Generative Modeling and Neural Compression

Dissertation Committee

promotoren:	prof. dr. S. Bhulai	Vrije Universiteit Amsterdam
	prof. dr. R.D. van der Mei	Centrum Wiskunde & Informatica
promotiecommissie:	prof. dr. G. Koole	Vrije Universiteit Amsterdam
	prof. dr. O. Winther	Technical University of Denmark
	dr. E. Bekkers	University of Amsterdam
	dr. E. Gavves	University of Amsterdam
	dr. F. Huot	Google DeepMind

ISBN: xxxxx

Typeset by LaTeX.
Printed by: TBD
Cover design by: Stable Diffusion

VRIJE UNIVERSITEIT

Generative Modeling and Neural Compression

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Bètawetenschappen
op [defenseDay] [DATE] om [TIME xx.xx] uur
in [de aula / het auditorium] van de universiteit,
De Boelelaan 1105

door

Yura Mercedes Perugachi Diaz

geboren te Amsterdam, Nederland

promotoren: prof.dr. S. Bhulai
prof.dr. R.D. van der Mei

ACKNOWLEDGEMENTS

Contents

List of Abbreviations

AE	Autoencoder
AI	Artificial Intelligence
AUC	Area Under the Curve
BPP	Bits per Pixel
CLipSwish	Concatenated LipSwish
CNN	Convolutional Neural Network
DenseBlocks	Densly Connected Blocks
DenseNet	Densly Connected Convolutional Network
ELBO	Evidence Lower Bound
FPR	False Positive Rate
FPS	frames-per-second
i-DenseNet	invertible Dense Network
KL	Kullback-Leibler
LC	Learnable weighted Concatenation
LR	Logistic Regression
MLP	Multi-Layered Perceptron
MLR	Multinomial Logistic Regression
MOS	Mean Opinion Scores

MR	Multirate
MSE	Mean Squared Error
PSNR	Peak Signal-to-Noise Ratio
R-D	Rate-Distortion
ResNet	Residual Network
ROC	Receiver Operating Characteristic
ROI	Regions-Of-Interests
SGA	Stochastic Gradient Gumbel Annealing
SGD	Stochastic Gradient Descent
SSF	Scale Space Flow
SSL	Sigmoid Scaled Logit
STE	Straight-Through Estimator
TPR	True Positive Rate
VAE	Variational Autoencoder

1

INTRODUCTION

Nowadays, deep learning is a popular topic. From face recognition to chatGPT to photo editing to text-to-image, deep learning has become increasingly important in our daily lives, and a lot of research has been conducted in this field. However, as with everything in life, deep learning comes with a package of costs and limitations. The amount of data and computer power it takes to train models is significant. Besides hardware issues, improving these models may raise even bigger concerns, such as the societal impact. What happens when deep learning is used for improper purposes? Some serious side effects we will point out at the end of almost every chapter.

In this thesis, we will explore a variety of deep learning models and their applications. This section will provide background information and notations, and we will show how the topics are intertwined. In Section 1.1, we give brief background information about discriminative modeling and explain what the aim of deep learning is to improve the business process for businesses operating in agriculture. Section 1.2 explains a complex open problem in generative modeling and introduces a general structure of how two generative models, namely, a normalizing flow and variational autencoder, learn the density distribution of data. Note that our proposed generative model, i-DenseNets, originates from the normalizing flow. Additionally, the variational autoencoder forms the base idea behind neural compression models. Section 1.3 demonstrates how the variational autoencoder inspires neural compression models and what similarities they have in common. We continue with a brief overview of this thesis, including some challenges that are faced. Finally, we illustrate the ideas behind the images that are created for this thesis.

1.1 PART I: DISCRIMINATIVE MODELS

In the early 1980s, back-propagation networks [80] were invented. The predecessor of what we now know as *convolutional neural networks* (CNNs). But it was not until late 2012 for a big breakthrough that brought convolutions to life. In [74], the AlexNet CNN was introduced that won the ImageNet challenge [114] by achieving record-breaking results. For the ImageNet challenge, participants are asked to label a dataset that consists of millions of high-resolution images. This may be with a technique of the participant's choice. Solving the ImageNet challenge with discriminative modeling requires a model to classify a high-dimensional input image. The images belong to a certain class, e.g., 'racket', 'lion', or 'candle'. Specifically, deep learning

models, implemented as a discriminative model, model the following conditional distribution:

1

$$p(y|\mathbf{x}), \tag{1.1}$$

where y and $\mathbf{x}$ are the random variables representing the data distribution. From here on, we let $\mathbf{x} \in \mathbb{R}^d$ be a high dimensional image with dimensionality d. In computers, a pixel is typically stored in uint8. Therefore, a pixel value is $x \in \{0, 1, \ldots, 255\}$. Hence, y represents the corresponding label of an image. After the challenge, the field developed at an accelerated pace, introducing new CNNs [56, 60, 118] and breathed new life into computer vision.

Nowadays, CNNs are widely deployed for computer vision purposes to solve tasks such as image classification, image segmentation, or object detection. They are utilized in different domains to solve real-world problems. One of these domains is agriculture. With a growing world population, food production needs to increase significantly in the future. Additionally, deep learning and artificial intelligence (AI) are becoming increasingly popular for decision-making. The combination of AI and the agrifood industry is essential to realize the goals for food production and to improve business processes. An example of such improvement is the automation of time-consuming jobs such as detecting diseases in plant leaf images [41] or identifying plant species [83].

Part I introduces a discriminative modeling task to improve the business process of a seed breeding corporation operating in agriculture. Seed breeding ensures the development of seeds for climate-changing conditions and to preserve the diversity of the seeds. The aim of seed breeders is to deliver high-quality seeds to their customers and stay competitive with the market. An important aspect of seed breeders is their ability to make early predictions on the growth success of seedlings kept in a growth chamber, which saves time and space for other seedlings. More specifically, for this research, we classify the (un)successfulness of white cabbage seedlings. In Chapter 3, we will elaborate on various machine learning methods, including five different CNN types. We demonstrate how to deploy machine learning methods and how they aid the prediction of the successfulness of white cabbage seedlings based on only an image. We show that the growth success of white cabbage seedlings can be best predicted with CNNs, and from a collection of famous CNNs, AlexNet achieves the best performance. With an accuracy of 94% AlexNet could be deployed as an early warning tool to aid professionals in making important decisions.

Modeling discriminative tasks with deep learning is a successful and popular method to predict labels from data. However, when these models are trained on specific data, in practice, the models only achieve good performance on trained data but fail on never-before-seen images.

1.2 PART II: GENERATIVE MODELING

Although deep learning classifiers are powerful, they are known to fail on never-before-seen images, so-called *outliers*. A complex open problem in machine learning

is the estimation of a distribution for high-dimensional data, such as images. Deep learning models that try to estimate an unknown distribution are called *generative models*. Generative models are known to the public for their ability to generate realistic-looking images, generate text, or nowadays are even capable of generating images from a text sentence. They are getting more and more integrated into our daily lives and aiding users in various ways. They are utilized in applications such as face recognition, photo editing, ChatGPT, GitHub Copilot, etc. Despite their many successes, these models are still in progress and do not fully solve the problem of estimating the true data distribution of high-dimensional data; rather, they estimate it. Resolving the high dimensional density estimation problem is key to assessing, for example, the perfect generation of all different types of data. This may be drug discovery or simulation of complex environmental processes. Additionally, these models may even estimate uncertainty, making it appealing for anomaly detection. Challenges lie in the difficulty and instability of training these models.

In deep learning, there are many types of generative models that try to model the true probability distribution of high-dimensional image data:

$$p(\mathbf{x}), \tag{1.2}$$

where $p(\cdot)$ is the probability density function and $\mathbf{x}$ is the high dimensional unknown image data distribution. Famous, well-known examples that try to estimate the unknown data distribution are generative adversarial networks [46], variational autoencoders (VAEs) [67], normalizing flows [108] or diffusion models [113].

Generative models, such as VAEs and normalizing flows, use a latent variable model to learn $p(\mathbf{x})$. Therefore, let $\mathbf{z} \in \mathbb{R}^{d_z}$, be a latent variable with dimensionality d_z. The base probability density distribution for both models: $p(\mathbf{z})$, is usually modeled as a standard Gaussian distribution: $\mathcal{N}(0, I)$ and known as the *latent space*. After a model is fully trained, which we will explain later on, an unlimited number of samples can be drawn from its latent space. Sampling follows the following procedure. First, sample:

$$\mathbf{z} \sim p(\mathbf{z}), \tag{1.3}$$

where $\mathbf{z}$ represents a sample drawn from the standard Gaussian distribution. To sample a reconstructed image $\mathbf{x}$, we use the following conditional distribution:

$$\mathbf{x} \sim p(\mathbf{x}|\mathbf{z}). \tag{1.4}$$

VAEs and normalizing flows are optimized differently. A VAE is a probabilistic model that consists of an encoder, decoder, and latent space. Typically, we let $\mathbf{z} \in \mathbb{R}^{d_z}$ be a lower dimensional latent space $d_z < d$ than the dimensionality of a high dimensional image. The encoder does not directly model $p(\mathbf{z}|\mathbf{x})$, as this results in an intractable computation. Therefore, variational inference is used to approximate $p(\mathbf{z}|\mathbf{x})$ with $q(\mathbf{z}|\mathbf{x})$ [67]. Variational inference approximates complex distributions by measuring the distance between two distributions with the Kullback-Leibler (KL) divergence,

as this is computationally tractable. $q(\mathbf{z}|\mathbf{x})$ is typically modeled by a deterministic Gaussian distribution. The KL divergence consists of a trade-off that consists of maximizing the evidence lower bound (ELBO) as it cannot be minimized exactly. The probabilistic decoder $p(\mathbf{x}|\mathbf{z})$ of a VAE can be modeled by e.g., a categorical distribution. Where $p(\mathbf{x}|\mathbf{z}) = \text{Categorical}(\mathbf{x}|f(\mathbf{z}))$, here $f(\mathbf{z})$ are the probabilities for the categorical distribution and $f(\cdot)$ is modeled by a neural net. The expected log-likelihood is then given by:

$$\mathcal{L}_{VAE} = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\Big[\underbrace{\log p(\mathbf{x}|\mathbf{z})}_{\text{decoder}}\Big] - \underbrace{KL\big(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big)}_{\text{encoder bottleneck}}, \tag{1.5}$$

where the *decoder* part measures the likelihood of $\mathbf{x}$ given $\mathbf{z}$. The *encoder bottleneck* part counts the bits per dimension.

Normalizing flows model $p(\mathbf{x})$ a bit differently than VAEs, but consist of a similar structure. Hence, now we let $d_z = d$, be exactly the dimensionality of an image. Furthermore, whereas VAEs use variational inference to approximate $p(\mathbf{z}|\mathbf{x})$, normalizing flows model this directly by using the change-of-variables formula. Therefore, they use a bijective function defined as $f : \mathbb{R}^d \to \mathbb{R}^d$. As f is invertible, its inverse is defined as $F = f^{-1}$. Now we can map an image directly to the latent space $\mathbf{z} = F(\mathbf{x})$, sample a latent variable $\mathbf{z} \sim p(\mathbf{z})$ and model a reconstructed image by modeling $\mathbf{x} = f(\mathbf{z})$. The log-likelihood is then given by:

$$\log p(\mathbf{x}) = \log p(\mathbf{z}) + \log|\det J_F(\mathbf{x})|\,, \tag{1.6}$$

where J_F is the Jacobian of F at point $\mathbf{x}$. A normalizing flow computes the exact log-likelihood and therefore seems very attractive. However, there are downsides, such as, the latent space that is as high dimensional as the image itself and computation of the Jacobian determinant is computationally very expensive.

In Part II, we will explore a special type of normalizing flow that aids the high-dimensional density estimation problem. In Chapter 4, we extend an already existing model known as Residual flows [11, 24] and introduce a new version called i-DenseNets. Deep learning architectures based on DenseNets are very successful in supervised learning tasks. However, it was unknown how to integrate this for a normalizing flow, which is constrained to Lipschitz. Therefore, we derive the mathematical conditions for DenseNets to be invertible, which allows them to be used as a generative model. Note that Chapter 4 consists of two papers merged together, where the first paper [4] uses smaller i-DenseNet architectures to experiment and explore the model capabilities. After its success, we further explored i-DenseNet in [5] and built upon this work by introducing a new activation function, which also integrates a concatenation that is constrained by Lipschitz continuity. Finally, we show how these models are not only successful in generating new images compared to their predecessors, but also can be successfully deployed as hybrid models that can be both generative and discriminative models.

As generative models demonstrate to aid in closing the gap in estimating the true data distribution, they also draw inspiration from neural compression models.

1.3 PART III: NEURAL COMPRESSION

With the growing amount of data worldwide, compression plays a fundamental part in data storage and transmission. The aim of compression is to optimize digital content such as cloud and streaming services. Besides being successful in discriminative modeling or estimating high-dimensional data distributions, deep learning can also be utilized to compress data. In fact, there is even a strong connection between generative modeling with VAEs [67] and neural image compression models, such as the mean-scale hyperprior [8, 9, 95, 145]. VAEs and compression models both have an encoder-decoder-like structure. Yet, instead of modeling the encoder stochastically as the VAE does, image compression models, model the encoder deterministically. On a high level, this simplifies the learning objective of the VAE in Equation (1.5) to:

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\Big[\log p(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z}) - \overbrace{\log q(\mathbf{z}|\mathbf{x})}^{=0}\Big] \tag{1.7}$$

$$= \underbrace{\log p(\mathbf{x}|E(\mathbf{x}))}_{\text{likelihood}} + \underbrace{\log p(E(\mathbf{x}))}_{\text{rate}}, \tag{1.8}$$

where $E(\cdot)$ is a discrete encoder and:

$$q(\mathbf{z}|\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{z} = E(\mathbf{x}) \\ 0, & \text{otherwise.} \end{cases} \tag{1.9}$$

Compression loss is typically defined as:

$$\mathcal{L}_{\text{compression}} = \underbrace{d\left(\mathbf{x}, D(E(\mathbf{x}))\right)}_{\text{distortion}} - \lambda \underbrace{\log p(E(\mathbf{x}))}_{\text{rate}}, \tag{1.10}$$

where $D(\cdot)$ is the decoder, $d(\cdot)$ is a distance metric and λ is a parameter that determines the rate-distortion trade-off. The *likelihood* in Equation (1.8) and *distortion* in Equation (1.10), have common similarities as they both measure the distance between the approximation using latent variable $\mathbf{z}$ and the true data. The *rate* in Equation (1.8) and in Equation (1.10) are the same. These count the bits it takes to encode an image. For the compression loss, there is an extra parameter λ added to measure the distortion. Additionally, compression models use an entropy model for: $p_Z(E(\mathbf{x}))$, which is the base distribution. In VAEs, the latent space $p_Z(\mathbf{z})$ is also modeled with a base distribution. Furthermore, image compression models, such as the mean-scale hyperprior, form the base of the video compression model: scale-space flow [2].

Lastly, Part III introduces the compression topic. Compression comprises the process of sending data, such as images or videos, from user to user. An important aspect is to compress data in such a way that it maintains good data quality (little distortion) and the transfer costs are as little as possible (low rate) after the compression process. While there are several models deployed for image compression, these models break for video since video contains an extra dimension that breaks the structure and results in a high dimensional problem. Chapter 5 uses a frequently

1

used neural image compression model known as the mean-scale hyperprior [8, 9, 95]. We show how to only further optimize the latent of pre-trained mean-scale hyperpriors to improve the compression performance without the need to re-train the entire network. This method is especially useful when a limited computational budget is available. In Chapter 6, we develop two neural video compression models, based on the scale-space flow [2], that focus on regions of interest (ROIs) of the user. The advantage of these models is that they compress ROIs with higher accuracy than non-ROIs. As a result, this makes transferring videos more efficient while maintaining high overall quality.

OVERVIEW

Recall that this thesis is partitioned into three parts. In each part, we aid the deep learning developments by extending existing models and explore the capabilities. Each part provides its own qualities and characteristics and is sectioned as follows:

- Part I provides a practical business application of deep learning for a corporation operating in agriculture. In Chapter 3, we aim to predict white cabbage seedling images with machine learning algorithms. This is a hands-on approach where we examine how these algorithms behave and how they can be deployed in practice.
- Part II covers a more theoretical topic of generative modeling. In Chapter 4, we improve the density estimation performance of a generative model. Therefore, we provide the necessary theory on how to derive Lipschitz bounds for our proposed model, i-DenseNet.
- Part III is all about neural compression. In this part, we show large-scale applications of deep neural nets to aid neural image and video compression models. In Chapter 5, we propose a method on how to improve the compression performance of a frequently used neural image compression model without the need to re-train an entire pre-trained network. Finally, Chapter 6 demonstrates the first neural video compression model that is able to compress ROIs with more visual quality than non-ROIs.

