budget. Together with that, the comparison of PadéNet-ID scores in Table 5.9 with the other architectures in Table 5.11 indicates that other models require a deeper residual refinement part to catch the performance of PadéNets having fewer blocks to refine the residual features.

5.2 Image Compression

Image compression is a data compression application in which the aim is to reduce the file size of digital images as much as possible, and also when lossy, to introduce artifacts as little as possible. In this section, *Paon*s are tested as a replacement alternative for convolutional layers. In the first subsection, *PaLa*s are used as a direct substitute, while in the second set of experiments, the number of layers is also reduced after substitution. For all the experiments in this section, the smooth version of proposed *Paon*, *Paon–S* in Equation (4.5), is used.

5.2.1 Direct Replacement

For this experiment, we choose the joint autoregressive hierarchical priors model by Minnen et al. [Minnen et al., 2018]. The architecture can be investigated in two main categories, transmitter and receiver parts. In the transmitter part, the image passes through an image encoder network that converts it into a latent representation, which is quantized before entering the arithmetic encoder. Meanwhile, another network, called hyper-prior encoder, further processes the latent features and then quantizes the output to provide correction for context-based predictions of the distribution parameters. In the receiver part, context model and hyper-encoder modules provide information to be used as entropy parameters. Using these, the arithmetic decoder converts the bit stream into the latent representation, which is converted to the image by the image decoder network. The pipeline is shown in Figure 5.7.

In this architecture, we decide to change the layers of the image encoder and image decoder parts. Both have four convolutional layers, between which there are three non-linear GDN operations. All layers use 5×5 kernels. The image encoder decreases the resolution in each layer by 2 using strided convolutions, while the image decoder uses transposed convolutions with stride 2 to increase the resolution. Every layer has N output channels except for the final layer of the encoder, which has M neurons, and the final layer of the decoder having 3 channels. In addition, every layer has N input channels except for the first layer of the encoder, which has

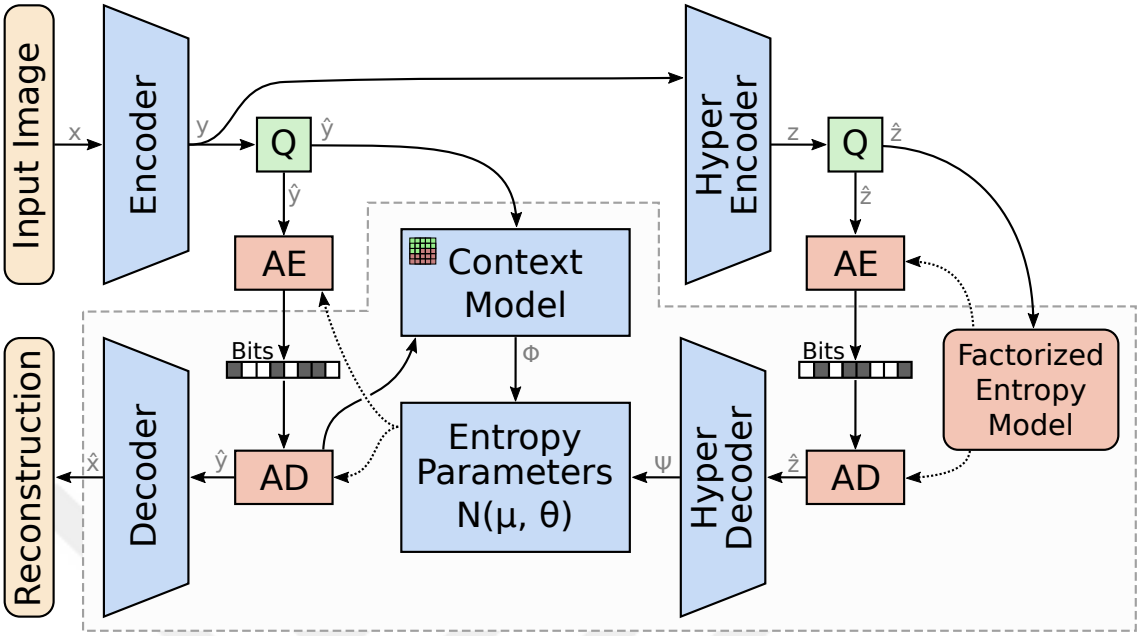


Figure 5.7: Illustration of the joint autoregressive hierarchical priors model. Only the convolutional layers in image encoder and decoder parts are replaced by *PaLa*s, the rest are kept unchanged. "Q", "AE" and "AD" refer to quantization, arithmetic encoding and arithmetic decoding, respectively. Directly taken from [Minnen et al., 2018].

3 channels, and the first layer of the decoder, which has M channels[1]. For the image encoder part, we use *PaLa*s with the enhanced Shifter module using deformable convolutions [Dai et al., 2017], and for the decoder part we use the first shifting strategy with transposed convolutions within *PaLa*, all of which are set to degrees [1/1]. The rest of the modules, namely hyper encoder and decoder, context model and the model estimating the entropy parameters, are kept unchanged as in the original model in [Minnen et al., 2018].

In addition to directly replacing convolutional layers with *Paon*s, we also establish an alternative configuration to observe how the inherent non-linear capabilities of *Paon*s compare to conventional layers having an external activation function.

[1]Please note that here M and N denote the number of channels, not numerator and denominator degrees in *Paon*s.

Therefore, we also remove the non-linear GDN layers from the PadéNet version of the joint autoregressive model.

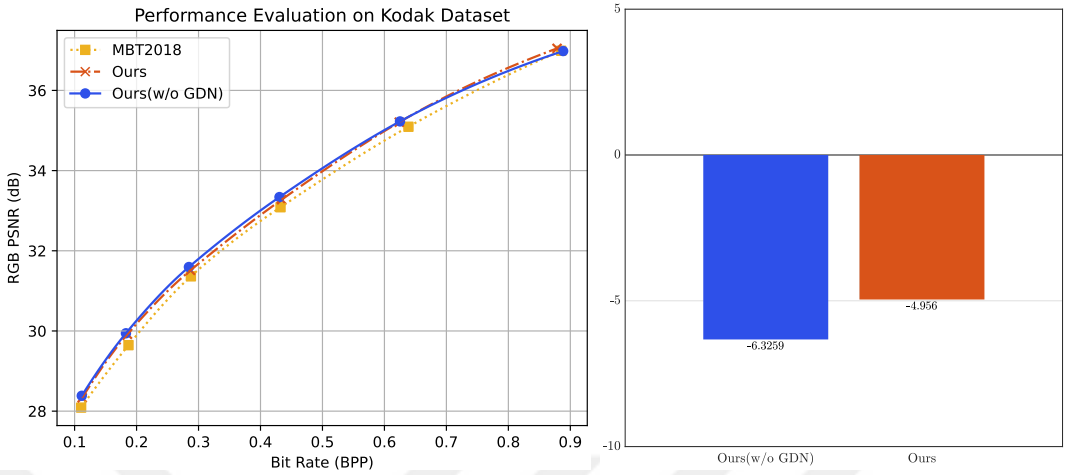
For training, we use a combination of selected images from the ImageNet dataset [Deng et al., 2009], DF2K [Lim et al., 2017], COCO 2017 [Lin et al., 2014] and CLIC train dataset [Toderici et al., 2020], resulting in a dataset that includes more than 100 000 images. We perform image compression experiments using the code base provided by the MLIC++ paper [Jiang and Wang, 2023] and the CompressAI framework [Bégaint et al., 2020]. In each experiment, random crops of size 256×256 are taken and training is carried out with a batch size of 24. The models are optimized for different values of the rate-distortion loss trade-off parameter $\lambda = \{0.0018, 0.0035, 0.0067, 0.0130, 0.0250, 0.0483\}$. The models are trained with Adam optimizer [Kingma and Ba, 2015] for 600 epochs with an initial learning rate of 10^{-4}. The learning rate is reduced by a factor of 10 in epochs 450 and 550. Furthermore, we apply gradient clipping to stabilize the training process by limiting the maximum norm of the gradients to 1.

Performance is evaluated on the Kodak dataset [Franzen, 2013] using the PSNR and BD-rate scores [Bjøntegaard, 2001]. PSNR is calculated on RGB images [Keleş et al., 2021b], and the BD-rate is computed using the script provided by Herglotz et al. [Herglotz et al., 2022].

The number of channels is set to $M = 192$ and $N = 192$ for the first four values of λ, and changed to $M = 320$ and $N = 192$ for the last two values[2]. The original model has 14.13M parameters for the first four λ, and 25.50M parameters for the last two. PadéNet version with GDN has 20.02M and 32.75M parameters, respectively, and PadéNet without GDN has 19.79M and 32.53M parameters, respectively.

Figure 5.8 shows the performance of the architectures with the proposed modifications. The values for drawing the RD curves of the original model by Minnen et al. [Minnen et al., 2018] are taken from the CompressAI benchmark [Bégaint et al., 2020]. In Figure 5.8a, RD curves of PadéNet versions of the joint autoregressive

[2]Again, please note that here M and N denote the number of channels, as this is the common practice, not numerator and denominator degrees in *Paons*.



(a) (b)

Figure 5.8: Graphical comparison of the modified model with the benchmark. (a) Rate-distortion curves of the proposed models and the network in Minnen et al. [Minnen et al., 2018]. (b) Average percent BD-rate savings calculated for RGB PSNR values with respect to the anchor model in [Minnen et al., 2018].

model show their superiority because they are located closer to the upper-left corner of the rate-distortion graph, which means that it requires fewer bits to display an image for a given quality. Notably, PadéNet without an external non-linearity also surpasses the original model. The superiority of PadéNet can also be understood from Figure 5.8b. The bar graph shows the bit rate savings calculated by the BD-rate [Bjøntegaard, 2001]. In this scenario, the model equipped with *PaLa*s consumes nearly 5% less bit rate on average than the anchor model which uses common convolutional layers. More strikingly, the model relying only on the non-linear power of *PaLa*s even extends the lead and saves more than 6% bit rate compared to the off-the-shelf model. These results indicate that even simply substituting the common convolutional layers with *PaLa*s brings a performance improvement without any more bells and whistles.