Besides the impressive range of applications deep learning is developed for, it also comes with limitations and faces challenges. In general, these models require massive amounts of high-quality data. For example, image data for discriminative, generative and neural compression models, require to contain little artifacts. Additionally, each model requires to take specific and different pre-processing steps to create a stable model. Therefore, it may be more straightforward in some cases to use traditional methods. For instance, if there were only 100 images of white cabbage seedlings for research, it may be more effective to train a logistic regression model than a deep neural net. Since a deep neural net performs poorly when trained with little data. Secondly, these networks lack interpretability or explanation, making it not always useful for companies to explain their output to their customers. Lastly, especially for generative modeling, we experienced that these models require a significant amount of computing resources, which is not always available. Therefore,

focusing on more efficient architectures or smaller architectures with comparable performance as larger models is crucial for the implementation of deep learning on devices. Besides, this makes it more accessible for research.

THESIS IMAGES

In a world where technology is developing rapidly, many of us probably use AI in our digital lives, albeit without even knowing it. Starting this thesis, chatGPT and generating images from text were not publicly born yet. Hence, with the rise of chatGPT, I greedily take part in it. My life became easier in many ways, for inspiration on how to write a sentence differently, saving time on technical coding problems, or even fixing LaTeX issues. In return, this improved the speed of my thesis but also came with a laziness cost, to say the least. Therefore, it probably comes as no surprise that the images for the cover and parts in my thesis are generated with a stable diffusion model. Every generated image in this thesis represents the topics, combined with a hint of my own artistic expression.

COVER

PART I - IMPROVING BUSINESS PROCESSES The image in this part shows generated seedlings on a computer. The seedlings look a lot like the real white cabbage seedlings from the research.

PART II - HIGH DIMENSIONAL DENSITY ESTIMATION The second part of the thesis shows a child painting. The image represents newborn generative models as their existence dates back to only a few years ago. Nowadays, these models learn to generate data, which often comes with limitations and boundaries, as you will experience when working with them. Just like children, they are quickly developing and learning, but they are not there yet.

PART III - COMPRESSION In general, the application of neural image and video compression models is wide-ranging. From uploading images on webpages to compressing video games. The image in the final part represents a videogame-like environment with a blurry background and pixelated figures in the foreground.

1

2

PUBLICATIONS

This thesis is based on the following publications:

[1] Y. Perugachi-Diaz, A. Gansekoele and S. Bhulai. 'Robustly overfitting latents for flexible neural image compression'. In: *arXiv preprint arXiv:2401.17789* (2024).

[2] Y. Perugachi-Diaz, G. Sautière, D. Abati, Y. Yang, A. Habibian and T. S. Cohen. 'Region-of-Interest Based Neural Video Compression'. In: *Proceedings of the 33rd British Machine Vision Conference, BMVC, London, UK, November 21-24*. 2022, pages 1–21.

[3] Y. Perugachi-Diaz, J. M. Tomczak and S. Bhulai. 'Deep learning for white cabbage seedling prediction'. In: *Computers and Electronics in Agriculture* 184 (2021), pages 1–9.

[4] Y. Perugachi-Diaz, J. M. Tomczak and S. Bhulai. 'Invertible densenets'. In: *Proceedings of the 3rd Symposium on Advances in Approximate Bayesian Inference, AABI, January-February, virtual*. 2021.

[5] Y. Perugachi-Diaz, J. M. Tomczak and S. Bhulai. 'Invertible DenseNets with Concatenated LipSwish'. In: *Proceedings of Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems, NeurIPS, December 6-14, virtual*. 2021, pages 17246–17257.

2

Part I

IMPROVING BUSINESS PROCESSES

3

DEEP LEARNING FOR WHITE CABBAGE SEEDLING PREDICTIONS

Abstract

In this research, we classify white cabbage seedling images with machine learning to improve the business process of a seed breeding company operating in agriculture. We deploy several algorithms that track the seedling growth over a period of four days. The models are trained to predict the (un)successful growth of the seedlings. We find that convolutional neural networks outperform the other models, with AlexNet being the best-performing model. On the test set, AlexNet classifies 94% of the seedlings accurately with an area under the curve of 0.95. Additionally, this model can be further deployed to automate the seedling grading process as an early warning tool to aid professionals in making important decisions.

Based on [3]:
Yura Perugachi-Diaz, Jakub M. Tomczak, Sandjai Bhulai
Deep learning for white cabbage seedling prediction
Elsevier, Computers and Electronics in Agriculture,
Volume 184 (2021), pages 1–9.

3.1 INTRODUCTION

Recently, deep learning has become an increasingly popular method to solve problems in areas such as image classification, speech recognition, and video analysis [79]. A big advantage of deep learning is the ability to process unstructured data (e.g., images), without the need for feature extraction methods. In particular, the Convolutional Neural Networks (CNNs) are known for their ability to recognize patterns in images.

In agriculture, an important factor of crop yield is determined by the growth of plants. In practice, part of the plants never emerge from the ground, or will never mature. This will eventually result in yield loss for farmers and food loss for the consumer. However, professionally analyzing plant growth is a time-consuming job. A growing body of work aims to automate agricultural processes using deep learning, see, e.g., Lee et al. [83] identify plant species, Ferentinos [41] detects diseases in plant leaves, and Zhang et al. [148] identify agriculture machinery.

In this chapter, we study *white cabbage* (Brassica Oleracea) seedlings. The seedlings are tracked during their growth in a phytotron in which the temperature and amount of light are regulated. Their growth is captured at four specific moments in time. Furthermore, after a period of 14 days, the resulting plants are evaluated by professionals. Our goal is to identify which seedlings will grow successfully as early as possible. In order to predict the outcome, we propose using CNNs. We experimentally show that CNNs outperform traditional methods compared on both loss and accuracy. Further, we find that as seedlings mature, growth success can be better predicted.

The main contributions of this chapter are: (1) *Predicting if a white cabbage seedling is going to grow successfully* with high accuracy. (2) *Analyzing how time influences the growth predictability of seedlings.* (3) *Comparing traditional methods with deep learning models.* This research shows that it is possible to classify white cabbage seedlings over time with around 94% accuracy. The CNN outperforms the Multi-Layered Perceptron and Logistic Regression model.

The chapter is organized as follows; Section 3.2 gives an overview of literature performing similar research. Section 3.3 describes the data, shows the chosen models, implementation, and how the models were trained. Section 3.4 describes the experiments and provides an analysis of the results. Finally, Section 3.5 provides a conclusion with further research for the development of image recognition for seedling growth.

3.2 RELATED WORK

With the rise of deep learning, image analysis has become increasingly popular in agriculture. More researchers are employing technologies and methods by integrating deep learning to improve and automate work [150]. Besides deep learning, traditional statistical methods are tried and tested for crop prediction or disease prediction. Prabhakar et al. [107] examined Yellow Mosaic disease on eight blackgram

plant leaves using a Multinomial Logistic Regression (MLR) model. Kalisz et al. [66] examined multiple regression to assess the yield and nutrient of Chinese cabbages Taisai and Pak Choy White.

Except for traditional statistical methods, the use of deep learning models has started to become increasingly popular. Lee et al. [83] examined the identification of plant leaves. Kumar et al. [75] benchmarked hand-crafted leaf images against raw leaf images using the pre-trained network from Krizhevsky et al. [74]. In Ferentinos [41], five popular CNNs were able to detect diseases in plant leaf images, containing 58 classes. Teimouri et al. [125] detect weed species and growth stages of weeds by counting the number of leaves of 18 different weed species, using the pre-trained Inception-v3 network [121]. Besides plant detection, other areas in agriculture are involving deep learning as well. Zhang et al. [148] identify 13 different types of agricultural machinery images using a ResNet, Inception-v3, and AMTNet.

There is limited literature concerning white cabbage prediction. This chapter provides an overview of the performance of three different methods, which are modeled to predict the (un)successful growth of a white cabbage seedling, and gives a recommendation. A traditional statistical method, Logistic Regression, and several types of deep learning frameworks, Multi-Layered Perceptron, and CNNs are deployed, trained, and tested. The Logistic Regression model serves as a baseline for making a clear comparison with deep learning models. Furthermore, we show that CNNs obtain a more accurate prediction than the traditional method or other deep learning models. The dataset contains gray-scale seedling images on days four to seven. To assess the performance of the models, the same validation set is used to test, compare, and select the models. Further, to evaluate the experiments, the models are selected on loss and accuracy. The best-performing models per species are tested on a test set to assess the ROC curve with the corresponding AUC score. The overall best-performing model, AlexNet, obtains on the test set the highest accuracy of 94%, the lowest loss of 0.175, and the highest AUC-score of 0.95 on day 7. In general, CNNs are known for their ability to accurately analyze high-dimensional data and show to be well suited for this type of research.

3.3 MATERIALS AND METHODS

3.3.1 *Data description*

The data examined was retrieved from a seed breeding corporation operating in agriculture, known as Bejo Zaden: https://www.bejo.com. The data consists of images of white cabbage seedlings, where the photos of the seedlings are taken on days four, five, six, and seven. The reason for this is that seedlings on days earlier than day four are still covered in soil and are not visible in a photo. Seedlings older than day seven are (mostly) overlapping each other, for which individual information gets lost. The seeds are sowed in trays of 150 seedlings each, see Figure 3.1. They are kept under controlled circumstances in a growth chamber, which is called a phytotron. For this species, the seedlings receive 16 hours of fluorescent light at a temperature of 20 degrees Celsius, and receive 8 hours of darkness at a temperature

of 15 degrees Celsius. On day 7, the seedlings are watered. On day 14, the quality of the seedlings is determined by professionals. The seedlings are classified into six different classes based on their growth and maturity. Specifically, classes 5 and 6 are considered to represent a successfully grown seedling; the remaining classes are considered to be unsuccessful. On days four, five, six, and seven photographs of the trays are taken. The size of each image is 1280 × 1024 pixels, where the pixels are gray-scaled and contain values between 0 (black) and 255 (white) (see Figure 3.1).

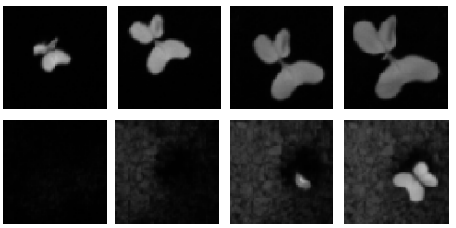


Figure 3.1: From left to right indicate seedlings on days 4, 5, 6, and 7, respectively. In the first row, a successful seedling is shown. The second row shows an unsuccessful seedling with some moss surrounding its area.

There are 88 trays per day available, each containing 150 labeled seedlings, for a total of 13,200 seedlings. The seedlings are followed over time, and therefore, no new seedlings have been added for other days. Since photographs are taken of entire trays, the seedlings are cropped individually and matched with their corresponding label. See Appendix A.1 for more details on the cropping process. Cropping each seedling individually from the entire image results in a variety of different resolutions per seedling, which vary between 64 and 75 pixels per image. Since each of the models requires a fixed input size, all individual seedling images were slightly downscaled to a resolution of 64 × 64 with the Python package PIL.

3.3.2 *Models*

Three different models are examined to classify the data. The first model is the Logistic Regression model (LR), which is used to set a baseline for image prediction. The second model is a Multi-Layered Perceptron (MLP). This is a frequently-used deep learning model in machine learning. The third model is a CNN, known for its ability to capture information from an image. In general, we model the classification of white cabbage seedling images as follows. We let $\mathbf{x}^{(t)} \in \mathbb{R}^d$ be an image with d pixels for $t \in \{4,5,6,7\}$, which represents the day the photo was taken. An image consists of $d = 64 \times 64 = 4096$ pixels. Let $y \in \{0,1\}$ denote whether the seedling successfully grows or not. This suggest $p(y|\mathbf{x}^{(t)})$ as a probabilistic classification model.

Logistic Regression

A widely used (statistical) method for classification tasks is the LR where we model the probability of y as follows for $t \in \{4,5,6,7\}$:

$$p(y=1|\mathbf{x}^{(t)}) = \sigma(\mathbf{w}^T\mathbf{x}^{(t)} + b) = \frac{1}{1+\exp(-(\mathbf{w}^T\mathbf{x}^{(t)}) + b)}, \tag{3.1}$$

where $p(y=0|\mathbf{x}^{(t)}) = 1 - p(y=1|\mathbf{x}^{(t)})$, $\sigma(\cdot)$ represents the logistic sigmoid and $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$ express the parameters [15]. Note that the LR is equivalent to a single-layer perceptron. Important properties of this model are that the model is computationally cheap to run and easy to interpret. The model in Equation (3.1) performs well when data is linearly separable. However, this model cannot capture more complicated input patterns.

Multi-Layered Perceptron

The MLP is able to model more complicated functions compared to logistic regression by stacking multiple layers. The non-linear activation function in-between each layer is capable of capturing non-linearity in the data. An example of a two-layer MLP is given as follows:

$$p(y=1|\mathbf{x}^{(t)}) = \sigma(\mathbf{w}_2^T\mathbf{h} + b_2), \qquad \text{where} \quad \mathbf{h} = g(\mathbf{W}_1\mathbf{x}^{(t)} + \mathbf{b}_1), \tag{3.2}$$

where g can be any non-linear activation function, and the final layer connected to the output uses the sigmoid. The MLP has the ability to learn non-linear mappings, which makes this model powerful Goodfellow et al. [45]. A downside is that this model is computationally expensive when building a large MLP.

In this research, we use the 500-MLP and (1000, 500)-MLP architectures for our examination. The 500-MLP is the model in Equation (3.2), where $\mathbf{h}$ has dimensionality 500, and the (1000, 500)-MLP has two hidden layers with dimensionality 1000 and 500, where the non-linearity g is modeled by the ReLU function. Claudiu Ciresan et al. [30] and Ganesh et al. [43] show that these architectures proved to obtain the best results for the classification of the MNIST [81] dataset.

Convolutional Neural Network models

A CNN is a subclass of the MLP and is known for its ability to capture patterns occurring in images. While the MLP makes use of matrix multiplications where every input signal interacts with the output signal, the CNN replaces at least one matrix multiplication with convolutions. In between convolutions, the weights are shared by using kernels. By making the kernel window smaller than the input, fewer parameters are stored compared to the MLP, which reduces computations and, therefore, needs less memory. CNNs tend to train easier when the network is deep and there is a lot of data available Goodfellow et al. [45].

Normally, training a deep learning model requires a large amount of training data, which might not always be available to the user. A more effective way to train a deep

learning model without gathering more data is the one with transfer learning [101]. Transfer learning is a method aimed at increasing the performance of a task by the use of pre-trained models. Transfer learning utilizes a network that is already trained on a large dataset. Typically, CNNs are pre-trained on ImageNet, which contains more than one million images. Literature shows that transfer learning is well suited for small datasets [74, 101].

We model $p(y = 1|\mathbf{x}^{(t)})$ using one of four CNN architectures: (i) AlexNet [74], (ii) DenseNet [60], (iii) ResNet [56], and (iv) VGG [118]. Since there was a limited amount of data available (only 13,200 images), transfer learning was used and improved the model's performance compared to non-pre-trained CNNs. Therefore, we employed one architecture of each model species, namely DenseNet121, ResNet101, and VGG16. Further, pre-trained networks typically use three input channels (RGB), whereas our data is a single channel (grey-scale). For that reason, the single channel is repeated three times to make it compatible with pre-trained networks.

3.3.3 *Pre-processing*

The aim of pre-processing is to improve performance and obtain faster convergence. A frequently used pre-processing step for image recognition is the normalization of the data. Therefore, we normalized the data by dividing each seedling image by 256 and subtracting it with 0.5 to be in the range $[-0.5, 0.5]$.

Another frequently used technique is data augmentation. Especially with high dimensional images, augmentation can greatly improve model performance Goodfellow et al. [45]. Augmentation includes transformations of the input images such as cropping, flipping, and rotating and has shown to be successful [139]. Krizhevsky et al. [74] suggest augmentation to reduce overfitting by using translations and horizontal reflections. Following [45, 74, 139], a number of augmentations were used to improve model performance: (i) 90 degrees random rotations, (ii) random horizontal flip, (iii) random vertical flip, and (iv) random affine transformations.