Visual examples of the model outputs are given in Figure 5.9. For the boxed area with red rectangle in `kodim08.png`, the result of PadéNet is cleaner and sharper

Figure 5.9: Visual evaluation of reconstructed `kodim08.png`, `kodim01.png` and `kodim21.png` images, respectively, in Kodak dataset. Crops are taken from ground truth, PadéNet version of joint autoregressive network with GDN, and original joint autoregressive models, respectively. For `kodim08.png`, the crops are taken from the models trained with $\lambda = 0.0018$, and the others are taken from the models with $\lambda = 0.0035$ in the rate-distortion loss.

compared to the output of the original joint autoregressive model. Also, the lines of window shutters start to appear in PadéNet's output, whereas in the original model's output, that region is smooth. In addition, the calligraphic A contains slightly more details in the output of the modified version of the joint autoregressive model. For the green crops in the same image, PadéNet is able to reconstruct the horizontal parallel lines on the roof, which the original model fails to do so, and the top of the antenna pole, which is more faded in the output of the joint autoregressive model. The same difference in reconstruction power can be seen in the crops extracted from `kodim01.png` image, in which PadéNet output has some clear parallel lines for the window blinds and the anchor model does not. PadéNet is also superior in areas with

fewer details. In image `kodim21.png`, the crop taken from the sky has darker stripes and a minor color shift in the output of off-the-shelf model whereas for modified network output, the fidelity is better preserved.

5.2.2 Replacement and Reduction of Blocks

In this experiment, we not only change the chosen layers to *PaLa*s but also reduce the number of some used blocks after the change to see whether the proposed neuron model can achieve more with less. For that purpose, we choose the efficient learned image compression (ELIC) model [He et al., 2022]. The main components are the same as the model of Minnen et al. [Minnen et al., 2018]. The transmitter side has an image encoder, a latent quantizer, a hyper encoder followed by a quantizer, and arithmetic encoder. The receiver side has an arithmetic decoder followed by a hyper decoder network, a context model for estimating the Gaussian parameters, an arithmetic decoder to obtain the latent representation, and an image decoder converting those latents into an image. The architecture is shown in Figure 5.10.

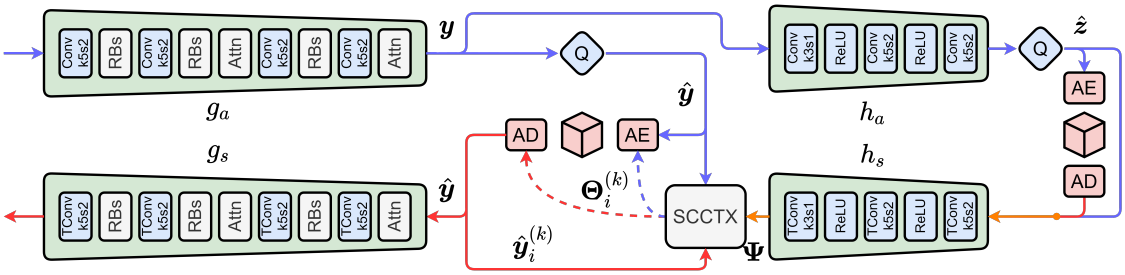
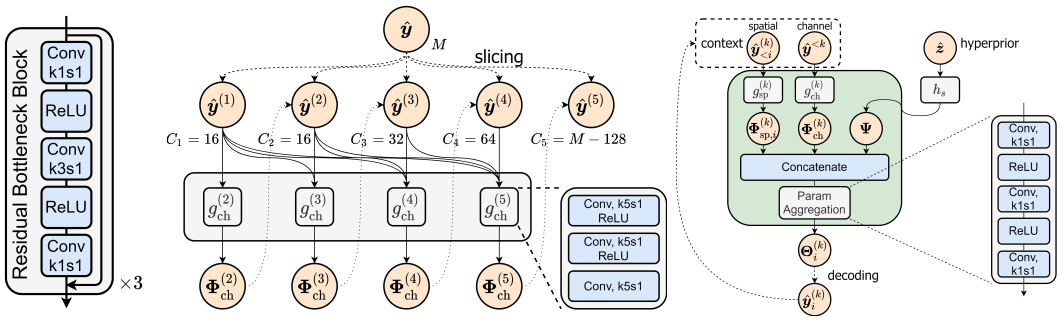


Figure 5.10: ELIC architecture. g_a, g_s, h_a and h_s are image encoder, image decoder, hyper encoder and hyper decoder networks, respectively. "Q", "AE", "AD" and "SCCTX" refer to quantization, arithmetic encoding, arithmetic decoding, and space-channel context model, respectively. Directly taken from [He et al., 2022].

Although the architecture overviews are the same, there are certain minor and major differences. The minor difference is the activation function used in hyper



(a) (b) (c)

Figure 5.11: Modified parts of the ELIC architecture. (a) Residual bottleneck block. All convolutional layers are converted into *PaLa*s, and the number of blocks are reduced to 1. (b) Channel-wise context model. All 5×5 layers are changed to *PaLa*. (c) Space-channel context model. All convolutonal layers in the spatial context model are modified. Directly taken from [He et al., 2022].

networks. Minnen et al. [Minnen et al., 2018] use leaky ReLU [Maas et al., 2013] in their hyper networks whereas ReLU is used in ELIC. Other than that, hyper networks are the same in terms of kernel and channel size, as well as stride and convolution type. For the PadéNet experiments, we do not touch the hyper network part and keep them as they are in ELIC.

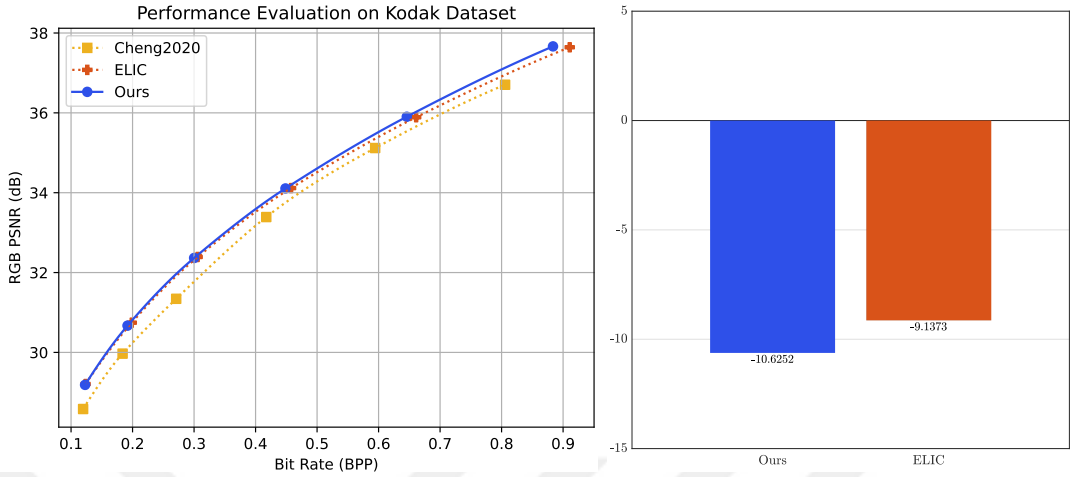
The first major difference is the size of the image encoder and decoder modules. The model of the study by Minnen et al. [Minnen et al., 2018] has simply four convolutional layers in each of them. However, in ELIC, these networks have two attention blocks [Cheng et al., 2020], four independent convolutional layers, and three residual bottleneck block groups each of which has three residual bottleneck blocks shown in Figure 5.11a. He et al. [He et al., 2022] claim that to replace the GDN and its inverse layers [Ballé et al., 2016a], it is necessary to stack enough non-linearity, thus the size of image encoder and decoder blocks. For more information on the attention block used in residual bottleneck blocks, please refer to Appendix A.4.

The second big difference is the context model. Minnen et al. [Minnen et al., 2018] use a single layer masked convolution [van den Oord et al., 2016], in which

the current pixel can be generated only from the previously generated pixels, as the context model. In ELIC, the space-channel context model is proposed. As the name implies, there are two sub-parts in the space-channel context model. The channel-wise context model separates the input features in the channel dimension unevenly. The size of the slices is chosen as 16, 16, 32, 64, and $M - 128$, respectively. Each slice is processed by a small module containing three 5×5 convolutional layers and two ReLU non-linearities, which are shown in Figure 5.11b. The space context model also slices the input features in the same way, but it has single 5×5 convolution layer without any non-linearity for each of the slices. After being processed, the slices coming from channel and spatial context models as well as hyper parameters are concatenated and aggregated by a network containing three 1×1 convolutional layers and two ReLU activations. The illustration of the spatial-channel context model is shown in Figure 5.11c.

In line with the approach described in the work of Luo [Luo, 2024], we modify only the decoder and context model layers. In the image decoder part, we convert the convolutional layers inside the residual bottleneck block, shown in Figure 5.11a, into their corresponding *PaLa*. For the layer with 3×3, we use deformable kernels, and for the layers with 1×1 kernels, we do not use shifting. Moreover, we decrease the number of blocks from 3 to 1 in the decoder. In addition, we substitute independent transposed convolutions with their corresponding *PaLa*s using the first shifting strategy and do not touch the attention blocks at all. In the channel-wise context model shown in Figure 5.11b, we change all layers in small modules processing different slices to *PaLa*s using the deformable strategy, since all those layers use 5×5 kernels. Finally, all layers of the spatial context part in the spatial-channel context module are converted to *PaLa*s using the enhanced Shifter module. In all substitutions, we set the degrees of the numerator and denominator to 1.

The training settings for PadéNet version of ELIC are exactly the same as described in Section 5.2.1 for the dataset, code base, loss, optimizer, λ values, learning rate setting and gradient clipping. The input number of channels in the image decoder M and the number of channels in the middle layers in image decoder



(a) (b)

Figure 5.12: Graphical comparison of the modified and reduced model with benchmarks. (a) Rate-distortion curves of the proposed model, the model in Cheng et al. [Cheng et al., 2020] and ELIC. (b) Average percent BD-rate savings calculated for RGB PSNR values with respect to the anchor model in [Cheng et al., 2020].

N for ELIC are set to 320 and 192, respectively, for all the values of λ[3]. ELIC has 36.93M parameters, and the PadéNet version has 48.78M parameters.

Performance plots for the modified ELIC model are shown in Figure 5.12. The RD curve for the anchor model of Cheng et al. [Cheng et al., 2020] is drawn by the values taken from the CompressAI benchmark [Bégaint et al., 2020], and ELIC is trained from scratch as its RD values are not available in CompressAI. As Figure 5.12a indicates, ELIC with *PaLa*s shows superior performance compared to its competitors. Remarkably, this is achieved in the case where the modifications are limited to the image decoder and context model parts only. More interestingly, PadéNet surpasses its ancestor model even if the modified version has significantly fewer layers compared to the original model. This superiority can also be seen in Figure 5.12b. Although the total number of layers is reduced by keeping only one-third of the residual

[3]Please note that here M and N denote the number of channels, not numerator and denominator degrees in *Paons*.

bottleneck blocks that the original model has, the modified ELIC saves more than 1% bit rate compared to the original model. These results quantitatively indicate that the representative power of *Paon*s achieved by the inherent non-linearity and feature shift is enough to reduce the number of layers in a convolutional architecture by substituting them with *PaLa*s.

Chapter 6

CONCLUSION

6.1 Summary

In this thesis, a novel and superior neuron model is presented called the Padé approximant neuron, or *Paon* in short. Inspired by the rational function approximation, *Paon*s use the Padé approximants to increase the non-linear capacity of a neuron not only by using the higher orders of the input features in the activation map calculation but also introducing them as a denominator term to form the final output. Also being better than the Taylor series expansion, the use of Padé approximants in the proposed *Paon*s increases the function approximation capability of networks.

Against the possible singularity problem of rational polynomial approximations that threatens the continuous mapping of neural networks, two different solutions are proposed, creating two distinct versions of *Paon*s. The first version, $Paon{-}A$, takes absolute value of the polynomial terms, which ensures that the denominator of the rational approximation stays away from 0. The second version, $Paon{-}S$, uses the smooth Padé approximants which in our implementation prevent the denominator from operating in a range that could create numerical instability. Although both versions work as intended, the second variant inherently achieves higher non-linearity with fewer number of parameters and multiplication operations compared to the first option if the same non-linearity is desired.

To increase the receptive field and the better feature extraction ability of *Paon*s, two different Shifter modules are designed. The first variant, Shifter-I, has the ability to learn the required shifts either by back-propagation as in the super neurons or by a very small network that allows data-dependent shift extraction. The enhanced option, Shifter-II, benefits from the approach in deformable convolution networks

that learn offsets in an element-wise manner for convolutional kernels. Together with Shifter, *Paon*s form a super set of previously known neuron models.

Experiments on the single image super-resolution problem qualitatively and quantitatively show that when the number of parameters is kept equal so that the only factor that affects performance comes from the processing power of tested models, *Paon*s are superior compared to their competitors. Moreover, when the Shifter module is used in its enhanced variant, the performance gain is higher even compared to the first Shifter version. Furthermore, the experimental results also indicate that removal of the external activation function does not hamper the performance; on the contrary, it even improves the metric scores in most cases.

Experiments on the image compression problem show that direct replacement of common convolutional layers with *PaLa*s increases the model performance. They also remarkably show that even without external activation functions, PadéNets are able to surpass their ancestor model in which the chosen convolutional and non-linear layers are replaced by *PaLa*s. Further experiments quantitatively indicate that even when the number of layers is reduced after replacement, the new PadéNet shows a performance equal to or even slightly better than the compared model.

6.2 Possible Future Work

Experiments show that *Paon* is a powerful alternative for common and recently proposed neurons. However, the performance gain after changing the Shifter strategy indicates that there may still be room for improvement. Mainly originating from deformable kernels [Dai et al., 2017], the increase in Shifter performance could be achieved by other later approaches such as transformable convolutions [Xiao et al., 2018], DCNv2 [Zhu et al., 2019], DCNv3 [Chen et al., 2021] or DCNv4 [Xiong et al., 2024], which may also help *Paon*s boost their feature extraction capacity.

The main idea of *Paon*s is to use better mathematical tools for function approximation. That being said, the employed Padé approximants may cause numerical issues around the singularities of the denominator polynomials. That problem is overcome by two different approaches to the rational and Padé approximants proposed

in Equation (4.4) and Equation (4.5). Although the theory presented in Beckermann and Kalyagin [Beckermann and Kalyagin, 1997] and experiments in Molina et al. [Molina et al., 2020] as well as in this thesis prove that those solutions work for neural networks, the search for better Padé approximants is still an active research area [Stahl, 1998, Gonnet et al., 2013, Mascarenhas, 2015, Beckermann and Matos, 2015, Tylavsky et al., 2022]. Therefore, another method that avoids division by zero of very small numbers might be encountered in previous and future literature. Another research for better mathematical tools may involve the inclusion of other function approximators, such as Chebyshev polynomials [Chebyshev, 1854] that operate on the range $[-1, 1]$, the same as the range that *Paon*s operate, to have the Chebyshev-Padé approximation [Trefethen and Gutknecht, 1987].

Another future work might include implementation of the method in other programming environments that use GPUs much more efficiently. Although *Paon*s are implemented in the PyTorch library [Paszke et al., 2019] which also supports GPU utilization, there are more effective frameworks such as Compute Unified Device Architecture (CUDA) [Kirk, 2007] or Triton [Tillet et al., 2019] programming languages.

Finally, all the experiments presented in this thesis show the performance of *Paon*s in a convolutional layer setting applied to the 2D signals. In the future, it would be interesting to see *Paon*s applied to other architectures and neuron types, such as transformers containing fully connected layers or other architectures that use 1D filters.

BIBLIOGRAPHY

[Agustsson and Timofte, 2017] Agustsson, E. and Timofte, R. (2017). Ntire 2017 challenge on single image super-resolution: Dataset and study. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, page 1122–1131. 19, 50

[Akima, 1970] Akima, H. (1970). A new method of interpolation and smooth curve fitting based on local procedures. *Journal of the ACM (JACM)*, 17(4):589–602. 103

[Bahdanau et al., 2015] Bahdanau, D., Cho, K. H., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations (ICLR)*. 6

[Baker and Graves-Morris, 1996] Baker, G. A. and Graves-Morris, P. (1996). *Padé Approximants*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2 edition. 39

[Ballé et al., 2016a] Ballé, J., Laparra, V., and Simoncelli, E. P. (2016a). Density modeling of images using a generalized normalization transformation. In *4th International Conference on Learning Representations (ICLR)*. 25, 71, 98

[Ballé et al., 2016b] Ballé, J., Laparra, V., and Simoncelli, E. P. (2016b). End-to-end optimization of nonlinear transform codes for perceptual quality. In *2016 Picture Coding Symposium (PCS)*, page 1–5. 104

[Ballé et al., 2017] Ballé, J., Laparra, V., and Simoncelli, E. P. (2017). End-to-end optimized image compression. In *5th International Conference on Learning Representations (ICLR)*, page 2961–2987. 105

[Ballé et al., 2018] Ballé, J., Minnen, D., Singh, S., Hwang, S. J., and Johnston, N. (2018). Variational image compression with a scale hyperprior. In *6th International Conference on Learning Representations (ICLR)*, volume 7, page 4961–5007. 26, 105

[Barron, 2019] Barron, J. T. (2019). A general and adaptive robust loss function. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 4326–4334. 50, 105, 106

[Beckermann and Kalyagin, 1997] Beckermann, B. and Kalyagin, V. A. (1997). The diagonal of the padé table and the approximation of the weyl function of second-order difference operators. *Constructive Approximation*, 13:481–510. 34, 77

[Beckermann and Matos, 2015] Beckermann, B. and Matos, A. C. (2015). Algebraic properties of robust padé approximants. *Journal of Approximation Theory*, 190:91–115. 77

[Bégaint et al., 2020] Bégaint, J., Racapé, F., Feltman, S., and Pushparaja, A. (2020). Compressai: A pytorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*. 67, 73

[Bellard, 2014] Bellard, F. (2014). Bpg image format. `https://bellard.org/bpg/`. Accessed: 2024-12-08. 27

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166. 5

[Bevilacqua et al., 2012] Bevilacqua, M., Roumy, A., Guillemot, C., and Morel, M.-L. A. (2012). Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *Proceedings of the British Machine Vision Conference (BMVC)*, page 135.1–135.10. BMVA Press. 50, 56

[Bjøntegaard, 2001] Bjøntegaard, G. (2001). Calculation of average psnr differences between rd curves. In *ITU-T SG16/Q6, 13th VCEG Meeting, Austin, Texas, USA, April 2001*. 27, 67, 68, 102

[Brooks et al., 2024] Brooks, T., Peebles, B., Holmes, C., DePue, W., Guo, Y., Jing, L., Schnurr, D., Taylor, J., Luhman, T., Luhman, E., Ng, C., Wang, R., and Ramesh, A. (2024). Video generation models as world simulators. https://openai.com/research/video-generation-models-as-world-simulators. 1

[Chan et al., 2022] Chan, K. C., Zhou, S., Xu, X., and Loy, C. C. (2022). Basicvsr++: Improving video super-resolution with enhanced propagation and alignment. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 5962–5971. 37