The pre-processing of this research was conducted in Python 3.7.2. The models used were trained and tested using PyTorch 1.3.0 [103]. All models were trained using the GPU of an NVIDIA-SMI GTX1080 card, in a Linux environment Debian GNU operating system.

3.3.4 *Training, validation and test set*

The data was split into a training, validation, and test set. The training dataset was used to train the models. The validation set was used to tune hyperparameters. On this set, the user can measure model performance and compare different models. Furthermore, a test set was held out during the entire training process, which contained images the model had never seen before. The best-performing model that arose during training was evaluated on the test set. This set was used to measure the final model performance. The partitioning of the train, validation, and test set was, respectively, 60 : 20 : 20, which resulted in 7920 : 2640 : 2640 examples for the seedling dataset.

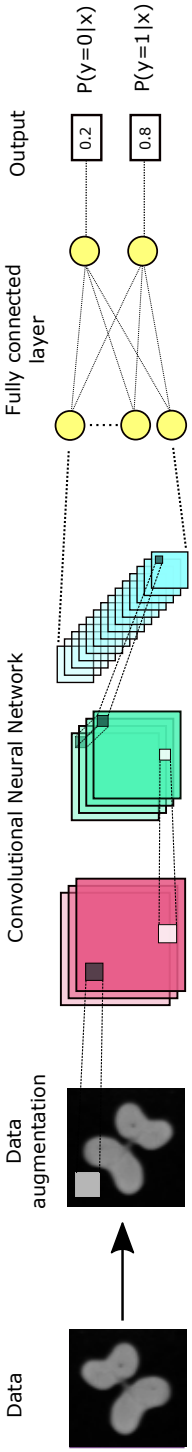


Figure 3.2: The framework from pre-processing to prediction.

Table 3.1: Fraction of (un)successful seedlings.

Day	Training		Validation		Test	
	Good	Bad	Good	Bad	Good	Bad
4	0.856	0.144	0.857	0.143	0.849	0.151
5	0.856	0.144	0.850	0.150	0.855	0.145
6	0.855	0.145	0.853	0.147	0.855	0.145
7	0.851	0.149	0.854	0.146	0.867	0.133

3.3.5 *Dealing with imbalanced data*

When data has imbalanced classes, models may place more emphasis on the majority class, which results in degraded performance for the minority class. In deep learning, there are several methods that deal with this problem. One of these methods is undersampling, where the majority class is undersampled. Another method is oversampling, where the minority class is oversampled [84]. [20] show that CNNs, specifically deployed for datasets such as MNIST and CIFAR-10 [73], have better performance when the imbalance of classes makes use of oversampling instead of undersampling. Shorten and Khoshgoftaar [116] provide a survey on the examination of data augmentation for deep learning and conclude that the combination of oversampling and data augmentation can be a very useful technique for the construction of a more informative dataset. Therefore, we used data augmentation to create diversity in the training set and to improve model performance.

Table 3.1 indicates that there was an imbalance between the two classes in our data. There were approximately seven times fewer unsuccessful seedling images compared to the successful seedlings. Since the dataset was small with 13,200 seedlings, we oversampled [84] the minority class of the training data for each day by copying random images from the minority class. We made random copies with the replacement of the minority class to obtain the same size as the majority class.

3.3.6 *Objective*

Each of our models was trained by maximizing the log-likelihood, $\log p(y_n|\mathbf{x}_n^{(t)})$, where the distribution is modeled by a Bernoulli distribution. This distribution is computed by our models, which output a value for each class, followed by a softmax function called $\hat{y}_{nk}$. Here n denotes the n-th example and $k \in \{0,1,\ldots,K-1\}$ denotes a specific class for K classification classes. Maximizing the log-likelihood is equivalent to minimizing the categorical cross-entropy loss [15]:

$$\begin{aligned}\mathcal{L}_n &= -\log p(y_n|\mathbf{x}_n^{(t)}) \\ &= -\sum_{k=0}^{K-1} \mathbb{1}_{\{y_n=k\}} \log(\hat{y}_{nk}),\end{aligned} \tag{3.3}$$

where $\mathbb{1}_{\{y_n=k\}}$ denotes an indicator function for the label, in our problem, we have $K=2$ indicating a binary classification task to have the label successful or unsuccess-

ful. The loss for our entire dataset is denoted with: $\mathcal{L} = \sum_{n=1}^{N} \mathcal{L}_n$, for N datapoints.

An adjustment for the categorical cross-entropy loss of Equation (3.3) is to assign a cost to each of the classes. Especially for unbalanced data, this emphasizes the minority class. As a result, the model attempts to minimize misclassification of this class:

$$\begin{aligned} \mathcal{L}_n &= -\log p(y_n \,|\, \mathbf{x}_n^{(t)}) \\ &= -\sum_{k=0}^{K-1} \gamma_k \mathbb{1}_{\{y_n = k\}} \log(\hat{y}_{nk}), \end{aligned} \tag{3.4}$$

where γ_k is the weight assigned to class k. In our case, unsuccessful seedlings are assigned a weight of $\gamma_0 = 1$, and for successful seedlings γ_1 is the ratio of the number of examples of the minority class over the number of examples of the majority class.

Since we trained our models by minimizing the cross-entropy loss Equation (3.3), we compared and selected our models based on the overall performance on loss. Furthermore, we also experimented with a weighted loss, Equation (3.4), to examine the effects of the best-performing model and make a clear comparison between several methods that deal with imbalanced classes.

Weight decay

Weight decay is a regularization method, also known as the L_2 norm, and is used to penalize the complexity of the model by adding a small regularization coefficient $\lambda > 0$, multiplied with the sum of squares of the weight (parameters) of our model to the loss function:

$$\mathcal{L}^E = \mathcal{L} + \frac{\lambda}{2} \|\mathbf{w}\|^2, \tag{3.5}$$

where $\mathcal{L}^E$ is the regularized categorical cross entropy loss and where $\mathcal{L}$ is derived from Equation (3.3). For $\lambda = 0$, we have the original loss function, while for $\lambda > 0$, we add a penalty to the complexity of the model. Due to the penalty, the weights of the model are prevented from becoming large, which can prevent overfitting. Further, $\|\mathbf{w}\|^2 = \sum_{i=1}^{d} \|w_i\|^2$ [15], where d indicates the number of parameters of weight w.

Optimizer and learning rate

In this study, we started with the Adam optimizer [68] to optimize our models. Due to unstable convergence, we switched to Stochastic Gradient Descent (SGD) [18]. As a result, our models were able to obtain a stable and better generalization.

The learning rate, η, of SGD is a hyperparameter controlling how much the weight parameter w_i of a model is updated during the training process per iteration i:

$$w_{i+1} \leftarrow w_i - \eta \frac{\delta \mathcal{L}^E}{\delta w_i}, \tag{3.6}$$

where $\eta > 0$. After each iteration i, a new weight w_{i+1} is computed, which determines the new direction towards the optimum of the loss function. Setting the learning rate too high will result in fast convergence with a risk of "overshooting" the optimum. Setting the learning rate too low can result in slow convergence [15].

3.4 EXPERIMENTS AND ANALYSIS

3

3.4.1 *Performance metrics*

A performance metric is used to express model performance and can be computed using the confusion matrix in Table 3.2 in the case of binary classification. The confusion matrix expresses a model's classification power. A frequently used performance metric when the data has an equal class balance is accuracy. The accuracy expresses the frequency of correctly classified classes and is given by:

$$\text{accuracy} = \frac{TP + TN}{TP + FN + FP + TN}. \tag{3.7}$$

Due to the imbalance of seedling classes (see Table 3.1), the default accuracy that can be obtained when a model predicts all examples as successful seedlings is approximately 85%. Therefore, improvements in accuracy on the validation set need to be compared to this baseline.

Table 3.2: The confusion matrix.

		Predicted	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

When a binary predictive model classifies a (new) data point, the model outputs a probability between 0 and 1 that the data point belongs to a certain class. A performance measure that expresses the performance of a binary classifier for all classification thresholds is the Receiver Operating Characteristic (ROC) curve. For each threshold, the confusion matrix (see Table 3.2) is computed following the True Positive Rate (TPR), which is also known as recall or *sensitivity* = $\frac{TP}{TP+FN}$ and False Positive Rate (FPR): $1 - \textit{specificity} = \frac{TN}{FP+TN}$. Next, the TPR is plotted against the FPR for all the thresholds, which form the ROC curve that visualizes the trade-off between the sensitivity and specificity. The ROC curve aids the user in selecting a threshold that gives the lowest cost for classifying data points incorrectly. The AUC expresses the entire area under the ROC curve and is in the range: $[0, 1]$. The higher the AUC score, the better the model predicts classes [19, 52].

In this study, the best-performing model was selected based on the lowest loss and highest accuracy curves with respect to validation. After selection, the test set was used to evaluate this model's generalization performance. In addition to accuracy

and loss, the overall model performance was also assessed using the ROC curve and corresponding AUC.

3.4.2 *Hyperparameters*

Every predictive model is described by parameters, where the parameters aid in the prediction task. In general, parameters are estimated from the data during training. Hyperparameters are characteristics of a model that cannot be estimated using gradient descent. Instead, their value is set before training. Deep learning modules usually provide default settings for hyperparameters, but the user can also search through the hyperparameter space to find optimal settings; this is known as a grid search.

Table 3.3 presents different hyperparameter settings. An epoch represents the number of times the model trains on the entire dataset. The models were trained for 500 epochs. Training a model requires each model to see every training example. In machine learning, small batch sizes, B, are typically chosen as $B \in \{32, 64, 128, 256\}$ [45]. These batch size settings showed to be successful in fields, such as reinforcement learning [97] and image recognition [118]. Therefore, we trained our CNN with a batch size of 128. Additionally, to stay consistent with treating each model with the same techniques, we trained our LR and MLP with a batch size of 128 and used the same batch size on the validation set. A grid search was applied by increasing the step size with the same value to find optimal settings for hyperparameter weight decay and the learning rate. In the upcoming sections, these hyperparameters will be explained.

Table 3.3: Hyperparameter settings.

(Hyper)parameter	**Setting**
Epochs	500
Batch size	128
Weight decay	$1e^{-3}, 1e^{-5}, 1e^{-7}$
Learning rate	$1e^{-2}, 1e^{-3}, 1e^{-4}$

3.4.3 *Experiments*

Due to the imbalance of the dataset and based on the literature, we used oversampling in combination with augmentation to create diversity and an equal class balance. For each of the model types (LR, MLPs, and CNNs), a grid search was applied, as specified in Table 3.3. The hyperparameter search was performed on the validation set. Based on loss and accuracy curves, the models were compared. To optimize the models, we first searched for an optimal learning rate setting, and secondly, with an optimal learning rate setting, we searched for an optimal weight decay value. After the application of a grid search, optimal settings were found and presented in Table 3.4. Note that two model types, namely, LR and CNN, performed

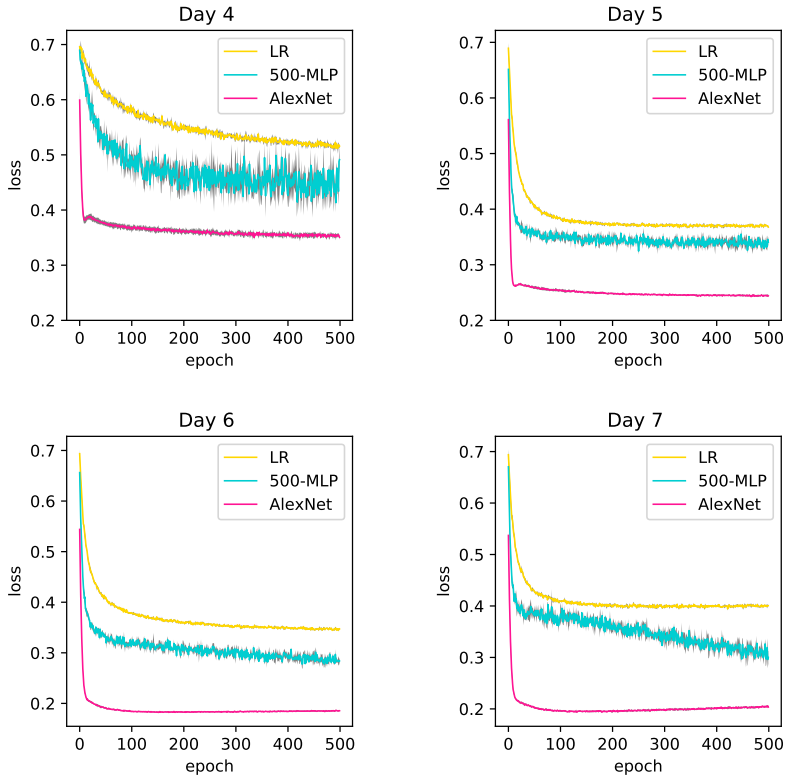


Figure 3.3: Loss on validation set of the best three model types with standard deviation in gray.

best with a learning rate of $1e^{-4}$. In most packages, this is usually the default setting. The MLP performed optimally with a higher learning rate setting of $1e^{-3}$. Furthermore, the best-performing model per species was selected based on the overall lowest loss and highest accuracy.

We ran the best-performing model per species with optimal learning rate settings three times. Figure 3.3 shows the mean of the loss curves of these three experiments. The light gray area around the mean expresses the standard deviation. Since AlexNet obtains stable loss curves, the standard deviation is hardly visible and can be best viewed electronically. As we can see, AlexNet scores significantly better than the other two models on all days. The model scores around 0.05 lower in loss on day 4 compared to the 500-MLP and 0.1 lower in loss on days 5 and 6. On day 7, the model quickly obtains a stable convergence, while in general, the 500-MLP seems to fluctuate a lot. Even though the LR scores the highest loss curves, the model does seem to obtain stable curves and convergence compared to the 500-MLP. A reason that the 500-MLP fluctuates the most on day 4 might have to do with the non-developed seedlings. These seedlings are in their earliest growth stadium and

Table 3.4: Best hyperparameter values.

Model type	Learning rate	Weight decay
AlexNet	$1e^{-4}$	$1e^{-5}$
500-MLP	$1e^{-3}$	$1e^{-5}$
LR	$1e^{-4}$	$1e^{-7}$

are sometimes still covered in the soil, see Figure 3.1, or it may be difficult to extract sufficient information for the model. The upcoming section will further investigate the overall performance by assessing the ROC curves on the test set. In general, we observe that AlexNet obtains a quick and stable convergence of the loss and accuracy curves, with little fluctuations compared to the other two models. Taking all these factors into account, AlexNet is most suitable for the prediction of seedlings, based on validation loss and accuracy only.

In addition, AlexNet was compared to other CNNs as well. As we can see in Figures 3.4 and 3.5, the other three CNNs had close performance compared to AlexNet based on loss and accuracy curves. On day 4, all four CNNs have comparable loss curves. DenseNet and ResNet fluctuate more in performance on this day. AlexNet has a slightly lower loss performance than VGG, while both models seem to obtain the most stable and least fluctuating curves. On days 5, 6, and 7, AlexNet outperformed the other models. Therefore, AlexNet was chosen as the best overall performing model. In general, all four CNNs performed better than the 500-MLP and LR, as in Figure 3.3. Further research should focus on VGG since this model seems to obtain the most stable and lower convergence after AlexNet or focus on testing other pre-trained architectures for DenseNet, ResNet, and VGG. If AlexNet performs well due to its small architecture, other smaller architectures might be better to capture the data structure than some of the bigger versions we used in this research.

3.4.4 *Results*

All three models were evaluated three times on a separate test set for the final average performance of the corresponding loss and accuracy. The results can be found in Table 3.5. AlexNet is the best-performing model in terms of loss and accuracy on all days. Note that on day 4, the accuracy of AlexNet is just above the baseline of 0.849, which might indicate that the model is weakly capable of predicting some of the unsuccessful seedlings correctly. The 500-MLP performs worse than the baseline on day 4 and around the baseline on days 5 and 6. On day 7, the 500-MLP scored the highest on that day, with an accuracy of 0.906. The LR scores are lower than the baseline for all days and even lower on day 7, compared to the previous day.