[Charbonnier et al., 1994] Charbonnier, P., Blanc-Feraud, L., Aubert, G., and Barlaud, M. (1994). Two deterministic half-quadratic regularization algorithms for computed imaging. In *Proceedings of 1st International Conference on Image Processing (ICIP)*, volume 2, page 168–172. 106

[Chebyshev, 1854] Chebyshev, P. L. (1854). Théorie des mécanismes connus sous le nom de parallélogrammes. *Mémoires des Savants étrangers présentés à l'Académie de Saint-Pétersbourg*, 7:539–568. 77

[Chen et al., 2021] Chen, F., Wu, F., Xu, J., Gao, G., Ge, Q., and Jing, X.-Y. (2021). Adaptive deformable convolutional network. *Neurocomputing*, 453:853–864. 76

[Chen et al., 2020] Chen, Y., Dai, X., Liu, M., Chen, D., Yuan, L., and Liu, Z. (2020). Dynamic convolution: Attention over convolution kernels. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 11027–11036. 5

[Cheng et al., 2020] Cheng, Z., Sun, H., Takeuchi, M., and Katto, J. (2020). Learned image compression with discretized gaussian mixture likelihoods and attention modules. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 7936–7945. xvii, xviii, 71, 73, 99

[Cheung and Leung, 1991] Cheung, K. F. and Leung, C. S. (1991). Rotational quadratic function neural networks. In *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*, volume 1, page 869–874. 5

[Chollet, 2017] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 1800–1807. 40

[Cuyt, 1999] Cuyt, A. (1999). How well can the concept of padé approximant be generalized to the multivariate case? *Journal of Computational and Applied Mathematics*, 105(1):25–50. 32

[Dai et al., 2017] Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., and Wei, Y. (2017). Deformable convolutional networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, page 764–773. 37, 66, 76

[de Montessus de Ballore, 1902] de Montessus de Ballore, R. (1902). Sur les fractions continues algébriques. *Bulletin de la Société Mathématique de France*, 30:28–36. 32

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 248–255. 67

[Dosovitskiy et al., 2021] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words:

Transformers for image recognition at scale. In *9th International Conference on Learning Representations (ICLR)*. 40

[Dosovitskiy and Brox, 2016] Dosovitskiy, A. and Brox, T. (2016). Generating images with perceptual similarity metrics based on deep networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 29, page 658–666. Curran Associates, Inc. 102

[Franzen, 2013] Franzen, R. (2013). Kodak lossless true color image suite. `https://r0k.us/graphics/kodak/`. Accessed: 2024-12-08. 27, 67

[Fritsch and Carlson, 1980] Fritsch, F. N. and Carlson, R. E. (1980). Monotone piecewise cubic interpolation. *SIAM Journal on Numerical Analysis*, 17(2):238–246. 103

[Fukushima, 1969] Fukushima, K. (1969). Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333. 3, 4

[Funahashi, 1989] Funahashi, K.-I. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192. 3, 30

[Giles and Maxwell, 1987] Giles, C. L. and Maxwell, T. (1987). Learning, invariance, and generalization in high-order neural networks. *Applied Optics*, 26(23):4972–4978. 5, 6, 10

[Gonnet et al., 2013] Gonnet, P., Güttel, S., and Trefethen, L. N. (2013). Robust padé approximation via svd. *SIAM Review*, 55(1):101–117. 77

[Gu and Dao, 2024] Gu, A. and Dao, T. (2024). Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling (COLM)*. 7

[Gu et al., 2022] Gu, A., Goel, K., and Re, C. (2022). Efficiently modeling long sequences with structured state spaces. In *The Tenth International Conference on Learning Representations (ICLR)*. 7

[Gu et al., 2021] Gu, A., Johnson, I., Goel, K., Saab, K., Dao, T., Rudra, A., and Ré, C. (2021). Combining recurrent, convolutional, and continuous-time models with linear state space layers. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, page 572–585. Curran Associates, Inc. 7

[Guillaume and Huard, 2000] Guillaume, P. and Huard, A. (2000). Multivariate padé approximation. *Journal of Computational and Applied Mathematics*, 121(1):197–219. 32

[Hahnloser and Seung, 2000] Hahnloser, R. and Seung, H. S. (2000). Permitted and forbidden sets in symmetric threshold-linear networks. *Advances in Neural Information Processing Systems (NIPS)*, 13:217–223. 4

[He et al., 2022] He, D., Yang, Z., Peng, W., Ma, R., Qin, H., and Wang, Y. (2022). Elic: Efficient learned image compression with unevenly grouped space-channel contextual adaptive coding. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 5708–5717. xvii, 70, 71

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, page 1026–1034. 36

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 770–778. 6, 18, 48

[Hendrycks and Gimpel, 2016] Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*. 48, 49, 99

[Herglotz et al., 2022] Herglotz, C., Kränzler, M., Mons, R., and Kaup, A. (2022). Beyond bjøntegaard: Limits of video compression performance comparisons. In *2022 IEEE International Conference on Image Processing (ICIP)*, page 46–50. 67, 103

[Hochreiter, 1991] Hochreiter, J. (1991). Untersuchungen zu dynamischen neuronalen netzen. Master's thesis, Technische Universität München. Diploma thesis, Advisor: Jürgen Schmidhuber. 3, 31

[Homma et al., 1987] Homma, T., Atlas, L., and Marks, R. (1987). An artificial neural network for spatio-temporal bipolar patterns: Application to phoneme classification. In *Neural Information Processing Systems (NIPS)*, volume 0, page 31–40. American Institute of Physics. 2

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366. 3, 30

[Householder, 1941] Householder, A. S. (1941). A theory of steady-state activity in nerve-fiber networks: I. definitions and preliminary lemmas. *The Bulletin of Mathematical Biophysics*, 3:63–69. 4

[Huang et al., 2015] Huang, J.-B., Singh, A., and Ahuja, N. (2015). Single image super-resolution from transformed self-exemplars. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 5197–5206. 50, 56

[Jaderberg et al., 2015] Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukçuoğlu, K. (2015). Spatial transformer networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 28, page 2017–2025. Curran Associates, Inc. 16, 36

[Jia et al., 2016] Jia, X., De Brabandere, B., Tuytelaars, T., and Gool, L. V. (2016).

Dynamic filter networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 29, page 667–675. Curran Associates, Inc. 5, 40

[Jiang and Wang, 2023] Jiang, W. and Wang, R. (2023). Mlic++: Linear complexity multi-reference entropy modeling for learned image compression. In *ICML 2023 Workshop Neural Compression: From Information Theory to Applications*. 67

[Jumper et al., 2021] Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukçuoğlu, K., Kohli, P., and Hassabis, D. (2021). Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589. 1

[Keleş and Tekalp, 2024] Keleş, O. and Tekalp, A. M. (2024). Paon: A new neuron model using padé approximants. In *2024 IEEE International Conference on Image Processing (ICIP)*, page 207–213. xii, xvi, 9, 30, 35, 47, 48, 49, 50, 52, 53, 54, 55

[Keleş et al., 2021a] Keleş, O., Tekalp, A. M., Malik, J., and Kıranyaz, S. (2021a). Self-organized residual blocks for image super-resolution. In *2021 IEEE International Conference on Image Processing (ICIP)*, page 589–593. xi, xiv, xv, 6, 9, 15, 18, 19, 20, 21, 22, 23, 24

[Keleş et al., 2021b] Keleş, O., Yılmaz, M. A., Tekalp, A. M., Korkmaz, C., and Doğan, Z. (2021b). On the computation of psnr for a set of images or video. In *2021 Picture Coding Symposium (PCS)*, page 1–5. 20, 51, 57, 67, 104

[Kingma and Ba, 2015] Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*. 19, 26, 67

[Kirk, 2007] Kirk, D. (2007). Nvidia cuda software and gpu parallel computing architecture. In *ISMM '07. Proceedings of the 6th International Symposium on Memory Management*, volume 7, page 103–104, New York, NY, USA. Association for Computing Machinery. 77

[Kolmogorov, 1957] Kolmogorov, A. N. (1957). On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. In *Doklady Akademii Nauk*, volume 114, page 953–956. Russian Academy of Sciences. 7

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems (NIPS)*, 25:1097–1105. 2, 4, 27, 51, 57, 102

[Kuck, 1974] Kuck, D. J. (1974). *NBS Technical Note 851, Computer System Capacity Fundamentals*. US Department of Commerce, National Bureau of Standards. 42

[Kunc and Kléma, 2024] Kunc, V. and Kléma, J. (2024). Three decades of activations: A comprehensive survey of 400 activation functions for neural networks. *arXiv preprint arXiv:2402.09092*. 5

[Kıranyaz et al., 2021] Kıranyaz, S., Malik, J., Abdallah, H. B., İnce, T., Iosifidis, A., and Gabbouj, M. (2021). Self-organized operational neural networks with generative neurons. *Neural Networks*, 140:294–308. 6, 13, 14, 16, 31

[Kıranyaz et al., 2024] Kıranyaz, S., Malik, J., Yamaç, M., Duman, M., Adalıoğlu, I., Güldoğan, E., İnce, T., and Gabbouj, M. (2024). Super neurons. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 8(1):206–228. 6, 16, 31, 35, 36

[Kıranyaz et al., 2017] Kıranyaz, S., İnce, T., Iosifidis, A., and Gabbouj, M. (2017). Progressive operational perceptrons. *Neurocomputing*, 224:142–154. 13

[Kıranyaz et al., 2020] Kıranyaz, S., İnce, T., Iosifidis, A., and Gabbouj, M. (2020). Operational neural networks. *Neural Computing and Applications*, 32:6645–6668. 6, 10, 11

[LeCun, 1987] LeCun, Y. (1987). *Modeles connexionnistes de lapprentissage*. PhD thesis, These de Doctorat, Universite Paris. 4