Based on loss and accuracy, AlexNet seems best capable of capturing the prediction of the seedlings. In general, all models have a low standard deviation, which is at least smaller than 0.007, indicating that the models obtain similar and stable scores per experiment. Even though the models sometimes seem to perform worse than

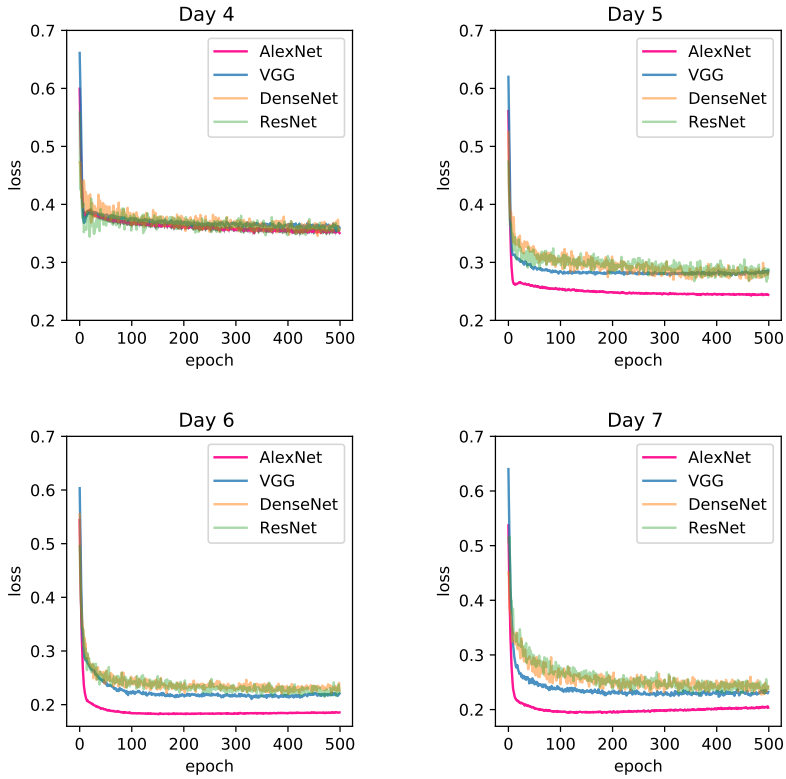


Figure 3.4: Loss curves on the validation set of all four CNN types.

the baseline, the trade-off between sensitivity (TPR) and specificity (1-FPR) is not visible in accuracy and loss only. Therefore, the models were further examined. We assessed the ROC curves to evaluate the generalization of the three models further. As we can see in Figure 3.6, AlexNet outperforms the other two models on each day. In Table 3.5, we can find the corresponding mean and standard deviation of the AUC scores of all experiments. We see that AlexNet outperforms the 500-MLP and LR on all days, based on these scores. On day 4 and 5 AlexNet scores 0.03 and 0.02 higher with *AUC* 0.91 and 0.94, respectively. On days 6 and 7, AlexNet obtains the best scores of, respectively, 0.94 and 0.95, with the lowest fluctuations in standard deviation. Yet, the model is closely followed by the 500-MLP with an AUC score of 0.01 lower on these days.

When we look at the ROC curves, we observe that AlexNet has a better trade-off between the sensitivity and specificity on day 4 since the line is slightly closer to the left border and closer to the top border compared to the other two models. For example, if we want to classify the successful seedling rate with a sensitivity of $TPR = 0.93$, this will for AlexNet approximately result in a specificity of around $1-FPR \approx 1-0.35 \approx 0.65$, indicating that more than half of the unsuccessful seedlings

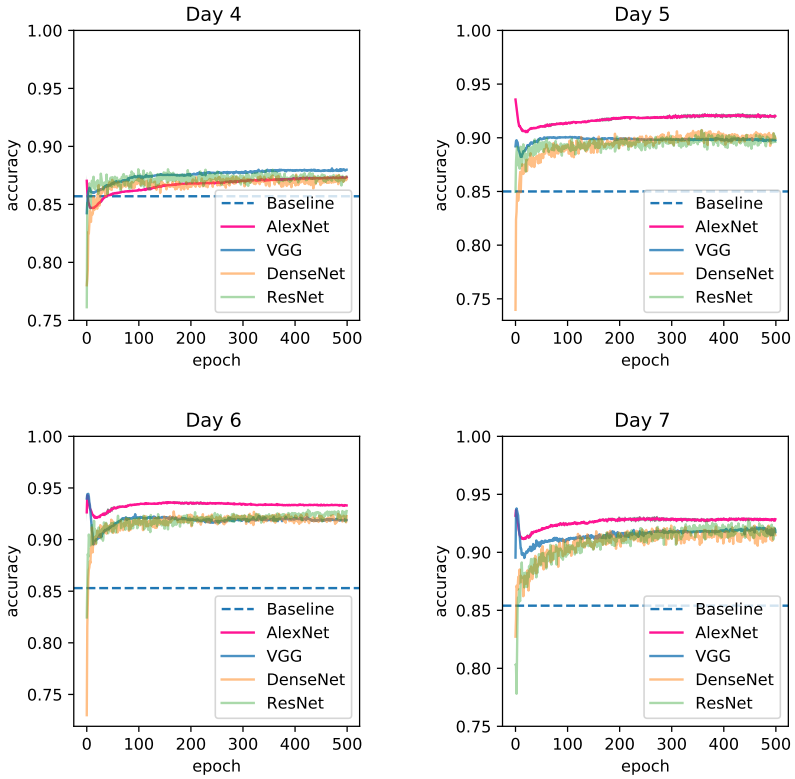


Figure 3.5: Accuracy curves on the validation set of all four CNN types.

will be classified correctly. While this threshold for the 500-MLP and LR will result in a specificity of around $1 - FPR \approx 1 - 0.55 \approx 0.45$, less than half of the unsuccessful seedlings will be classified correctly. For these thresholds, the model seems to be more accurate in the prediction of classifying unsuccessful seedlings. On day 5 and 6, the ROC curves of AlexNet are the best. Yet, the curves of the 500-MLP and LR follow approximately the same shape as AlexNet, which indicates that all models obtain more or less the same trade-off. On day 7, we observe that AlexNet and the 500-MLP are close to each other while the LR has a more gentle bend line, indicating that the trade-off between the sensitivity and specificity is less accurate to capture.

Table 3.5: Loss, accuracy, and AUC results obtained by taking the average performance over three runs. The gray highlights indicate the best-performing model per day.

Day	**Loss** (mean±stdev)			**Accuracy** (mean±stdev)			**AUC** (mean±stdev)		
	LR	500-MLP	AlexNet	LR	500-MLP	AlexNet	LR	500-MLP	AlexNet
4	0.503±0.003	0.384±0.006	**0.354**±0.004	0.701±0.004	0.809 ±0.005	**0.865**±0.001	0.87±0.001	0.88±0.000	**0.91**±0.000
5	0.378±0.001	0.34 ±0.001	**0.262**±0.001	0.825±0.000	0.872 ±0.003	**0.911**±0.001	0.92±0.000	0.92±0.000	**0.94**±0.000
6	0.376±0.001	0.294±0.002	**0.207**±0.001	0.847±0.001	0.892±0.001	**0.93**±0.000	0.92±0.001	0.93±0.001	**0.94**±0.000
7	0.381±0.001	0.256±0.005	**0.175**±0.001	0.835±0.001	0.906±0.003	**0.94**±0.001	0.92±0.001	0.94±0.000	**0.95**±0.000

In conclusion, since accuracy and loss only do not give sufficient insights into the trade-off between the (un)successful seedling prediction, the ROC curves with corresponding AUC show that AlexNet is best capable of capturing the seedling classification compared to the 500-MLP and LR. Note that instead of randomly guessing, all three models are capable of finding patterns in the images, even on day 4. Depending on the trade-off between the sensitivity and specificity, the user can decide where to place the threshold.

3.4.5 *Additional experiments*

Besides our design choices based on literature to deal with the imbalanced classes, we experimented with several other methods with our optimized AlexNet. We tested the model on (1) original data without any data transformations, (2) only augmented data without oversampling, (3) a weighted loss, which is another method to balance the unequal classes, see Equation (3.4), (4) weighted loss with augmentation, and (5) only oversampling without augmentation. We compared the results with our design choices (6). In Table 3.6, the results of these methods on the validation set are shown. To exemplify the table:

1. Compared to our design, training AlexNet with original data obtained slightly better performance on the validation set in terms of loss and accuracy. To further investigate the model performance when no transformations are applied, we also trained and evaluated the generalization performance of this model by assessing the ROC curve with the corresponding AUC. Remarkably, this experiment resulted in scores comparable to our method on days 6 and 7 in terms of AUC-score, respectively, 94 and 95. Yet, on days 4 and 5, training on original data seems to lose a lot of information. On day 5, the model scores an $AUC = 0.89$, which is 0.05 lower than our design choice for AlexNet and even 0.03 than the 500-MLP and LR for that day. As we can see in Figure 3.7, on day 4 the model performs even worse than baseline between FPR 0 till 0.4. The corresponding AUC score is 0.29 lower with a score of 0.62, compared to our method with $AUC = 0.91$. In conclusion, training on original data (without augmentation and oversampling) tends to cause information to be lost in earlier days. Yet, further research should investigate if the model improves

when optimizing on original data only and then analyze if the other days might contain sufficient information to make this model compatible with ours.

2. Augmentation obtained similar performance compared to training on the experiment in 1, the original data only.
3. The weighted loss worsened the results for the loss on days 4 and 5. Yet, results obtained on days 6 and 7 were similar to those of our design choices. Based on the accuracy, this method obtained a performance similar to ours. Further research should investigate if this method might be compatible with ours.
4. Weighted loss in combination with augmentation obtained the same performance as without augmentation. Therefore, it performed worse than our design.
5. Only oversampling worsened the results. Additionally, the model was overfitting on validation when comparing training and validation. Looking at the literature, this might have to do with the fact that augmentation gives more diversity to the data than using only replicates of the data.

Table 3.6: Data experiments improving/worsening AlexNet performance.

Nr.	Experiment	Model improvements	Overall result
1	Original	✓/✗	No improvement or worsening
2	Augmentation	✓/✗	No improvement or worsening
3	Weighted loss	✗	Worsening
4	Weighted loss + augmentation	✗	Worsening
5	Oversampling	✗	Worsening
6	**Oversampling + augmentation**	✓	**Best performance**

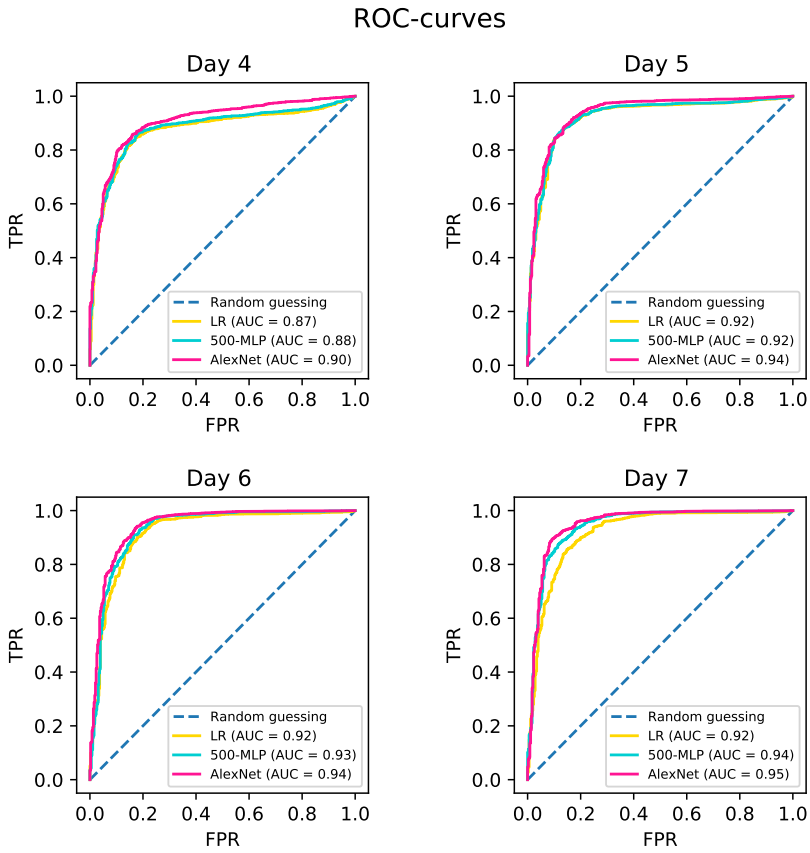


Figure 3.6: ROC-curves on the test set of the best performing models.

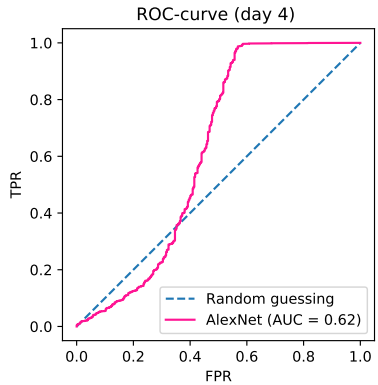
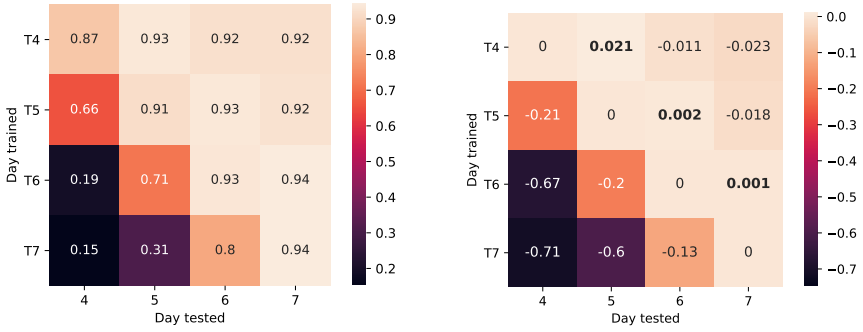


Figure 3.7: ROC-curve on the test set of AlexNet on day 4. The model was trained on the data without augmentation and oversampling.

3.4.6 *Generalization of AlexNet*

In this research, each best-performing model was trained and tested on the same day. To further investigate the generalization of AlexNet to other days, we fixed the day the model was trained and tested this model every other day. The heatmap of the results can be found in Figure 3.8. The y-axis expresses the day the model was trained on, while the x-axis shows the days the fixed model was tested on. The heatmap in Figure 3.8a expresses the accuracy, where the accuracy for the models trained on days 4, 5, 6, and 7 and tested on the same day is respectively 0.87, 0.91, 0.93 and 0.94 (see also the accuracy in Table 3.5). Figure 3.8b expresses the difference in accuracy with respect to the diagonal. To clarify the heatmap, a model trained and tested on day 5 obtains an accuracy of (a) 0.91, resulting in a difference of (b) 0. A model trained on day 6 and tested on day 5 obtained a difference of (b) -0.2, resulting in an accuracy of (a) $0.91 - 0.2 \approx 0.71$ from the model trained and tested on the same day.



(a) Accuracy **(b)** Difference Accuracy

Figure 3.8: A heatmap that expresses (a) the accuracy and (b) the difference in accuracy with respect to the diagonal. The y-axis indicates the day the model was trained and fixed on and on the x-axis the day the fixed model was tested on.

We observe that AlexNet trained on day 7 and tested on day 4, 5, 6 obtains a much lower accuracy than when the model is trained and tested on the same day. In general, we see that the trained models obtain worse results when tested on earlier days. Remarkably, we notice that models trained on earlier days perform quite comparable when tested on later days with only a small difference. Additionally, these models even (slightly) exceed the accuracy of models that are trained and tested on data of the same day. For example, AlexNet trained on day 4 and tested on day 5, obtains a positive difference of 0.021, resulting in an accuracy of 0.932, which is higher than the AlexNet trained and tested on day 5 (accuracy 0.911). In conclusion, models trained on earlier days seem to generalize for later days and seem to capture sufficient information from early-developing seedlings. Furthermore, models trained on later days perform worse when tested on seedlings from earlier days. Further research should focus on the growth of seedlings over time

and investigate the generalization of models only trained on (an) earlier day(s) or examine the generalization when models are trained on a mixture of seedlings where there is no distinction between early and later days.

3.5 CONCLUSION

3

During this research we examined three different model types, namely, LR, MLP, and CNN, to classify the (un)successful growth of white cabbage seedlings based on only an image of the seedling. The dataset was retrieved from Bejo, a company operating in agriculture, and consisted of 13,200 gray-scaled seedling images. The seedlings lived under controlled circumstances in a phytotron and were labeled after 14 days by professionals. According to research concerning image classification with little data, four pre-trained CNNs were developed. To compare the CNNs with other methods, two recommended MLP architectures and a traditional statistical method LR were deployed. Oversampling was applied to create an equal class balance. In combination with data augmentation, diversity in the dataset was created which improved model performance. Further, the model was trained with an SGD optimizer, and weight-decay was applied. AlexNet outperformed the LR and MLP and outperformed the other four CNN types based on the lowest loss and the highest accuracy on validation data. On the test set, AlexNet outperformed the 500-MLP and LR based on (1) the lowest loss, (2) the highest accuracy, and (3) the best ROC curve with corresponding AUC-score for each day. According to these three points, the model showed to be robust for the prediction of (un)successful white cabbage seedlings. To answer the main questions of this research in detail:

1. *Can the model predict if a white cabbage seedling is going to grow (un) successfully?*
   Based on the information of only a seedling image, the prediction of AlexNet classifying (un)successful seedlings obtained a loss of **0.175**, an accuracy of **0.94** and an AUC-score of **0.95** all on day **7**. Based on this information, we can conclude that the model can accurately determine if a seedling is going to grow (un)successfully.

2. *How does time influence the growth predictability of seedlings?*
   We observe that the most predictable day is day 7, as presented in Table 3.5. On this day, the loss, accuracy, and AUC are higher than on the other days. On day 6, a small decrease in predictability is observed compared to day 7. The biggest difference in performance metrics is between day 4 and 5. Here we observe a big increase in loss of 0.044, lower accuracy of 0.046 and lower AUC-score 0.03 compared to day 5. The results show that there is a trade-off between time and accuracy. Given that the accuracy increases most from day 4 to day 5, it may be worth waiting until day 5. In contrast, the difference between day 6 and 7 is relatively small, and predictions from day 6 may be sufficiently accurate. Interestingly, the model trained on day 4 performed reasonably on all other days (see Figure 3.8). This indicates that training on earlier days generalizes to predictions of later days. However, the models trained on later days do not generalize well to earlier days.

3. *How is the comparison between the traditional model and deep learning models?*
   This research points out that CNN, AlexNet, overall outperformed the LR and MLP. This might be due to the fact that a CNN is able to capture information from an image instead of information per pixel value. In general, we observed that all four CNNs performed best on validation compared to the best MLP and LR. Therefore, the CNNs outperformed the other methods. We suggest further investigating the other CNN architectures, especially the VGG network, since this network obtains the most stable and lowest convergence after AlexNet.

To further clarify the best-performing model of this research, we suggest a more in-depth AlexNet investigation with attention maps that should offer insights into local seedling recognition. This should aid in making adjusting decisions of the network by offering a deeper network visualization and likely understanding. Additionally, we recommend further investigating the optimization of AlexNet on the original dataset. Furthermore, future research should focus on the growth of seedlings over time. This might improve the classification due to information that is hidden between the days.

A APPENDIX FOR WHITE CABBAGE SEEDLINGS

A.1 *Pre-processing details*

For the pre-processing, each photograph containing 150 seedlings are cropped from their original photo. As can be seen in Figure A.1, the seedlings are equally sowed into 10 rows and 15 columns. On the borders of each photo, there is a stroke of black space representing earth. Since we are only interested in the seedlings, they are cropped with Algorithm 1, to have as little black space in between each seed as possible. As input, each photo of format 1280×1024 is binarized with Python package openCV, to make a clear distinction between seeds (1) and earth (0). Next, with the binarized photo, the algorithm computes the starting coordinates on how to divide the area where the seedlings are centered. As input, the function takes in the binary image x, and threshold φ which serves as a threshold to determine when the most outer seedling (leave) is hit. To find exact row starting coordinates $I = 1024$ and $J = 1280$, and *vice versa* for the column starting coordinates. In Figure A.1 the starting coordinates of this image are marked with red dots. The function searches for the most outer located seedling, where the threshold φ determines how many pixels need to be hit to assume a seedling is found.

After determining the starting coordinates for rows and columns with Algorithm 1, the area between the coordinates is equally divided into 10 rows and 15 columns for seedlings on day 6 and 7. Since seedlings on these days (sometimes) start to overlap and are growing larger, this method aids in centering the crops accurately. For seedlings on day 4 and 5, a black space of 17 pixels between the row and 20 pixels between columns of the starting coordinates and outer border are added. In this manner, an individual seedling image is more or less centered. When no black space was added, this resulted in irregular cropping such as two seedlings in one image or no seedling at all.

Algorithm 1 Determining starting coordinates

```
function STARTING_COORDINATES(x, φ, I, J)
    start, end ← 0, 0
    for i = 1, ..., I do
        count ← 0
        for j = 1, ..., J do
            if x[i, j] = 1 then
                count ← count + 1
            else if count > φ & r_start = 0 then
                start ← i
            else if count > φ then
                end = j
            end if
        end for
    end for
    return start, end
end function
```

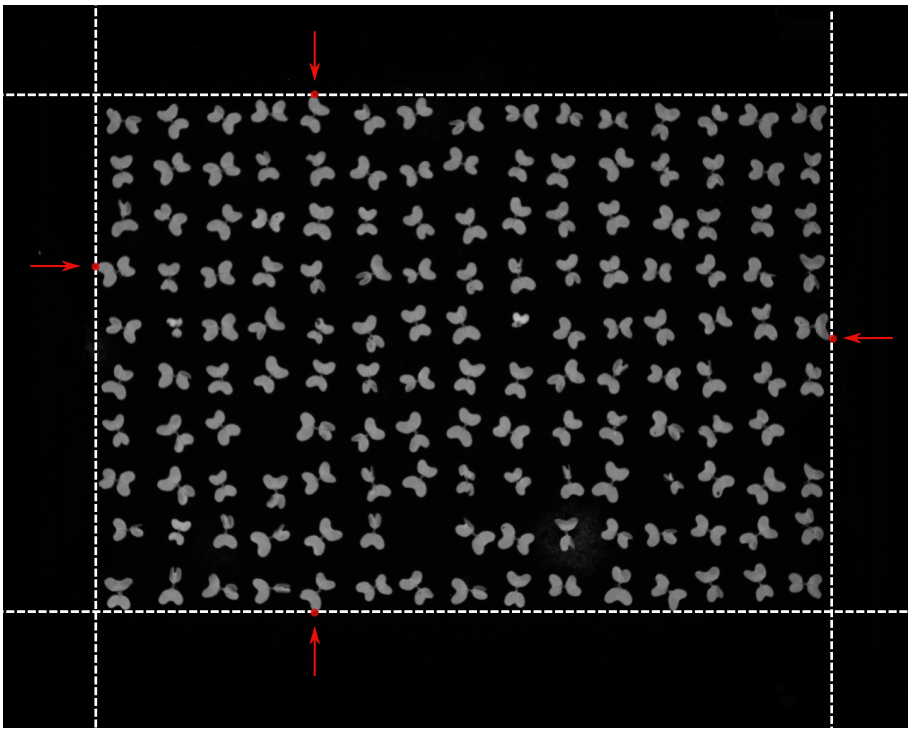


Figure A.1: Starting coordinates, marked with red dots, for cropping of the image (seedlings on day 6 with no black space between the starting coordinates and outer border).

3

Part II

HIGH-DIMENSIONAL DENSITY ESTIMATION

4

INVERTIBLE DENSENETS

Abstract

Normalizing Flows have restricted architectures because they need to be invertible. In this chapter we introduce i-DenseNet, a more flexible invertible neural network that is better in estimating the distribution density of the data, based on Residual Flows. Our method relies on an analysis of the Lipschitz continuity of the concatenation in DenseNets, where we enforce the invertibility of the network by satisfying the Lipschitz constant. We propose a learnable weighted concatenation, which not only improves the model performance but also indicates the importance of the concatenated weighted representation. Furthermore, we introduce the Concatenated LipSwish as an activation function, for which we show how to enforce the Lipschitz condition and which boosts performance. The new architecture, i-DenseNet, outperforms Residual Flow and other flow-based models on density estimation evaluated in bits per dimension, where we utilize an equal parameter budget. Moreover, we show that the proposed model outperforms Residual Flows when trained as a hybrid model where the model is both a generative and a discriminative model.

Based on [4]:
Yura Perugachi-Diaz, Jakub M. Tomczak, Sandjai Bhulai
Invertible DenseNets
3rd Symposium on Advances in Approximate Bayesian Inference, AABI, January - February, virtual, 2021.

Based on [5]:
Yura Perugachi-Diaz, Jakub M. Tomczak, Sandjai Bhulai
Invertible Dense Networks with Concatenated LipSwish
Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems, NeurIPS, December 6-14, virtual. 2021.

4.1 INTRODUCTION

Neural networks are widely used in supervised learning tasks such as classification, where models are trained to predict labels. Besides classification, neural networks are also utilized to build flexible density estimators of the true distribution of the observed data [93, 110]. The resulting deep density estimators are called deep generative models. Since this is an open problem in deep learning, deep generative models try to approximate the true data distribution for high-dimensional data. These models are used to generate realistic-looking images which are hard to separate from real ones, detect adversarial attacks [42, 63], and are even used for hybrid modeling [98], which has the property to both predict a label (classify) and generate.

4

Many deep generative models are trained by maximizing the (log-)likelihood function and their architectures come in different designs. For instance, causal convolutional neural networks are used to parameterize autoregressive models [99, 100] or various neural networks can be utilized in Variational Auto-Encoders [67, 109]. The other group of likelihood-based deep density estimators, *flow-based models* (or *flows*), consist of invertible neural networks since they are used to compute the likelihood through the change of variable formula [108, 122, 123]. The main difference that determines an exact computation or approximation of the likelihood function for a flow-based model lies in the design of the transformation layer and tractability of the Jacobian-determinant. Many flow-based models formulate the transformation that is invertible and its Jacobian is tractable [14, 34–36, 70, 102, 108, 129].

Recently, Behrmann et al. [11] proposed a different approach, namely, deep-residual blocks as a transformation layer. The deep-residual networks (ResNets) of [56] are known for their successes in supervised learning approaches. In a ResNet block, each input of the block is added to the output, which forms the input for the next block. Since ResNets are not necessarily invertible, Behrmann et al. [11] enforce the Lipschitz constant of the transformation to be smaller than 1 (i.e., it becomes a contraction) that allows applying an iterative procedure to invert the network. Furthermore, Chen et al. [24] proposed Residual Flows, an improvement of i-ResNets, that uses an unbiased estimator for the logarithm of the Jacobian-determinant.

In supervised learning, an architecture that uses fewer parameters and is even more powerful than the deep-residual network is the Densely Connected Convolution Network (DenseNet), which was first presented in [60]. Contrary to a ResNet block, a DenseNet layer consists of a concatenation of the input with the output. The network showed to improve significantly in recognition tasks on benchmark datasets such as CIFAR10, SVHN, and ImageNet, by using fewer computations and having fewer parameters than ResNets while performing at a similar level.

In this chapter, we extend Residual Flows [11, 24], and use densely connected blocks (DenseBlocks) as a residual layer. First, we introduce invertible Dense Networks (i-DenseNets), and we show that we can derive a bound on the Lipschitz constant to create an invertible flow-based model. Furthermore, we propose the Concatenated LipSwish (CLipSwish) as an activation function, and derive a stronger Lipschitz

bound. The CLipSwish function preserves more signal than LipSwish activation functions. Finally, we demonstrate how i-DenseNets can be efficiently trained as a generative model, outperforming Residual Flows and other flow-based models under an equal parameter budget.

4.2 BACKGROUND

FLOW-BASED MODELS Let us consider a vector of observable variables $x \in \mathbb{R}^d$ and a vector of latent variables $z \in \mathbb{R}^d$. We define a bijective function $f : \mathbb{R}^d \to \mathbb{R}^d$ that maps a latent variable to a datapoint $x = f(z)$. Since f is invertible, we define its inverse as $F = f^{-1}$. We use the *change of variables formula* to compute the likelihood of a datapoint x after taking the logarithm, that is:

$$\ln p_X(x) = \ln p_Z(z) + \ln |\det J_F(x)|, \tag{4.2.1}$$

where $p_Z(z)$ is a base distribution (e.g., the standard Gaussian) and $J_F(x)$ is the Jacobian of F at x. The bijective transformation is typically constructed as a sequence of K invertible transformations, $x = f_K \circ \cdots \circ f_1(z)$, and a single transformation f_k is referred to as a *flow* [108]. The change of variables formula allows for evaluating the data in a tractable manner. Moreover, the flows are trained using the log-likelihood objective where the Jacobian-determinant compensates for the change of volume of the invertible transformations.

RESIDUAL FLOWS Behrmann et al. [11] construct an invertible ResNet layer which is only constrained in Lipschitz continuity. A ResNet is defined as: $F(x) = x + g(x)$, where g is modeled by a (convolutional) neural network and F represents a ResNet layer (see Figure 4.1.1a) which is in general not invertible. However, g is constructed in such a way that it satisfies the Lipschitz constant being strictly lower than 1, Lip(g) < 1, by using spectral normalization of [47, 96]:

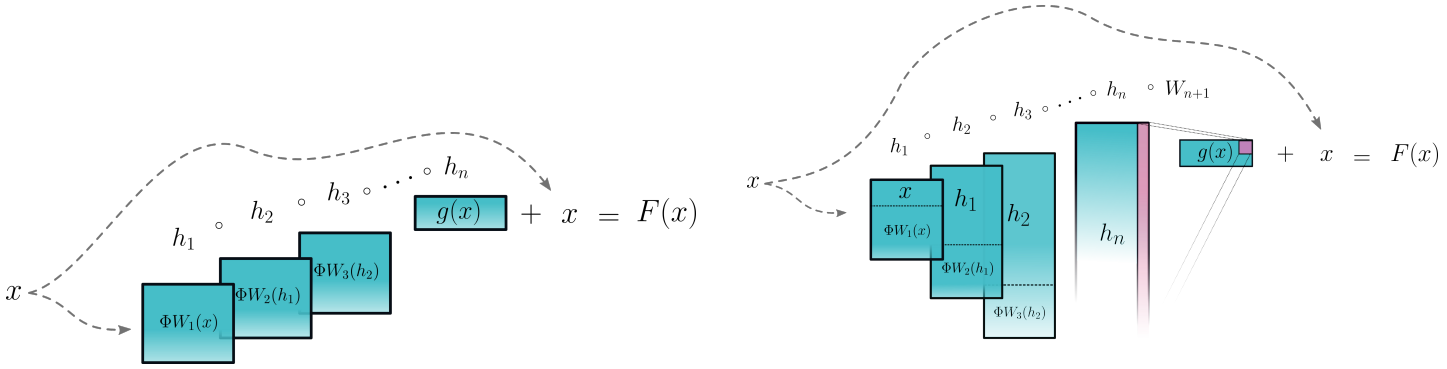
$$\text{Lip}(g) < 1, \quad \text{if} \quad ||W_i||_2 < 1, \tag{4.2.2}$$

where $||\cdot||_2$ is the ℓ_2 matrix norm. Then Lip(g) = K < 1 and Lip(F) < 1 + K. Only in this specific case, the Banach fixed-point theorem holds, and the ResNet layer F has a unique inverse. As a result, the inverse can be approximated by fixed-point iterations.

To estimate the log-determinant is, especially for high-dimensional spaces, computationally intractable due to expensive computations. Since ResNet blocks have a constrained Lipschitz constant, the log-likelihood estimation of Equation (4.2.1) can be transformed to a version where the logarithm of the Jacobian-determinant is cheaper to compute, tractable, and approximated with guaranteed convergence [11]:

$$\ln p(x) = \ln p(f(x)) + \text{tr}\left(\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} [J_g(x)]^k\right), \tag{4.2.3}$$

where $J_g(x)$ is the Jacobian of g at x that satisfies $||J_g||_2 < 1$. The Skilling-Hutchinson trace estimator [61, 119] is used to compute the trace at a lower cost than to fully



(a) Residual block

(b) Dense block

Figure 4.1.1: A schematic representation for: (a) a residual block, (b) a dense block. The pink part in (b) expresses a 1×1 convolution to reduce the dimension of the last dense layer. W_i denotes the (convolutional) layer at step i that satisfy $||W_i||_2 < 1$.

compute the trace of the Jacobian. Residual Flows [24] use an improved method to estimate the power series at an even lower cost with an unbiased estimator based on "Russian roulette" of [65]. Intuitively, the method estimates the infinite sum of the power series by evaluating a finite amount of terms. In return, this leads to less computation of terms compared to invertible residual networks. To avoid derivative saturation, which occurs when the second derivative is zero in large regions, the LipSwish activation is proposed.

4.3 INVERTIBLE DENSE NETWORKS

In this section, we propose Invertible Dense Networks by using a DenseBlock as a residual layer. We show how the network can be parameterized as a flow-based model and refer to the resulting model as i-DenseNets. The code can be retrieved from: https://github.com/yperugachidiaz/invertible_densenets.