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551. 2, 4, 6

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. 4

[Ledig et al., 2017] Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., et al. (2017). Photo-realistic single image super-resolution using a generative adversarial network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 105–114. 48

[Lim et al., 2017] Lim, B., Son, S., Kim, H., Nah, S., and Mu Lee, K. (2017). Enhanced deep residual networks for single image super-resolution. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, page 1132–1140. xiv, 18, 19, 49, 67

[Lin et al., 2014] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, page 740–755. Springer. 67

[Liu et al., 2025] Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., Hou, T. Y., and Tegmark, M. (2025). KAN: Kolmogorov-arnold networks. In *The Thirteenth International Conference on Learning Representations*. 7

[Loshchilov and Hutter, 2017] Loshchilov, I. and Hutter, F. (2017). Sgdr: Stochastic gradient descent with warm restarts. In *The Fifth International Conference on Learning Representations (ICLR)*, volume 3, page 1769–1784. 50

[Ludgate, 1909] Ludgate, P. E. (1909). On a proposed analytical machine. In *The Origins of Digital Computers: Selected Papers*, page 73–87. Springer. 42

[Luo, 2024] Luo, J. (2024). Rethinking learned image compression: Context is all you need. *arXiv preprint arXiv:2407.11590*. 72

[Maas et al., 2013] Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, volume 28, page 3–8. 5, 71

[Malik et al., 2021] Malik, J., Kıranyaz, S., and Gabbouj, M. (2021). Self-organized operational neural networks for severe image restoration problems. *Neural Networks*, 135:201–211. 15

[Malo et al., 2006] Malo, J., Epifanio, I., Navarro, R., and Simoncelli, E. P. (2006). Nonlinear image representation for efficient perceptual coding. *IEEE Transactions on Image Processing*, 15(1):68–80. 98

[Martin et al., 2001] Martin, D., Fowlkes, C., Tal, D., and Malik, J. (2001). A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, page 416–423. 50, 56

[Mascarenhas, 2015] Mascarenhas, W. F. (2015). Robust padé approximants may have spurious poles. *Journal of Approximation Theory*, 189:76–80. 77

[Matsui et al., 2017] Matsui, Y., Ito, K., Aramaki, Y., Fujimoto, A., Ogawa, T., Yamasaki, T., and Aizawa, K. (2017). Sketch-based manga retrieval using manga109 dataset. *Multimedia Tools and Applications*, 76:21811–21838. 50, 56

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5:115–133. 1, 2, 3, 4

[Minnen et al., 2018] Minnen, D., Ballé, J., and Toderici, G. D. (2018). Joint autoregressive and hierarchical priors for learned image compression. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, page 10771–10780. Curran Associates, Inc. xvi, xvii, 25, 26, 65, 66, 67, 68, 70, 71

[Molina et al., 2020] Molina, A., Schramowski, P., and Kersting, K. (2020). Padé activation units: End-to-end learning of flexible activation functions in deep networks. In *The Eighth International Conference on Learning Representations (ICLR)*, volume 5, page 3201–3217. 5, 33, 77

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, page 807–814. 4

[NobelPrize.org, 2024] NobelPrize.org (2024). All nobel prizes. https://www.nobelprize.org/prizes/lists/all-nobel-prizes/. Accessed: 2024-11-19. 1

[Ouyang et al., 2022] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. (2022). Training language models to follow instructions

with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, page 27730–27744. Curran Associates, Inc. 1

[Padé, 1892] Padé, H. E. (1892). Sur la représentation approchée d'une fonction par des fractions rationnelles. In *Annales Scientifiques de l'École Normale Supérieure*, volume 9, page 3–93 (supplément). 31

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, page 8026–8037. Curran Associates, Inc. 40, 41, 77

[Petersen et al., 2022] Petersen, F., Borgelt, C., Kuehne, H., and Deussen, O. (2022). Deep differentiable logic gate networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, page 2006–2018. Curran Associates, Inc. 7

[Petersen et al., 2024] Petersen, F., Kuehne, H., Borgelt, C., Welzel, J., and Ermon, S. (2024). Convolutional differentiable logic gate networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, page 121185–121203. Curran Associates, Inc. 7

[Rissanen and Langdon, 1981] Rissanen, J. and Langdon, G. G. J. (1981). Universal modeling and coding. *IEEE Transactions on Information Theory*, 27(1):12–23. 25, 104

[Rombach et al., 2022] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 10684–10695. 1

[Rosenblatt, 1957] Rosenblatt, F. (1957). *The Perceptron, a Perceiving and Recognizing Automaton (Project Para)*. Cornell Aeronautical Laboratory, Inc., New York. 2

[Rosenblatt, 1961] Rosenblatt, F. (1961). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, DC. 2

[Ruiz et al., 2023] Ruiz, N., Li, Y., Jampani, V., Pritch, Y., Rubinstein, M., and Aberman, K. (2023). Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 22500–22510. 1

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536. 2

[Salimans and Kingma, 2016] Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 29, page 901–909. Curran Associates, Inc. 40

[Shi et al., 2016] Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D., and Wang, Z. (2016). Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 1874–1883. xviii, 18, 48, 97

[Shin and Ghosh, 1991] Shin, Y. and Ghosh, J. (1991). The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume 1, page 13–18. 5, 6

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukçuoğlu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489. 1

[Silver et al., 2017] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*. 1

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations (ICLR)*, volume 1, page 1–14. 27, 51, 57, 102

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958. 40

[Stahl, 1998] Stahl, H. (1998). Spurious poles in padé approximation. *Journal of Computational and Applied Mathematics*, 99(1–2):511–527. 77

[Szegedy et al., 2017] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, volume 31, page 4278–4284. 48

[Tillet et al., 2019] Tillet, P., Kung, H.-T., and Cox, D. (2019). Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings*

of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL), page 10–19. Association for Computing Machinery. 77

[Toderici et al., 2020] Toderici, G., Shi, W., Timofte, R., Theis, L., Balle, J., Agustsson, E., Johnston, N., and Mentzer, F. (2020). Workshop and challenge on learned image compression (clic2020). In *CVPR*. 67

[Transformer and Zhavoronkov, 2022] Transformer, C. G. P.-t. and Zhavoronkov, A. (2022). Rapamycin in the context of pascal's wager: Generative pre-trained transformer perspective. *Oncoscience*, 9:82–84. 1

[Trefethen and Gutknecht, 1987] Trefethen, L. N. and Gutknecht, M. H. (1987). *Algorithms for Approximation*, volume 10, chapter Padé, stable padé, and chebyshev-padé approximation, page 227–264. Clarendon Press. 77

[Tylavsky et al., 2022] Tylavsky, D., Li, S., and Shi, D. (2022). The padé matrix pencil method with spurious pole information assimilation. *arXiv preprint arXiv:2201.05208*. 77

[van den Oord et al., 2016] van den Oord, A., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukçuoğlu, K. (2016). Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems (NIPS)*, volume 29, page 4790–4798. Curran Associates, Inc. 71

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, volume 30, page 5998–6008. Curran Associates, Inc. 6, 40

[Voelker and Eliasmith, 2018] Voelker, A. R. and Eliasmith, C. (2018). Improving spiking dynamical networks: Accurate delays, higher-order synapses, and time cells. *Neural Computation*, 30(3):569–609. 7

[Volper and Hampson, 1990] Volper, D. J. and Hampson, S. E. (1990). Quadratic function nodes: Use, structure and training. *Neural Networks*, 3(1):93–107. 5, 10

[Waibel et al., 1989] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328–339. 2

[Wang et al., 2004] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612. xviii, 20, 51, 57, 100, 101

[Wang et al., 2003] Wang, Z., Simoncelli, E., and Bovik, A. (2003). Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, page 1398–1402. 104

[Wilson and Cowan, 1972] Wilson, H. R. and Cowan, J. D. (1972). Excitatory and inhibitory interactions in localized populations of model neurons. *Biophysical Journal*, 12(1):1–24. 4

[Woods et al., 1985] Woods, J., Biemond, J., and Tekalp, A. M. (1985). Boundary value problem in image restoration. In *ICASSP '85. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 10, page 692–695. 36, 38

[Xiao et al., 2018] Xiao, L., Zhang, H., Chen, W., Wang, Y., and Jin, Y. (2018). Transformable convolutional neural network for text classification. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, page 4496–4502. 76

[Xie et al., 2024] Xie, X., Zhou, P., Li, H., Lin, Z., and Yan, S. (2024). Adan: Adaptive nesterov momentum algorithm for faster optimizing deep models. *IEEE Trans-*

actions on Pattern Analysis and Machine Intelligence (TPAMI), 46(12):9508–9520. 50

[Xiong et al., 2024] Xiong, Y., Li, Z., Chen, Y., Wang, F., Zhu, X., Luo, J., Wang, W., Lu, T., Li, H., Qiao, Y., Lu, L., Zhou, J., and Dai, J. (2024). Efficient deformable convnets: Rethinking dynamic and sparse operator for vision applications. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 5652–5661. 76

[Xue et al., 2019] Xue, T., Chen, B., Wu, J., Wei, D., and Freeman, W. T. (2019). Video enhancement with task-oriented flow. *International Journal of Computer Vision*, 127:1106–1125. 26

[Yang et al., 2019] Yang, B., Bender, G., Le, Q. V., and Ngiam, J. (2019). Condconv: Conditionally parameterized convolutions for efficient inference. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, page 1307–1318. Curran Associates, Inc. 5, 40

[Yılmaz et al., 2021] Yılmaz, M. A., Keleş, O., Güven, H., Tekalp, A. M., Malik, J., and Kıranyaz, S. (2021). Self-organized variational autoencoders (self-vae) for learned image compression. In *2021 IEEE International Conference on Image Processing (ICIP)*, page 3732–3736. xi, xv, 6, 9, 15, 18, 25, 26, 27, 28, 29