4.3.1 *Dense blocks*

The main component of the proposed flow-based model is a DenseBlock that is defined as a function $F : \mathbb{R}^d \to \mathbb{R}^d$ with $F(x) = x + g(x)$, where g consists of n dense layers $\{h_i\}_{i=1}^n$. Note that an important modification to make the model invertible is to output $x + g(x)$, whereas a standard DenseBlock would only output $g(x)$. The function g is expressed as follows:

$$g(x) = W_{n+1} \circ h_n \circ \cdots \circ h_1(x), \tag{4.3.1}$$

where W_{n+1} represents a 1×1 convolution to match the output size of $\mathbb{R}^d$. A layer h_i consists of two parts concatenated to each other. The upper part is a copy of the input signal. The lower part consists of the transformed input, where the transformation is a multiplication of (convolutional) weights W_i with the input signal, followed by a non-linearity ϕ having $\text{Lip}(\phi) \leq 1$, such as ReLU, ELU, LipSwish, or tanh. As an example, a dense layer h_2 can be composed as follows:

$$h_1(x) = \begin{bmatrix} x \\ \phi(W_1 x) \end{bmatrix} \text{ and } h_2(h_1(x)) = \begin{bmatrix} h_1(x) \\ \phi(W_2 h_1(x)) \end{bmatrix}. \tag{4.3.2}$$

In Figure 4.1.1, we outline a residual block (Figure 4.1.1a) and a dense block (Figure 4.1.1b). We refer to concatenation *depth* as the number of dense layers in a DenseBlock and *growth* as the channel growth size of the transformation in the lower part.

4.3.2 *Constraining the Lipschitz constant*

If we enforce function g to satisfy $\text{Lip}(g) < 1$, then DenseBlock F is invertible since the Banach fixed point theorem holds. As a result, the inverse can be approximated in the same manner as in [11]. To satisfy $\text{Lip}(g) < 1$, we need to enforce $\text{Lip}(h_i) < 1$ for all n layers, since $\text{Lip}(g) \leq \text{Lip}(h_{n+1}) \cdot \ldots \cdot \text{Lip}(h_1)$. Therefore, we first need to determine the Lipschitz constant for a dense layer h_i. For the full derivation, see

Appendix B.1. We know that a function f is K-Lipschitz if for all points v and w the following holds:

$$d_Y(f(v), f(w)) \leq \mathrm{K} d_X(v, w), \tag{4.3.3}$$

where we assume that the distance metrics $d_X = d_Y = d$ are chosen to be the ℓ_2-norm. Further, let two functions f_1 and f_2 be concatenated in h:

$$h_v = \begin{bmatrix} f_1(v) \\ f_2(v) \end{bmatrix}, \quad h_w = \begin{bmatrix} f_1(w) \\ f_2(w) \end{bmatrix}, \tag{4.3.4}$$

4

where function f_1 is the upper part and f_2 is the lower part. We can now find an analytical form to express a limit on K for the dense layer in the form of Equation (4.3.3):

$$\begin{aligned} d(h_v, h_w)^2 &= d(f_1(v), f_1(w))^2 + d(f_2(v), f_2(w))^2, \\ d(h_v, h_w)^2 &\leq (\mathrm{K}_1^2 + \mathrm{K}_2^2) d(v, w)^2, \end{aligned} \tag{4.3.5}$$

where we know that the Lipschitz constant of h consist of two parts, namely, $\mathrm{Lip}(f_1) = \mathrm{K}_1$ and $\mathrm{Lip}(f_2) = \mathrm{K}_2$. Therefore, the Lipschitz constant of layer h can be expressed as:

$$\mathrm{Lip}(h) = \sqrt{\mathrm{K}_1^2 + \mathrm{K}_2^2}. \tag{4.3.6}$$

With spectral normalization of Equation (4.2.2), we know that we can enforce (convolutional) weights W_i to be at most 1-Lipschitz. Hence, for all n dense layers, we apply the spectral normalization on the lower part, which locally enforces $\mathrm{Lip}(f_2) = \mathrm{K}_2 < 1$. Further, since we enforce each layer h_i to be at most 1-Lipschitz and we start with h_1, where $f_1(x) = x$, we know that $\mathrm{Lip}(f_1) = 1$. Therefore, the Lipschitz constant of an entire layer can be at most $\mathrm{Lip}(h) < \sqrt{1^2 + 1^2} = \sqrt{2}$, thus dividing by this limit enforces each layer to be at most 1-Lipschitz.

4.3.3 *Learnable weighted concatenation*

We have shown that we can enforce an entire dense layer to have $\mathrm{Lip}(h_i) < 1$ by applying a spectral norm on the (convolutional) weights W_i and then divide the layer h_i by $\sqrt{2}$. Although learning a weighing between the upper and lower part would barely affect a standard dense layer, it matters in this case because the layers are regularized to be 1-Lipschitz. To optimize and learn the importance of the concatenated representations, we introduce learnable parameters η_1 and η_2 for the upper and lower part of each layer h_i, respectively.

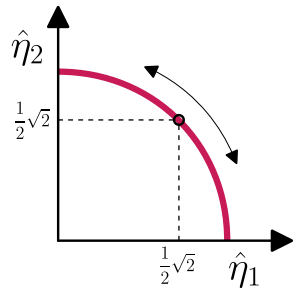


Figure 4.3.1: Range of the possible normalized parameters $\hat{\eta}_1$ and $\hat{\eta}_2$.

Since the upper and lower part of the layer can be at most 1-Lipschitz, multiplication by these factors results in functions that are at most η_1-Lipschitz and η_2-Lipschitz. As indicated by Equation (4.3.6), the layer is

then at most $\sqrt{\eta_1^2+\eta_2^2}$−Lipschitz. Dividing by this factor results in a bound that is at most 1-Lipschitz.

In practice, we initialize η_1 and η_2 at value 1 and, during training, use a softplus function to avoid them being negative. The range of the normalized parameters is between $\hat{\eta}_1, \hat{\eta}_2 \in [0,1]$ and can be expressed on the unit circle as shown in Figure 4.3.1. In the special case where $\eta_1 = \eta_2$, the normalized parameters are $\hat{\eta}_1 = \hat{\eta}_2 = \frac{1}{2}\sqrt{2}$. This case corresponds to the situation in Section 4.3.2 where the concatenation is not learned. An additional advantage is that the normalized $\hat{\eta}_1$ and $\hat{\eta}_2$ express the importance of the upper and lower signal. For example, when $\hat{\eta}_1 > \hat{\eta}_2$, the input signal is of more importance than the transformed signal.

4.3.4 *CLipSwish*

When a deep neural network is bounded to be 1-Lipschitz, in practice, each consecutive layer reduces the Jacobian norm. As a result, the Jacobian norm of the entire network is becoming much smaller than 1, and the expressive power is getting lost. This is known as *gradient norm attenuation* [5, 86]. This problem arises in activation functions in regions where the derivative is small, such as the left tail of the ReLU and the LipSwish. Non-linearities ϕ modeled in i-DenseNets are required to be at most 1-Lipschitz and thus face gradient-norm attenuation issues. For this reason, we are introducing a new activation function, which mitigates these issues.

Recall that Residual Flows use the LipSwish activation function [24]:

$$\text{LipSwish}(x) = x\sigma(\beta x)/1.1, \tag{4.3.7}$$

where $\sigma(\beta x) = 1/(1+\exp(-x\beta))$ is the sigmoid, β is a learnable constant, initialized at 0.5 and is passed through a softplus to be strictly positive. This activation function is not only Lip(LipSwish) = 1, but also resolves the derivative saturation problem [24]. However, the LipSwish function has large ranges on the negative axis where its derivative is close to zero.

Therefore, we propose the Concatenated LipSwish (CLipSwish) which concatenates two LipSwish functions with inputs x and $-x$. This is a concatenated activation function as in [115] but using a LipSwish instead of a ReLU. Intuitively, even if an input lies in the tail of the upper part, it will have a larger derivative in the bottom part and thus suffer less from gradient norm attenuation. Since using CLipSwish increases the channel growth and stays in line with the channel growth that non-concatenated activation functions use, we use a lower channel growth when using CLipSwish. To utilize the CLipSwish, we need to derive Lipschitz continuity of the activation function Φ defined below and enforce it to be 1-Lipschitz. We could use the result obtained in Equation (4.3.6) to obtain a $\sqrt{2}$-bound. However, by using knowledge about the activation function Φ, we can derive a tighter $1.004 < \sqrt{2}$ bound. A tighter bound is generally preferred since more expressive power will be preserved in the network. To start with, we define function $\Phi : \mathbb{R} \to \mathbb{R}^2$ for a point x as:

$$\Phi(x) = \begin{bmatrix}\phi_1(x)\\ \phi_2(x)\end{bmatrix} = \begin{bmatrix}\text{LipSwish}(x)\\ \text{LipSwish}(-x)\end{bmatrix}, \quad \text{CLipSwish}(x) = \Phi(x)/\text{Lip}(\Phi), \tag{4.3.8}$$

where the LipSwish is given by Equation (4.3.7) and the derivative of $\Phi(x)$ exists. To find $\text{Lip}(\Phi)$, we use that for a differentiable ℓ_2-Lipschitz bounded function Φ, the following identity holds:

$$\text{Lip}(\Phi) = \sup_x ||J_\Phi(x)||_2, \tag{4.3.9}$$

where $J_\Phi(x)$ is the Jacobian of Φ at x and $||\cdot||_2$ represents the induced matrix norm which is equal to the spectral norm of the matrix. Rewriting the spectral norm results in solving: $\det(J_\Phi(x)^T J_\Phi(x) - \lambda I_n) = 0$, which gives us the final result (see Appendix B.1 for the full derivation):

$$\sup_x ||J_\Phi(x)||_2 = \sup_x \sigma_{\max}(J_\Phi(x)) = \sup_x \sqrt{\left(\frac{\partial \phi_1(x)}{\partial x}\right)^2 + \left(\frac{\partial \phi_2(x)}{\partial x}\right)^2}, \tag{4.3.10}$$

where $\sigma_{\max}(\cdot)$ is the largest singular value. Now $\text{Lip}(\Phi)$ is the upper bound of the CLipSwish and is equal to the supremum of: $\text{Lip}(\Phi) = \sup_x ||J_\Phi(x)||_2 \approx 1.004$, for all values of β. This can be numerically computed by any solver, by determining the extreme values of Equation (4.3.10). Therefore, dividing $\Phi(x)$ by its upper bound 1.004 results in $\text{Lip}(\text{CLipSwish}) = 1$. The generalization to higher dimensions can be found in Appendix B.1. The analysis of the preservation of signals for (CLip)Swish activation by simulations can be found in Section 4.5.1.

4.4 EXPERIMENTS

To make a clear comparison between the performance of Residual Flows and i-DenseNets, we train both models on 2-dimensional toy data and high-dimensional image data: CIFAR10 [72], and ImageNet32 [28]. Since we have a constrained computational budget, we use smaller architectures for the exploration of the network architectures.

An in-depth analysis of different settings and experiments can be found in Section 4.5. For density estimation, we run the full model with the best settings for 1,000 epochs on CIFAR10 and 20 epochs on ImageNet32 where we use single-seed results following [11, 24, 69], due to little fluctuations in performance. In all cases, we use the density estimation results of the Residual Flow and other flow-based models using uniform dequantization to create a fair comparison and benchmark these with i-DenseNets. We train i-DenseNets with learnable weighted concatenation (LC) and CLipSwish as the activation function, and utilize a similar number of parameters for i-DenseNets as Residual Flows; this can be found in Table 4.4.1. i-DenseNets uses slightly fewer parameters than the Residual Flow. A detailed description of the smaller and full architectures can be found in Appendix B.2. To speed up training, we use 4 GPUs.

Table 4.4.1: The number of parameters of Residual Flows and i-DenseNets for the full models as trained in Chen et al. [24]. In brackets, the number of parameters of the smaller models.

Model/Data	CIFAR10	ImageNet32
Residual Flows	25.2M (8.7M)	47.1M
i-DenseNets	24.9M (8.7M)	47.0M

Table 4.4.2: Negative log-likelihood results on test data in nats (toy data). i-DenseNets w/ and w/o LC are compared with the Residual Flow.

Model	2 circles	checkerboard	2 moons
Residual Flows	3.44	3.81	2.60
i-DenseNets	3.32	3.68	**2.39**
i-DenseNets+LC	**3.30**	**3.66**	**2.39**

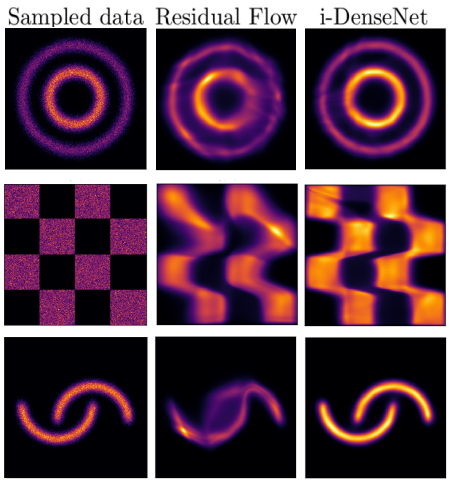


Figure 4.4.1: Density estimation for **smaller** architectures of Residual Flows and i-DenseNets, trained on 2-dimensional toy data.

4.4.1 *Toy data*

We start with testing i-DenseNets and Residual Flows on toy data, where we use smaller architectures. Instead of 100 flow blocks, we use 10 flow blocks. We train both models for 50,000 iterations and, at the end of the training, we visualize the learned distributions.

The results of the learned density distributions are presented in Figure 4.4.1. We observe that Residual Flows are capable of capturing high-probability areas. However, they have trouble with learning low-probability regions for two circles and moons. i-DenseNets are capable of capturing all regions of the datasets. The good performance of i-DenseNets is also reflected in better performance in terms of the negative-log-likelihood (see Table 4.4.2).

4.4.2 *Smaller models for MNIST & CIFAR10*

Due to the long run time and a constrained computational budget, we first experimented with smaller architectures of the i-DenseNet before utilizing the full models. This is presented in our first work [4], where we train smaller architectures of Residual Flow [24] and i-DenseNet on high-dimensional image data, MNIST and

CIFAR10. For i-DenseNet, we set the number of scales for the image data to 4 flow blocks per 3 scales instead of 16 blocks per 3 scales. For the other arguments, default settings are used. Note that the smaller i-DenseNet utilizes the LipSwish as an activation function instead of the CLipSwish, which we later found. To compare Residual Flows with i-DenseNets, we utilize an architecture that uses a similar number of parameters for each dataset trained on. A detailed description of this architecture can be found in Appendix B.2. Furthermore, we train with and without the option to learn the parameters of the concatenation. The models are trained for 200 epochs per dataset. During training on MNIST, the original Residual Flow suffered from unstable results. This might be due to the coefficient for the spectral normalization, which controls the Lipschitz constraint. In return, this leads to an unstable Jacobian determinant estimation. We adjusted the Lipschitz coefficient for the spectral normalization by setting it to 0.93 for all models. Additionally, the concatenation in DenseNets is multiplied by 0.98. Due to slight fluctuations, the results are averaged over the last 5 epochs. The results of the models trained on MNIST and CIFAR10 data are presented in Table 4.4.3, we observe that i-DenseNets outperform Residual Flows in bits per dimension (bpd) on CIFAR10 with 3.41 bpd without LC and 3.39 bpd with LC, against 3.42 bpd for the Residual Flow. We observe that i-DeseNets without and with LC outperform the Residual Flow with respectively 1.05 bpd and 1.04 bpd against 1.08 bpd of the Residual Flow. In general, we observe that i-DenseNets with LC outperform Residual Flows and i-DenseNets without LC. Furthermore, Table 4.4.4 presents results of the smaller i-DenseNet with LC and CLipSwish instead of LipSwish activation function. Comparing the results, we see how the activation function CLipSwish boosts performance with 3.37 bpd compared to 3.39 bpd with LipSwish.

Table 4.4.3: Density estimation results in bits per dimension on MNIST and CIFAR10, for smaller architectures of i-DenseNets w/ and w/o LC and Residual Flows.

Model	MNIST	CIFAR10
Residual Flow	1.08	3.42
Invertible DenseNet	1.05	3.41
Invertible DenseNet+LC	**1.04**	**3.39**

4.4.3 *Density estimation*

We test the full i-DenseNet models with LC and CLipSwish activation. To utilize a similar number of parameters as the Residual Flow with 3 scale levels and flow blocks set to 16 per scale trained on CIFAR10, we set for the same number of blocks, DenseNets growth to 172 with a depth of 3. Residual Flow trained on ImageNet32 uses 3 scale levels with 32 flow blocks per scale. Therefore, we set for the same number of blocks DenseNets growth to 172 and depth of 3 to utilize a similar number of parameters. DenseNets depth set to 3 proved to be the best setting for smaller architectures; see the analysis in Section 4.5.