[Zagoruyko and Komodakis, 2016] Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, page 87.1–87.12. BMVA Press. 48

[Zahid et al., 2023] Zahid, M. U., Kıranyaz, S., and Gabbouj, M. (2023). Global ecg classification by self-operational neural networks with feature injection. *IEEE Transactions on Biomedical Engineering*, 70(1):205–215. 15

[Zeyde et al., 2012] Zeyde, R., Elad, M., and Protter, M. (2012). On single image scale-up using sparse-representations. In *International Conference on Curves*

and Surfaces, Avignon, France, June 24-30, 2010, Revised Selected Papers 7, Lecture Notes in Computer Science, volume 6920, page 711–730. Springer Berlin Heidelberg. 50, 56

[Zhang et al., 2018] Zhang, R., Isola, P., Efros, A. A., Shechtman, E., and Wang, O. (2018). The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 586–595. xviii, 20, 51, 57, 102

[Zhou et al., 2021] Zhou, J., Jampani, V., Pi, Z., Liu, Q., and Yang, M.-H. (2021). Decoupled dynamic filter networks. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 6643–6652. 5

[Zhu et al., 2019] Zhu, X., Hu, H., Lin, S., and Dai, J. (2019). Deformable convnets v2: More deformable, better results. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 9300–9308. 76

Appendix A

RELATED LAYERS

A.1 Sub-Pixel Convolution Layer

The pixel shuffler or sub-pixel convolution layer is proposed for super-resolution models to increase the resolution of the input at the end of the network by an integer value [Shi et al., 2016]. The convolutional layer produces activation maps without non-linearity, and a rearrangement operation orders them to create the high-resolution image.

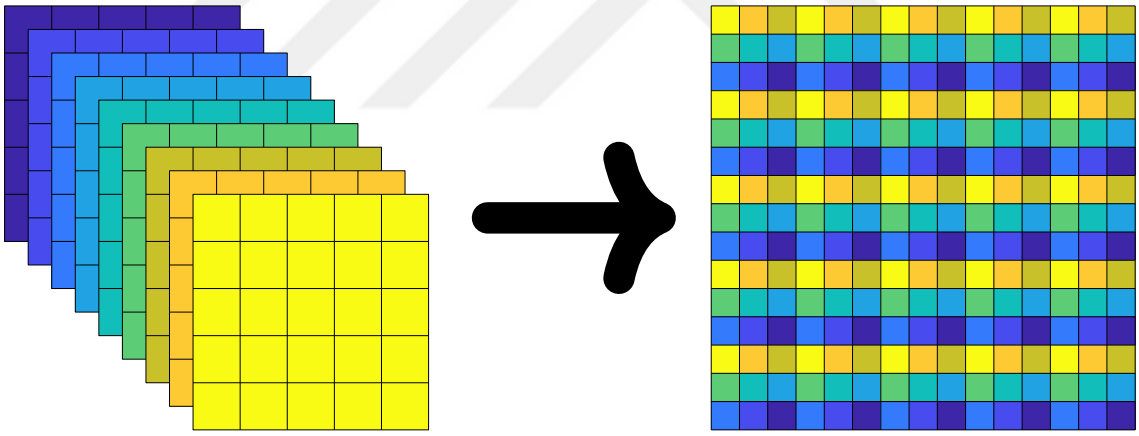


Figure A.1: Illustration of periodic shuffling for $r = 3$. Adapted from [Shi et al., 2016].

Specifically, for the increase ratio r, a convolutional layer having r^2 neurons produces r^2 activation maps from low-resolution feature maps. In terms of dimensions, the layer takes a tensor with dimensions $H \times W \times C$ and outputs one with $H \times W \times C \cdot r^2$. Later, the features are passed through the periodic shuffling operator. Each pixel on each activation map is rearranged by that operator in such a way that they are scattered periodically, and each $r \times r$ patch on the high-resolution output, whose

dimensions are $r \cdot H \times r \cdot W \times C$, has one feature element from each of r^2 activation maps. An illustration of the periodic shuffling is shown in Figure A.1.

A.2 Generalized Divisive Normalization (GDN)

Divisive normalization is proposed to control output gain by adjusting the values so that they are roughly limited to a desired range without loosing their relative magnitude [Malo et al., 2006], which can be formularized as

$$y_i = \frac{\text{sgn}(z_i)|z_i|^a}{\beta_i + \sum_j g_{ij}|z_i|^a} \quad \text{for} \quad \mathbf{z} = \mathbf{H}\mathbf{x}. \tag{A.1}$$

The generalized version, proposed by Ballé et al. [Ballé et al., 2016a], is defined as

$$\mathbf{y} = \text{GDN}(\mathbf{x}; \boldsymbol{\theta}) \tag{A.2}$$

where

$$y_i = \frac{z_i}{\left(\beta_i + \sum_j g_{ij}|z_j|^{a_{ij}}\right)^{\varepsilon_i}} \quad \text{for} \quad \mathbf{z} = \mathbf{H}\mathbf{x}. \tag{A.3}$$

Here, $\boldsymbol{\theta}$ contains all the parameters, which are the vectors $\boldsymbol{\beta}$ and $\boldsymbol{\varepsilon}$, and the matrices $\mathbf{H}$, $\mathbf{A} = [a_{ij}]$ and $\mathbf{G} = [g_{ij}]$. For the stable training, the following constraints are enforced:

$$a_{ij} \geq 1, \quad \beta_i > 0, \quad g_{ij} \geq 1, \quad 0 \leq \varepsilon_i \leq a_{ii}^{-1}. \tag{A.4}$$

The inverse GDN can simply be calculated as

$$z_i^{(n+1)} = \left(\beta_i + \sum_j g_{ij}|z_j^{(n)}|^{a_{ij}}\right)^{\varepsilon_i} y_i \quad \text{for} \quad z_i^{(0)} = \text{sgn}(y_i)\left(g_{ii}^{\varepsilon_i}|y_i|\right)^{\frac{1}{1-a_{ii}\varepsilon_i}} \tag{A.5}$$

In both GDN and inverse GDN layers, there are additional $2N + 3N^2$ parameters for input with N channels.

A.3 Gaussian Error Linear Unit (GELU)

GELU is proposed as a non-linearity in attempt to obtain an "activation function by combining of properties from dropout, zoneout and ReLUs" [Hendrycks and Gimpel,

2016]. Hendrycks and Gimpel try to combine their deterministic *or* stochastic masking strategy by merging their "functionality by multiplying the input by zero or one, but the values of this zero-one mask are stochastically determined while also dependent upon the input" [Hendrycks and Gimpel, 2016]. GELU is defined as

$$\text{GELU}(x) = x\Phi(x) = \frac{x}{2}\left[1 + \text{erf}(x/\sqrt{2})\right] = \frac{x}{2}\left[1 + \frac{2}{\sqrt{\pi}}\int_0^{x/\sqrt{2}} e^{-t^2} dt\right]. \quad \text{(A.6)}$$

Also, it can be approximated in two different formulations:

$$\text{GELU}(x) \approx 0.5x\left(1 + \tanh\left(\sqrt{2/\pi}\left(x + 0.044715x^3\right)\right)\right), \quad \text{(A.7)}$$

$$\text{GELU}(x) \approx x \cdot \sigma(1.702x), \quad \text{(A.8)}$$

where $\sigma(\cdot)$ is sigmoid in Equation (A.8) which is fast but a little bit inexact evaluation.

A.4 Attention Block

The use of attention blocks for the image compression task is proposed to increase the performance of the model [Cheng et al., 2020]. The proposed attention block contains six residual blocks, one independent 1×1 convolution, and one sigmoid layer to scale the values for the final multiplication. The residual block, right of Figure A.2, has two layers with kernels 1×1 and 3×3, whose number of neurons is half of the number of channels in input features, and ReLU activation that follows. The last 1×1 convolution increases the number of channels to the original value. Finally, the input and residual output are added at the end, followed by another ReLU activation. The block architecture is shown in Figure A.2.

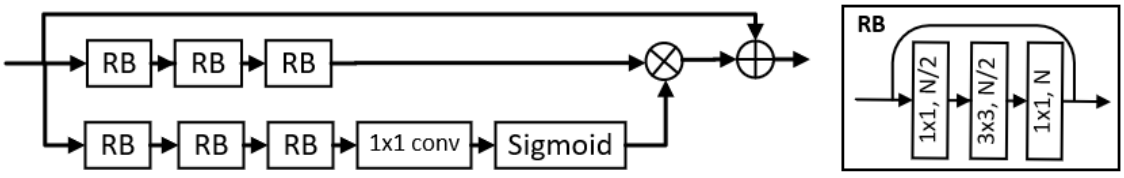


Figure A.2: Illustration of an attention block (left) and residual block (right) used also in ELIC architecture. Directly taken from [Cheng et al., 2020].

Appendix B

RELATED METRICS

B.1 Structural Similarity (SSIM)

SSIM is proposed to assess image quality in a referenced way based on the degradation of structural information assuming that the human vision system is very capable of perceiving the structural aspects of a scene [Wang et al., 2004]. It is a hand-crafted matric, and its calculation pipeline is shown in Figure B.1.

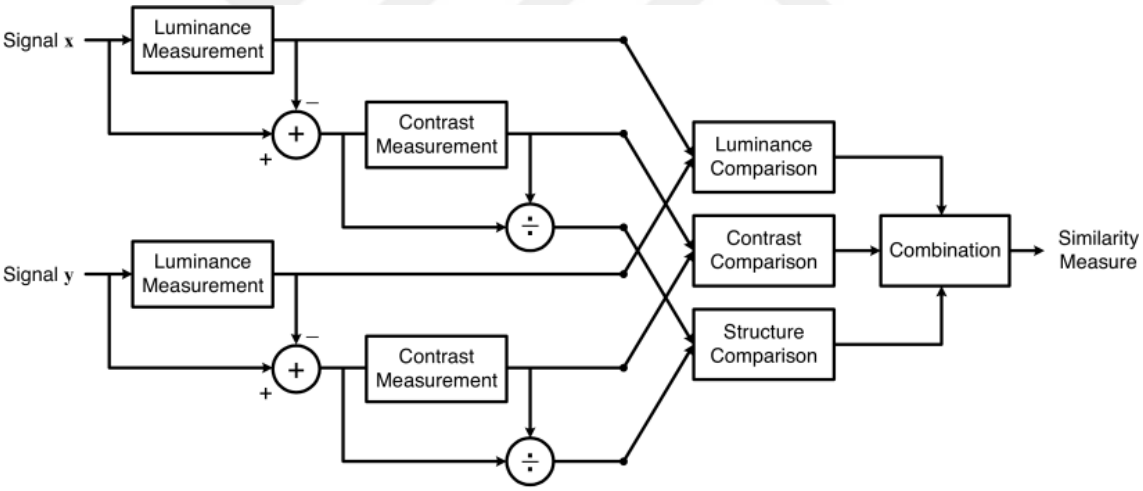


Figure B.1: SSIM calculation. Directly taken from [Wang et al., 2004].

For the gray-scale image x, the luminance is defined as mean of the signal:

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i. \tag{B.1}$$

Luminance comparison of two images, x and y, is performed by the equation

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}, \tag{B.2}$$

where C_1 is a constant to prevent numerical instability, which is defined as

$$C_1 = (K_1 L)^2, \tag{B.3}$$

where L is the dynamic range of the image, and $K_1 \ll 1$ is a small constant.

Unbiased estimation of standard deviation is used for the estimate of the signal contrast:

$$\sigma_x = \left(\frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu_x)^2 \right)^{1/2}. \tag{B.4}$$

Similar to luminance, the contrast comparison function $c(x, y)$ is defined as

$$c(x, y) = \frac{2\sigma_x \sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}, \tag{B.5}$$

where $C_2 = (K_2 L)^2$ for $K_2 \ll 1$ is again used for numerical stability.

Finally, the structure comparison function is defined as

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x \sigma_y + C_3}, \tag{B.6}$$

where C_3 is a small constant for stability, and σ_{xy} can be found by

$$\sigma_{xy} = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu_x)(y_i - \mu_y). \tag{B.7}$$

Structural similarity (SSIM) is defined as

$$\mathrm{SSIM}(x, y) = l^{\alpha}(x, y) \cdot c^{\beta}(x, y) \cdot s^{\gamma}(x, y) \tag{B.8}$$

for $\{\alpha, \beta, \gamma\} > 0$. Considering 8-bit gray-scale images ($L = 255$), choosing the exponents $\alpha = \beta = \gamma = 1$ and constants $C_3 = C_2/2$, Equation (B.8) simplifies to

$$\mathrm{SSIM}(x, y) = \frac{(2\mu_x \mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}. \tag{B.9}$$

The study [Wang et al., 2004] also proposes the use of 11×11 Gaussian kernels as a window function which has the effect of introducing weights w_i in front of the summed terms in Equations (B.1), (B.4) and (B.7), and $K_1 = 0.01$, $K_2 = 0.03$ for the dynamic range $L = 255$.

B.2 Learned Perceptual Image Patch Similarity (LPIPS)

LPIPS metric is proposed to rate images according to their similarity in terms of human perceptual judgments instead of their fidelity. Although PSNR and SSIM are frequently used as measures of image fidelity, they do not closely match human perception. Moreover, perceptual losses using features of some pre-trained networks [Dosovitskiy and Brox, 2016] give more visually pleasing results. Therefore, the study by Zhang et al. [Zhang et al., 2018] proposes a new image metric, called LPIPS, which measures human perceptual similarity based on deep network features.

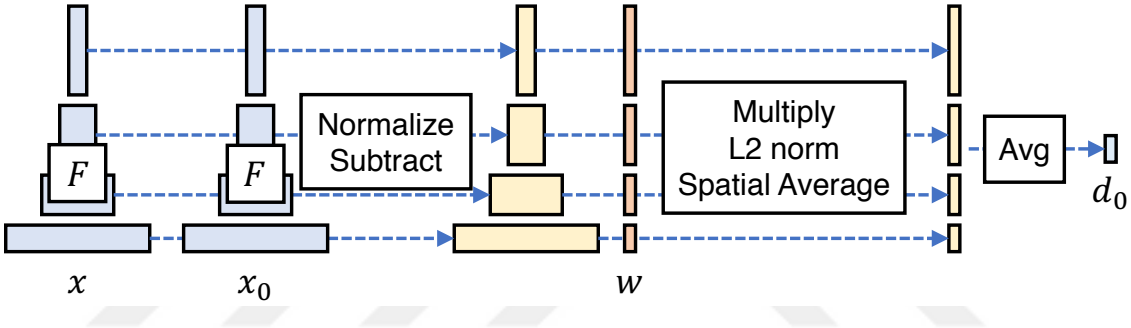


Figure B.2: LPIPS calculation. Directly taken from [Zhang et al., 2018].

The calculation steps are shown in Figure B.2. Two patches are passed through a network F, which may be AlexNet [Krizhevsky et al., 2012] or VGG-Net [Simonyan and Zisserman, 2015], and their deep embeddings are computed. The features are then normalized and scaled in channel dimension, and their l_2 distance is taken, which is averaged at the end.

B.3 Bjøntegaard Delta (BD)

The method is proposed by Bjøntegaard [Bjøntegaard, 2001] to calculate the average difference of rate-distortion curves of different image or video compression frameworks. Assuming that p denotes distortion and r denotes the bit rate for image or video, the original calculation follows the steps below:

- (p_i, r_i) points where $i = 1, 2, \ldots, N$ for $N \geq 4$ are needed for each rate-distortion

curve. In other words, at least 4 distortion-rate pairs are obtained for each method.

- The logarithm of the rates at the base 10 is calculated bringing rate to the same logarithmic scale as the distortion, which is often measured in dB.

- A curve passing through all N points is fitted for each framework using the formula:

$$y = f(x) = a + bx + cx^2 + dx^3, \tag{B.10}$$

where a, b, c and d are found so that the curve passes through all points.

- After fitting, an expression for the integral of each curve is computed.

- Then the average difference is defined by the difference between the integrals divided by the integration interval.

If the function in Equation (B.10) is a function of rate; in other words, $p = f(r)$, then the result $\bar{\Delta}_p$ is in dB and called BD-PSNR. If $r = f(p)$ and the difference is $\bar{\Delta}_r$, then the percentage of bit rate saving, BD-rate, is defined as $100 \cdot (10^{\bar{\Delta}_r} - 1)$.

There are some details that must be taken into account when calculating the BD values. The first is that the integration interval should not include any cross-over between the curves because the integration result cannot give the correct outcome to be interpreted for that situation. In that case, the integral must be calculated separately for the parts before and after the cross-over point. Second, the curves of different methods should cover a similar range in the integral variable. When this is not the case, the BD values do not give a correct idea about performance. Finally, although the interpolation function in Equation (B.10) uses cubic interpolation for polynomial fitting, there are other choices such as piecewise cubic Hermite interpolating polynomials [Fritsch and Carlson, 1980] and Akima interpolation [Akima, 1970, Herglotz et al., 2022]. For the correct result to be given by the integration, the interpolation methods should match.

Appendix C

RELATED LOSS FUNCTIONS

C.1 Rate-Distortion (RD) Loss

Rate-distortion (RD) loss is a function combination of two penalty functions to minimize both the bit rate of a compressed image and the total distortion it has after decompression at the same time. It can be formularized as

$$\mathcal{L} = \mathcal{L}_R + \lambda \mathcal{L}_D, \tag{C.1}$$

where $\mathcal{L}_R$ is rate loss, $\mathcal{L}_D$ is distortion loss, and λ is the Lagrange multiplier to balance the rate and distortion.

Starting from the simpler part, the distortion can be defined as l_1 loss, MSE, or multi-scale SSIM (MS-SSIM) [Wang et al., 2003]. In this thesis, MSE loss is chosen as the distortion loss since minimizing MSE guarantees the maximization of PSNR by definition [Keleş et al., 2021b]:

$$\mathcal{L}_D = \text{MSE}(\mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{i=1}^{N} (x_i - y_i)^2, \tag{C.2}$$

where N is the total number of elements in both $\mathbf{x}$ and $\mathbf{y}$.

The rate loss $\mathcal{L}_R$ penalizes the number of bits required for the reconstruction of the compressed image, which can be obtained by quantizing the latent representation of the image. Quantization is required by the arithmetic encoder [Rissanen and Langdon, 1981] since it operates on discrete values. However, quantization produces useless derivatives for model training via back-propagation. This operation can be approximated by adding uniform noise in the $[-0.5, 0.5]$ interval to the latent variables during training [Ballé et al., 2016b].