The density estimation on CIFAR10 and ImageNet32 are benchmarked against the

results of Residual Flows and other comparable flow-based models, where the results are retrieved from Chen et al. [24]. We measure performances in bits per dimension (bpd). The results can be found in Table 4.4.4. We find that i-DenseNets outperform Residual Flows and other comparable flow-based models on all considered datasets in terms of bpd. On CIFAR10, i-DenseNet achieves 3.25bpd, against 3.28bpd of the Residual Flow. On ImageNet32 i-DenseNet achieves 3.98bpd against 4.01bpd of the Residual Flow. Samples of the i-DenseNet models can be found in Figure 4.4.2. Samples of the model trained on CIFAR10 are presented in Figure 4.4.2b and samples of the model trained on ImageNet32 in Figure 4.4.2d. For more unconditional samples, see Appendix B.3. Note that this work does not compare against flow-based models using variational dequantization. Instead, we focus on extending and making a fair comparison with Residual Flows, which, similar to other flow-based models, use uniform dequantization. For reference, note that Flow++ [57] with variational dequantization obtains 3.08bpd on CIFAR10 and 3.86bpd on ImageNet32, which is better than the model with uniform dequantization, which achieves 3.29bpd on CIFAR10.

Table 4.4.4: Density estimation results in bits per dimension for models using **uniform dequantization**. In brackets results for the smaller Residual Flow and i-DenseNet run for 200 epochs.

Model	CIFAR10	ImageNet32
Real NVP [36]	3.49	4.28
Glow [69]	3.35	4.09
FFJORD [48]	3.40	-
Flow++ [57]	3.29	-
ConvSNF [58]	3.29	-
i-ResNet [11]	3.45	-
Residual Flow [24]	3.28 (3.42)	4.01
i-DenseNet	**3.25** (**3.37**)	**3.98**

4.4.4 *Hybrid modeling*

Besides density estimation, we also experiment with hybrid modeling [98]. We train the joint distribution $p(\mathbf{x}, y) = p(\mathbf{x})\ p(y|\mathbf{x})$, where $p(\mathbf{x})$ is modeled with a generative model and $p(y|\mathbf{x})$ is modeled with a classifier, which uses the features of the transformed image onto the latent space. Due to the different dimensionalities of y and $\mathbf{x}$, the emphasis of the likelihood objective is more likely to be focused on $p(\mathbf{x})$ and a scaling factor for a weighted maximum likelihood objective is suggested, $\mathbb{E}_{x,y\sim\mathcal{D}}\left[\log p(y|\mathbf{x}) + \lambda \log p(\mathbf{x})\right]$, where λ is the scaling factor expressing the trade-off between the generative and discriminative parts. Unlike [98] where a linear layer is integrated on top of the latent representation, we use the architecture of [24] where the set of features are obtained after every scale level. Then, they are concatenated and are followed by a linear softmax classifier. We compare our experiments with the results of [24] where Residual Flow, coupling blocks [35] and 1×1 convolutions [69] are evaluated.

Table 4.4.5: Results of hybrid modeling on CIFAR10. Arrows indicate if low or high values are of importance. Results are averaged over the last five epochs.

	$\lambda = 0$	$\lambda = \frac{1}{D}$		$\lambda = 1$	
Model \Evaluation	Acc ↑	Acc ↑	bpd ↓	Acc ↑	bpd ↓
Coupling	89.77%	87.58%	4.30	67.62%	3.54
+ 1×1 conv	90.82%	87.96%	4.09	67.38%	3.47
Residual Blocks (full)	91.78%	90.47%	3.62	70.32%	3.39
Dense Blocks (full)	**92.40%**	**90.79%**	**3.49**	**75.67%**	**3.31**

4

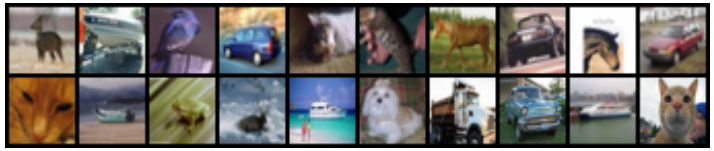
Table 4.4.5 presents the hybrid modeling results on CIFAR10, where we used $\lambda = \{0, \frac{1}{D}, 1\}$. We ran the three models for 400 epochs and noted that the model with $\lambda = 1$ was not fully converged in both accuracy and bits per dimension after training. The classifier model obtains a converged accuracy after around 250 epochs. This is in line with the accuracy for the model with $\lambda = \frac{1}{D}$, yet based on bits per dimension, the model was not fully converged after 400 epochs. This indicates that even though the accuracy is not further improved, the model keeps optimizing the bits per dimension, which gives room for future research. Results in Table 4.4.5 show the average result over the last 5 epochs. We find that Dense Blocks outperform Residual Blocks for all possible λ settings. Interestingly, Dense Blocks have the biggest impact using no penalty ($\lambda = 1$) compared to the other models. We obtain an accuracy of 75.67% with 3.31bpd, compared to 70.32% accuracy and 3.39bpd of Residual Blocks, indicating that Dense Blocks significantly improve classification performance with more than 5%. In general, the Dense Block hybrid model is outperforming Real NVP, Glow, FFJORD, and i-ResNet in bits per dimension (see Appendix B.3 for samples of the hybrid models).

4.5 ANALYSIS AND FUTURE WORK

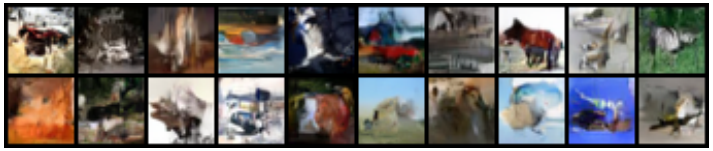
To get a better understanding of i-DenseNets, we perform additional experiments, explore different settings, analyze the results of the model and discuss future work. We use smaller architectures for these experiments due to a limited computational budget. For Residual Flows and i-DenseNets we use 3 scale levels set to 4 Flow blocks instead of 16 per scale level and train models on CIFAR10 for 200 epochs. We will start with a short explanation of the limitations of 1-Lipschitz deep neural networks.

Table 4.5.1: The mean and maximum ratio for different dimensions with sample size set to 100,000.

Activation\ Measure	$D = 1$		$D = 128$		$D = 1024$	
	Mean	Max	Mean	Max	Mean	Max
Sigmoid	0.22	0.25	0.21	0.22	0.21	0.21
LipSwish	0.46	1.0	0.51	0.64	0.51	0.55
CLipSwish	0.72	1.0	0.71	0.77	0.71	0.73
Identity	1.0	1.0	1.0	1.0	1.0	1.0



(a) Real CIFAR10 images.



(b) Samples of i-DenseNets trained on CIFAR10.

(c) Real ImageNet32 images.

(d) Samples of i-DenseNets trained on ImageNet32.

Figure 4.4.2: Real and samples of CIFAR10 and ImageNet32 data.

4.5.1 *Analysis of activations and preservation of signals*

Since gradient-norm attenuation can arise in 1-Lipschitz bounded deep neural nets, we analyze how much signal of activation functions is preserved by examining the maximum and average distance ratios of sigmoid, LipSwish, and CLipSwish. Note that the maximum distance ratio approaches the Lipschitz constant and it is desired that the average distance ratio remains high.

We sample 100,000 datapoints $v, w \sim \mathcal{N}(0,1)$ with dimension set to $D = \{1, 128, 1024\}$. We compute the mean and maximum of the sampled ratios with:

$$\frac{\ell_2(\phi(v), \phi(w))}{\ell_2(v, w)}, \tag{4.5.1}$$

and analyze the expressive power of each function. Table 4.5.1 shows the results. We find that CLipSwish for all dimensions preserves most of the signal on average compared to the other non-linearities. This may explain why i-DenseNets with CLipSwish activation achieves better results than using, e.g., LipSwish. This experiment indicates that on randomly sampled points, CLipswish functions suffer from considerably less gradient norm attenuation. Note that sampling from a distribution with larger parameter values is even more pronounced in preference of CLipSwish, see Appendix B.4.

4.5.2 *Activation functions*

We start with exploring different activation functions for both networks and test these with the smaller architectures. We compare our CLipSwish to the LipSwish and the LeakyLSwish as an additional baseline, which allows freedom of movement in the left tail as opposed to a standard LipSwish:

$$\text{LeakyLSwish}(x) = \alpha x + (1 - \alpha)\text{LipSwish}(x), \tag{4.5.2}$$

with $\alpha \in (0,1)$ by passing it through a sigmoid function σ. Here α is a learnable parameter which is initialized at $\alpha = \sigma(-3)$ to mimic the LipSwish at initialization. Note that the dimension for Residual Flows with CLipSwish activation function is set to 652 instead of 512 to maintain a similar number of parameters (8.7M) as with LipSwish activation.

Table 4.5.2 shows the results of each model using different activation functions. With 3.37bpd we conclude that i-DenseNet with our CLipSwish as the activation function obtains the best performance compared to the other activation functions, LipSwish and LeakyLSwish. Furthermore, all i-DenseNets outperform Residual Flows with the same activation function. We want to point out that CLipSwish as the activation function not only boosts the performance of i-DenseNets but it also significantly improves the performance of Residual Flows with 3.38bpd. The running time for the forward pass, train time, and sampling time, expressed in percentage faster or slower than Residual Flow with the same activation functions, can be found in Appendix B.4.

Next, we examine the effect of different concatenation depth settings for i-DenseNets. We run experiments with concatenation depth set to 2, 3, 4, and 5 with CLipSwish. Furthermore, to utilize 8.7M parameters of the Residual Flow, we choose a fixed depth and appropriate DenseNet growth size to have a similar number of parameters. This results in a DenseNet depth 2 with a growth size of 318 (8.8M), depth 3 with a growth of 178 (8.7M), depth 4 with a growth of 122 (8.7M), and depth 5 with a growth of 92 (8.8M). The effect of each architecture can be found in Figure 4.5.1. We observe that the model with a depth of 3 obtains the best scores, and after 200 epochs, it achieves the lowest bits per dimension with 3.37bpd. A concatenation depth of 5 results in 3.42bpd after 200 epochs, which is the least preferred. This could indicate that the corresponding DenseNet growth of 92 is too little to capture the density structure sufficiently, and due to the deeper depth, the network might lose important signals.

4.5.3 *DenseNets concatenation depth*

Further, the figure clearly shows how learnable weighted concatenation after 25 epochs boosts training for all i-DenseNets. See Appendix B.4 for an in-depth analysis of the results of the learnable weighted concatenation. Furthermore, we performed an additional experiment (see Appendix B.4) where we extended the width and depth of the ResNet connections in $g(x)$ of Residual Flows in such a way that it matches the size of the i-DenseNet. As a result, on CIFAR10, this puts the extended Residual Flow at a considerable advantage as it utilizes 19.1M parameters instead of 8.7M. However, when looking at performance the model performs worse (7.02bpd) than i-Densenets (3.39bpd) and even worse than its original version (3.42bpd) in terms of bpd. A possible explanation of this phenomenon is that by forcing more convolutional layers to be 1-Lipschitz, the gradient norm attenuation problems increase, and in practice, they become considerably less expressive. This indicates that modeling a DenseNet in $g(x)$ is indeed an important difference that gives better performance.

4.5.4 *Future work*

We introduced a new framework, i-DenseNet, that is inspired by Residual Flows and i-ResNets. We demonstrated how i-DenseNets out-performs Residual Flows and alternative flow-based models for density estimation and hybrid modeling, constraint by using uniform dequantization. For future work, we want to address several interesting aspects we came across and where i-DenseNets may be further deployed and explored.

Table 4.5.2: Results in bits per dimensions for small architectures, testing different activation functions.

Model	LipSwish	LeakyLSwish	CLipSwish
Residual Flow	3.42	3.42	3.38
i-DenseNet	**3.39**	**3.39**	**3.37**

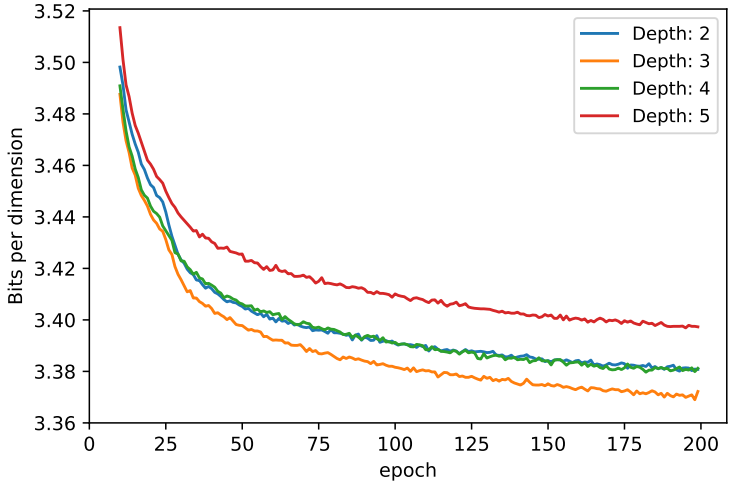


Figure 4.5.1: Effect of different concatenation depths with CLipSwish activation function for i-DenseNets in bits per dimension.

First, we find that smaller architectures have more impact on performance than full models compared to Residual Flows. Especially for the exploration of the network, we recommend experimenting with smaller architectures or when a limited computational budget is available. This brings us to the second point. Due to a limited budget, we trained and tested i-DenseNets on 32×32 CIFAR10 and ImageNet32 data. It will be interesting to test higher resolution and other types of datasets.

Further exploration of DenseNets depth and growth for other or higher-resolution datasets may be worthwhile. In our studies, deeper DenseNets did not result in better performance. However, it would also be beneficial to further examine the optimization of DenseNets architectures. Similarly, we showed how to constrain DenseBlocks for the ℓ_2-norm. For future work, it may be interesting to generalize the method to different norm types, as well as the norm for CLipSwish activation function. Note that CLipSwish as an activation function not only boosts the performance of i-DenseNets but also for Residual Flows. We recommend this activation function for future work.

We want to stress that we focused on extending Residual Flows, which uses uniform dequantization. However, we believe that the performance of our network may be improved using variational dequantization or augmentation. Finally, we found that especially hybrid model with $\lambda = 1$, achieve better performance than its predecessors. This may be worthwhile to further investigate in the future.

4.6 SOCIETAL IMPACT

We discussed methods to improve normalizing flow, a method that learns high-dimensional distributions. We generated realistic-looking images and also used

hybrid models that both predict the label of an image and generate new ones. Besides generating images, these models can be deployed to, e.g., detect adversarial attacks. Additionally, this method is applicable to all different kinds of fields, such as chemistry or physics. An increasing concern is that generative models in general, have an impact on society. They can not only be used to aid society but can also be used to generate misleading information by those who use these models. Examples of these cases could be generating real-looking documents, Deepfakes or even detection of fraud with the wrong intentions. Even though current flow-based models are not there yet to generate flawless reproductions, this concern should be kept in mind. It even raises the question of whether these models should be used in practice when the detection of misleading information becomes difficult or even impossible to track.

4.7 CONCLUSION

In this chapter, we proposed i-DenseNets, a parameter-efficient alternative to Residual Flows. Our method enforces invertibility by satisfying the Lipschitz continuity in dense layers. In addition, we introduced a version where the concatenation of features is learned during training that indicates which representations are of importance for the model. Furthermore, we showed how to deploy the CLipSwish activation function. For both i-DenseNets and Residual Flows, this significantly improves performance. Smaller architectures under an equal parameter budget were used for the exploration of different settings.

The full model for density estimation was trained on 32×32 CIFAR10 and ImageNet32 data. We demonstrated the performance of i-DenseNets and compared the models to Residual Flows and other comparable Flow-based models on density estimation in bits per dimension. Yet, it also demonstrated how the model could be deployed for hybrid modeling that includes classification in terms of accuracy and density estimation in bits per dimension. Furthermore, we showed that modeling ResNet connections matching the size of an i-DenseNet obtained worse performance than the i-DenseNet and the original Residual Flow. In conclusion, i-DenseNets outperform Residual Flows and other competitive flow-based models for density estimation on all considered datasets in bits per dimension and hybrid modeling that includes classification. The obtained results clearly indicate the high potential of i-DenseNets as powerful flow-based models.

B APPENDIX FOR I-DENSENET

B.1 *Derivations*

This section of the appendix provides the reader with full derivations of the Lipschitz constant for the concatenation in DenseNets and a bound of the Lipschitz for the activation functions.