Assume that $p_{\boldsymbol{y}}$ denotes the fully factorized density model of the latent variable

$\boldsymbol{y}$ [Ballé et al., 2017]. In that case,

$$p_{\boldsymbol{y}}(\boldsymbol{y}) = \prod_{i}^{N} p_{y_i}(y_i), \tag{C.3}$$

where $y_i \in \boldsymbol{y}$, and N is the total number of elements in $\boldsymbol{y}$. If independent uniform noise $n \sim \mathcal{U}(-0.5, 0.5)$ is added to each of the variables such that

$$\tilde{y}_i = y_i + n, \tag{C.4}$$

then new distribution $p_{\tilde{\boldsymbol{y}}}$ becomes

$$p_{\tilde{\boldsymbol{y}}}(\tilde{\boldsymbol{y}}) = \prod_{i} \left(p_{y_i} * \mathcal{U}(-0.5, 0.5)\right)(y_i), \tag{C.5}$$

where $*$ denotes convolution. Since this is differentiable, the entropy can be used as the rate loss:

$$\mathcal{L}_{R_{\boldsymbol{y}}} = -\frac{1}{N} \log_2 \left(p_{\tilde{\boldsymbol{y}}}(\tilde{\boldsymbol{y}})\right) = -\frac{1}{N} \sum_{i=1}^{N} \log_2 \left(\left(p_{y_i} * \mathcal{U}(-0.5, 0.5)\right)(y_i)\right). \tag{C.6}$$

Since the study by Ballé et al. [Ballé et al., 2018], learned compression pipelines also send side information to modify the entropy model to reduce mismatch. Denoted as $\boldsymbol{z}$, this information is also sent after quantization. Thus the same procedure above can also be applied for them, creating the following loss term:

$$\mathcal{L}_{R_{\boldsymbol{z}}} = -\frac{1}{M} \log_2 \left(p_{\tilde{\boldsymbol{z}}}(\tilde{\boldsymbol{z}})\right) = -\frac{1}{M} \sum_{i=1}^{M} \log_2 \left(\left(p_{z_i} * \mathcal{U}(-0.5, 0.5)\right)(z_i)\right), \tag{C.7}$$

where M is the total number of elements in $\boldsymbol{z}$. Finally, the total rate loss becomes

$$\mathcal{L}_R = \mathcal{L}_{R_{\boldsymbol{y}}} + \mathcal{L}_{R_{\boldsymbol{z}}}. \tag{C.8}$$

For further and more mathematically involved explanation, see the study by Ballé et al. [Ballé et al., 2018].

C.2 A General Loss Function

Barron [Barron, 2019] proposes a general loss function that can also transform into other popular loss functions such as MSE loss or Charbonnier loss [Charbonnier

et al., 1994]. Its basic formulation is as follows:

$$f(x, \alpha, c) = \frac{|\alpha - 2|}{\alpha} \left(\left(\frac{(x/c)^2}{|\alpha - 2|} + 1 \right)^{\alpha/2} - 1 \right) \tag{C.9}$$

where $\alpha \in \mathbb{R}$ is a parameter that controls the shape, and $c > 0$ is a variable controlling the "bowl" size for x values close to 0. Since this formulation causes numerical problems in computation for $\alpha = \{-\infty, 0, 2, \infty\}$, the results of the limiting behavior of the function are used in these cases. After that, the final form of the loss becomes the following.

$$f(x, \alpha, c) = \begin{cases} 1 - \exp\left(-\frac{1}{2}(x/c)^2\right) & \text{when } \alpha = -\infty \\ \log\left(\frac{1}{2}(x/c)^2 + 1\right) & \text{when } \alpha = 0 \\ \frac{1}{2}(x/c)^2 & \text{when } \alpha = 2 \\ \exp\left(-\frac{1}{2}(x/c)^2\right) - 1 & \text{when } \alpha = \infty \\ \frac{|\alpha - 2|}{\alpha} \left(\left(\frac{(x/c)^2}{|\alpha - 2|} + 1 \right)^{\alpha/2} - 1 \right) & \text{otherwise} \end{cases} \tag{C.10}$$

The plots of the function in Equation (C.10) and its derivative are given in Figure C.1 for $\alpha = 1.5$ and $c = 2$. For further information, please see the work by Barron [Barron, 2019].

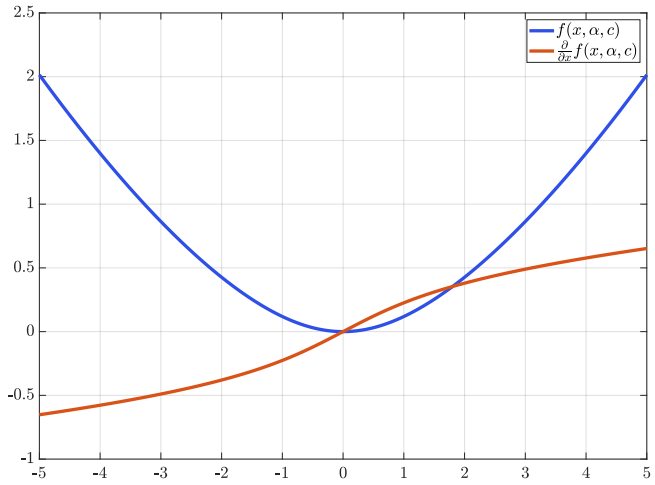


Figure C.1: Loss function and its derivatives for $\alpha = 1.5$ and $c = 2$.

Appendix D

DETAILS FOR COMPLEXITY CALCULATIONS

For `fvcore` results in Table 4.1, the following function is used to calculate the number of MACs for all layers.

```
import torch
from fvcore.nn import FlopCountAnalysis

def fvcore_analysis(layer_type, layer, inp):
    print("")
    print("================================================")
    print(f"fvcore.nn for {layer_type}")
    print("================================================")
    print("")

    torch.cuda.empty_cache()
    inp = inp.float().to("cuda")
    layer = layer.float().to("cuda")

    torch.cuda.memory._record_memory_history()
    flops = FlopCountAnalysis(layer, inp).total()
    torch.cuda.memory._dump_snapshot(f"{layer_type}_fvcore.pickle")
    print("{:>16s} : {:<.4f}".format("MACs", flops))

    num_parameters = sum(map(lambda x: x.numel(), layer.parameters()))
    print("{:>16s} : {:<.4f}".format("#Params", num_parameters))
```

For `ptflops` results in Table 4.1, the following function is used to calculate the number of MACs for all layers.

```
import torch
from ptflops import get_model_complexity_info

def ptflops_analysis(layer_type, layer, inp):
    print("")
    print("================================================")
    print(f"ptflops for {layer_type}")
    print("================================================")
    print("")

    torch.cuda.empty_cache()
    with torch.cuda.device(0):
        net = layer.float()
        torch.cuda.memory._record_memory_history()
        macs, params = get_model_complexity_info(
            net, tuple(inp.shape[1:]), as_strings=True, backend='pytorch', print_per_layer_stat=True,
            verbose=True, flops_units="MAC"
        )
        torch.cuda.memory._dump_snapshot(f"{layer_type}_ptflops_pytorch.pickle")
        print('{:<30}  {:<8}'.format('Computational complexity: ', macs))
```

```
            print('{:<30}  {:<8}'.format('Number of parameters: ', params))

            torch.cuda.memory._record_memory_history()
            macs, params = get_model_complexity_info(
                net, tuple(inp.shape[1:]), as_strings=True, backend='aten', print_per_layer_stat=True,
                verbose=True, flops_units="MAC"
            )
            torch.cuda.memory._dump_snapshot(f"{layer_type}_ptflops_aten.pickle")
            print('{:<30}  {:<8}'.format('Computational complexity: ', macs))
            print('{:<30}  {:<8}'.format('Number of parameters: ', params))
```

For `thop` results in Table 4.1, the following function is used to calculate the number of MACs for all layers.

```
import torch
from thop import profile

def thop_analysis(layer_type, layer, inp):
    print("")
    print("============================================")
    print(f"thop for {layer_type}")
    print("============================================")
    print("")

    torch.cuda.empty_cache()
    inp = inp.float().to("cuda")
    layer = layer.float().to("cuda")

    torch.cuda.memory._record_memory_history()
    macs, params = profile(layer, inputs=(inp, ), verbose=True, report_missing=True)
    torch.cuda.memory._dump_snapshot(f"{layer_type}_torchprofile.pickle")
    print(f"MACs: {macs}")
    print(f"Parameters: {params}")
```

For `torchprofile` results in Table 4.1, the following function is used to calculate the number of MACs for all layers.

```
import torch
from torchprofile import profile_macs

def torchprofile_analysis(layer_type, layer, inp):
    print("")
    print("============================================")
    print(f"torchprofile for {layer_type}")
    print("============================================")
    print("")

    torch.cuda.empty_cache()
    inp = inp.float().to("cuda")
    layer = layer.float().to("cuda")

    torch.cuda.memory._record_memory_history()
    macs = profile_macs(layer, inp)
    torch.cuda.memory._dump_snapshot(f"{layer_type}_torchprofile.pickle")
    print(f"MACs: {macs}")
```

For `flop_counter` results in Table 4.1, the following function is used to calculate the number of FLOPS for all layers.

```
import torch
from torch.utils.flop_counter import FlopCounterMode

def torch_flop_counter_analysis(layer_type, layer, inp):
    print("")
    print("================================================")
    print(f"torch.utils.flop_counter.FlopCounterMode for {layer_type}")
    print("================================================")
    print("")

    torch.cuda.empty_cache()
    inp = inp.float().to("cuda")
    layer = layer.float().to("cuda")

    torch.cuda.memory._record_memory_history()
    with FlopCounterMode(display=True, depth=None) as flop_counter:
    out = layer(inp)
    torch.cuda.memory._dump_snapshot(f"{layer_type}_torch_flop_counter.pickle")
    print(f"Total FLOPs -> {flop_counter.get_total_flops()}")
```

For the time results in Table 4.2, the following function is used for all layers.

```
import torch
from torch.profiler import profile as profile_th, record_function, ProfilerActivity

def torch_profiler_analysis(layer_type, layer, inp):
    print("")
    print("================================================")
    print(f"torch.profiler for {layer_type}")
    print("================================================")
    print("")

    torch.cuda.empty_cache()
    inp = inp.float().to("cuda")
    layer = layer.float().to("cuda")

    with profile_th(activities=[ProfilerActivity.CUDA], record_shapes=True) as prof:
        with record_function("forward"):
            torch.cuda.memory._record_memory_history()
            layer(inp)
            torch.cuda.memory._dump_snapshot(f"{layer_type}_torch_profiler.pickle")

    print(prof.key_averages().table(sort_by="cuda_time_total", row_limit=50))
```

The memory results in Table 4.2 are obtained by opening the `pickle` file that each given function creates in https://pytorch.org/memory_viz.