Derivation of Lipschitz constant K for the concatenation

We know that a function f is K-Lipschitz if for all points v and w the following holds:

$$d_Y(f(v), f(w)) \leq \mathrm{K} d_X(v, w), \tag{B.1}$$

where d_Y and d_X are distance metrics and K is the Lipschitz constant.

Consider the case where we assume to have the same distance metric $d_Y = d_X = d$, and where the distance metric is assumed to be chosen as any p-norm, where $p \geq 1$, for vectors: $||\delta||_p = \sqrt[p]{\sum_{i=1}^{len(\delta)} |\delta_i|^p}$. Further, we assume a DenseBlock to be a function h where the output for each data point v and w is expressed as follows:

$$h_v = \begin{bmatrix} f_1(v) \\ f_2(v) \end{bmatrix}, \quad h_w = \begin{bmatrix} f_1(w) \\ f_2(w) \end{bmatrix}, \tag{B.2}$$

where in this chapter, for a Dense Layer and for a data point x, the function $f_1(x) = x$ and f_2 expresses a linear combination of (convolutional) weights with x followed by a non-linearity, for example, $\phi(W_1 x)$. We can rewrite Equation (B.1) for the DenseNet function as:

$$d(h_v, h_w) \leq \mathrm{K} d(v, w), \tag{B.3}$$

where K is the unknown Lipschitz constant for the entire DenseBlock. However, we can find an analytical form to express a limit on K. To solve this, we know that the distance between h_v and h_w can be expressed by the p-norm as:

$$d(h_v, h_w) = \sqrt[p]{\sum_{i=1}^{len(h_v)} |h_{v,i} - h_{w,i}|^p}, \tag{B.4}$$

where we can simplify the equation by taking the p-th power:

$$d(h_v, h_w)^p = \sum_{i=1}^{len(f_1(v))} |f_1(v)_i - f_1(w)_i|^p + \sum_{i=1}^{len(f_2(v))} |f_2(v)_i - f_2(w)_i|^p. \tag{B.5}$$

Since we know that the distance of f_1 can be expressed as:

$$d(f_1(v), f_1(w)) = \sqrt[p]{\sum_{i=1}^{len(f_1(v))} |f_1(v)_i - f_1(w)_i|^p}, \tag{B.6}$$

which is similar for the distance of f_2, re-writing the second term of Equation (B.5) in the form of Equation (B.3) is assumed to be of form:

$$d(f_1(v), f_1(w))^p \leq \mathrm{K}_1^p d(v, w)^p, \tag{B.7}$$

which is similar for f_2, $d(f_2(v), f_2(w))^p \leq K_2^p d(v,w)^p$. Assuming this, we can find a form of Equation (B.3) by substituting Equation (B.5) and Equation (B.7):

$$\begin{aligned} d(h_v, h_w)^p = \sum_i^{len(h_v)} |h_{v,i} - h_{w,i}|^p &\leq d(f_1(v), f_1(w))^p + d(f_2(v), f_2(w))^p \\ &= (K_1^p + K_2^p) d(v,w)^p. \end{aligned} \tag{B.8}$$

Now, taking the p-th root we have:

$$d(h_v, h_w) \leq \sqrt[p]{(K_1^p + K_2^p)}\, d(v,w), \tag{B.9}$$

where we have derived the form of Equation (B.3) and where $\text{Lip}(h) = K$ is expressed as:

$$\text{Lip}(h) = \sqrt[p]{(K_1^p + K_2^p)}, \tag{B.10}$$

where $\text{Lip}(f_1) = K_1$ and $\text{Lip}(f_2) = K_2$, which are assumed to be known Lipschitz constants.

Derivation bounded Lipschitz Concatenated ReLU

We define function $\phi : \mathbb{R} \to \mathbb{R}^2$ as the Concatenated ReLU for a point x:

$$\phi(x) = \begin{bmatrix} \text{ReLU}(x) \\ \text{ReLU}(-x) \end{bmatrix}. \tag{B.11}$$

Let points $v, w \in \mathbb{R}$. From Section B.1, Equation (B.4), we know that the distance between points transformed with ϕ and using the ℓ_2-norm can be written as:

$$\begin{aligned} d(\phi(v), \phi(w))^2 &= \sum_{i=1}^{len(\phi(v))} |\phi(v)_i - \phi(w)_i|^2 \\ &= (\phi(v)_1 - \phi(w)_1)^2 + (\phi(v)_2 - \phi(w)_2)^2 \\ &= (\text{ReLU}(v) - \text{ReLU}(w))^2 + (\text{ReLU}(-v) - \text{ReLU}(-w))^2. \end{aligned} \tag{B.12}$$

Furthermore, we know that the distance between the two points is:

$$\begin{aligned} d(v,w)^2 &= \sum_{i=1}^{len(v)} (v_i - w_i)^2 \\ &= (v-w)^2 \\ &= v^2 + w^2 - 2vw. \end{aligned} \tag{B.13}$$

We have four different situations that can happen. If $v > 0, w > 0$, then the distance between the points will be:

$$\begin{aligned} d(\phi(v), \phi(w))^2 &= (v-w)^2 + 0 \\ &= d(v,w)^2. \end{aligned} \tag{B.14}$$

In this specific case we have that $d(v,w)^2 = v^2 + w^2 - 2vw$, where $2vw > 0$. The same holds for $v \leq 0, w \leq 0$, when the first term becomes zero and instead of zero, the second term becomes $d(v,w)^2$ with $2vw \geq 0$.

If $v > 0, w \leq 0$, the distance between the points is equal to:

$$\begin{aligned} d(\phi(v),\phi(w))^2 &= (v-0)^2 + (0-w)^2 \\ &= v^2 + w^2 \\ &= (v-w)^2 + \underbrace{2vw}_{\leq 0} \leq (v-w)^2 = d(v,w)^2. \end{aligned} \tag{B.15}$$

4

The same derivation holds in the case $v \leq 0, w > 0$. Combining all cases, we find that $d(\phi(v),\phi(w)) \leq d(v,w)$, therefore:

$$\text{Lip}(\text{CReLU}) = 1. \tag{B.16}$$

Derivation Lipschitz bound of CLipSwish

We propose the Concatenated LipSwish (CLipSwish) and show how we can enforce the CLipSwish to be 1-Lipschitz for a 1-dimensional input signal x and generalization to a higher dimension in the upcoming subsections.

CLIPSWISH 1-DIMENSIONAL INPUT SIGNAL We derive the upper bound of Concatenated LipSwish and show that $\text{CLipSwish}(x) = \Phi(x)/1.004$ is enforced to satisfy $\text{Lip}(\text{CLipSwish}) = 1$. To start with, we define function $\Phi : \mathbb{R} \to \mathbb{R}^2$ for a point x as:

$$\Phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \end{bmatrix} = \begin{bmatrix} \text{LipSwish}(x) \\ \text{LipSwish}(-x) \end{bmatrix}, \tag{B.17}$$

where:

$$\text{LipSwish}(x) = x\sigma(\beta x)/1.1,$$

and the partial derivative of $\Phi(x)$ exists. Then the Jacobian matrix of Φ is well-defined as:

$$J_\Phi(x) = \begin{bmatrix} \frac{\partial \phi_1(x)}{\partial x} \\ \frac{\partial \phi_2(x)}{\partial x} \end{bmatrix}. \tag{B.18}$$

Furthermore, we know that for a ℓ_2-Lipschitz bounded function Φ, the following holds:

$$\text{Lip}(\Phi) = \sup_x ||J_\Phi(x)||_2, \tag{B.19}$$

where $J_\Phi(x)$ is the Jacobian of Φ and norm and $||\cdot||_2$ represents the induced matrix norm which is equal to the spectral norm of the matrix. Furthermore, we know that for a matrix A the following holds: $||A||_2 = \sigma_{max}(A)$, where σ_{max} is the largest singular value and the largest singular value is given by $\sigma_{max}(A) = \sqrt{\lambda_1}$, since $\sigma_i =$

$\sqrt{\lambda_i}$ for $i = 1, \ldots, n$ [77]. Now determining the singular values of $J_\Phi(x)$ is done by solving $\det(J_\Phi(x)^T J_\Phi(x) - \lambda I_n) = 0$. Combining and solving gives:

$$\begin{aligned} &\det(J_\Phi(x)^T J_\Phi(x) - \lambda I_n) = 0 \\ &\left[\left(\frac{\partial \phi_1(x)}{\partial x}\right)^2 + \left(\frac{\partial \phi_2(x)}{\partial x}\right)^2\right] - \lambda = 0 \\ &\lambda = \left(\frac{\partial \phi_1(x)}{\partial x}\right)^2 + \left(\frac{\partial \phi_2(x)}{\partial x}\right)^2 \end{aligned} \tag{B.20}$$

where $\lambda = \lambda_1$ the largest eigenvalue, thus: $\lambda_1 = \left(\frac{\partial \phi_1(x)}{\partial x}\right)^2 + \left(\frac{\partial \phi_2(x)}{\partial x}\right)^2$. Therefore, the spectral norm of Equation (B.19), can be re-written as:

$$||J_\Phi(x)||_2 = \sigma_{max}(J_\Phi(x)) = \sqrt{\left(\frac{\partial \phi_1(x)}{\partial x}\right)^2 + \left(\frac{\partial \phi_2(x)}{\partial x}\right)^2}. \tag{B.21}$$

Now Lip(Φ) is the upper bound of the CLipSwish and is equal to the supremum of: Lip(Φ) = $\sup_x ||J_\Phi(x)||_2 \approx 1.004$, for all values of β. This can be numerically computed by any solver by determining the extreme values of Equation (B.21).

GENERALIZATION TO HIGHER DIMENSIONS To generalize the Concatenated LipSwish activation function activation function to higher dimensions, we take $\Phi : \mathbb{R}^d \to \mathbb{R}^{2d}$, which represents the CLipSwish activation function for a vector $\mathbf{x} = \{x_1, x_2, \ldots, x_d\}$. Then the CLipSwish is given by the concatenation $\Phi(\mathbf{x}) = \big[\text{LipSwish}(\mathbf{x}), \text{LipSwish}(-\mathbf{x})\big]$, where $\phi_1(\mathbf{x}) = \text{LipSwish}(\mathbf{x})$ and $\phi_2(\mathbf{x}) = \text{LipSwish}(-\mathbf{x})$ elementwise. The Jacobian matrix $J_\Phi(\mathbf{x})$ with shape $2d \times d$, looks as follows:

$$J_\Phi(\mathbf{x}) = \begin{bmatrix} \frac{\partial \phi_1(\mathbf{x})_1}{\partial x_1} & \frac{\partial \phi_1(\mathbf{x})_1}{\partial x_2} & \cdots & \frac{\partial \phi_1(\mathbf{x})_1}{\partial x_d} \\ \vdots & \vdots & & \vdots \\ \frac{\partial \phi_1(\mathbf{x})_d}{\partial x_1} & \frac{\partial \phi_1(\mathbf{x})_d}{\partial x_2} & \cdots & \frac{\partial \phi_1(\mathbf{x})_d}{\partial x_d} \\ \frac{\partial \phi_2(\mathbf{x})_1}{\partial x_1} & \frac{\partial \phi_2(\mathbf{x})_1}{\partial x_2} & \cdots & \frac{\partial \phi_2(\mathbf{x})_1}{\partial x_d} \\ \vdots & \vdots & & \vdots \\ \frac{\partial \phi_2(\mathbf{x})_d}{\partial x_1} & \frac{\partial \phi_2(\mathbf{x})_d}{\partial x_2} & \cdots & \frac{\partial \phi_2(\mathbf{x})_d}{\partial x_d} \end{bmatrix}, \tag{B.22}$$

where $\frac{\partial \phi_{i,j}}{\partial x_k} = \begin{cases} 0, & \text{if } j \neq k \\ \frac{\partial \phi_{i,j}}{\partial x_k}, & \text{otherwise.} \end{cases}$

The determinant is computed as $\det(J_\Phi(\mathbf{x})^T J_\Phi(\mathbf{x}) - \lambda I_n) = 0$, where $J_\Phi(\mathbf{x})^T J_\Phi(\mathbf{x})$ is of shape $d \times d$ with off-diagonals equal to zero. Therefore, the determinant is given by multiplication of the diagonal entries and each eigenvalue is given by each diagonal entry. The general form of determinant and eigenvalues is written as:

$$\det(J_\Phi(\mathbf{x})^T J_\Phi(\mathbf{x}) - \lambda I_n) = \prod_{j=1}^{d} \lambda_j, \tag{B.23}$$

where each eigenvalue is given by:

$$\lambda_j = \left(\frac{\partial \phi_{1,j}}{\partial x_j}\right)^2 + \left(\frac{\partial \phi_{2,j}}{\partial x_j}\right)^2. \tag{B.24}$$

Then:

$$\begin{aligned} \text{Lip}(\Phi) &= \sup_x ||J_\Phi(x)||_2 \\ &= \sup_x \max_j \sqrt{\lambda_j} \\ &= \sup_x \max_j \sqrt{\left(\frac{\partial \phi_{1,j}}{\partial x_j}\right)^2 + \left(\frac{\partial \phi_{2,j}}{\partial x_j}\right)^2} \approx 1.004, \end{aligned} \tag{B.25}$$

where the last step is numerically approximated for the CLipSwish function, where ϕ is the LipSwish. Therefore, we plot Equation (B.25) in Python and compute the absolute extrema, which can be found in Figure B.1. For this figure we plotted CLipSwish with $\beta = 0.5$ and passed it through a softplus function, as it is initialized in the code on GitHub. Next, we can numerically obtain the absolute extrema by computing the maximum value and argmax of the maximum value of Equation (B.25), which respectively represent the y-coordinate and x-coordinate of the absolute maximum. This accounts for all β's being strictly positive since changing β does not change the y-coordinate of the extreme value but only shifts the x-coordinate more to or further away from the origin.

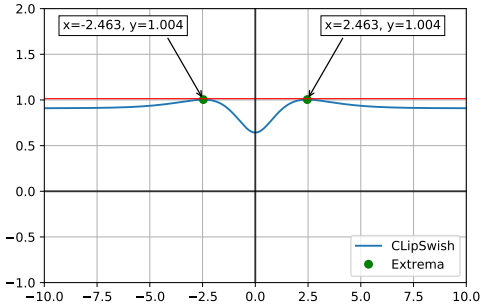


Figure B.1: ClipSwish activation function with indicated absolute maximums.

B.2 *Implementation*

This appendix provides implementation details for both the smaller and full models.

Architecture full models

For the full models, we followed the architecture of [24] and, during training, used a batch size of 64. We used the same datasets as in [24]: CIFAR10 and ImageNet32, to

Table B.1: The general DenseNet architecture for the smaller and full models, modeled in function g for image data.

Model	Nr. of scales	Nr. of blocks per scale	DenseNet Depth	DenseNet Growth	Dense Layer	Output
small	3	4 (CIFAR10) 4 (MNIST)	3	108 (CIFAR10) 124 (MNIST)	$\begin{bmatrix} 3\times 3 & \text{conv} \\ \text{LipSwish} \\ \text{concat} \end{bmatrix}$	$\begin{bmatrix} 1\times 1 & \text{conv} \end{bmatrix}$
full	3	16 (CIFAR10) 32 (ImageNet32)	3	172 (CIFAR10) 172 (ImageNet32)	$\begin{bmatrix} 3\times 3 & \text{conv} \\ \text{CLipSwish} \\ \text{concat} \end{bmatrix}$	$\begin{bmatrix} 1\times 1 & \text{conv} \end{bmatrix}$

4

make a fair comparison. CIFAR10 and ImageNet32 are of size 32×32. CIFAR10 contains 50,000 training images and 10,000 test images. ImageNet32 contains 1,281,167 training images and 50,000 validation images. CIFAR10 has an MIT License and the ImageNet terms of access can be found here: https://image-net.org/download.php. Before training, uniform dequantization is applied to the images after which a logit transformation is applied. For hybrid models, instead of the logit transform, the images use normalization $x = \frac{x-\mu}{\sigma}$. As in [24], for evaluation, at least 20 terms of the power series for the Jacobian-determinant are computed while the remaining terms to compute are determined by the unbiased estimator. Furthermore, we set a bound on the Lipschitz constant of each dense layer with a Lipschitz coefficient of 0.98. We use Adam optimizer with a learning rate set to 0.001 to train the models.

For all our models we ensured an equal parameter budget as the architecture of Residual Flows [24]. For CIFAR10, the full i-DenseNets utilize 24.9M to utilize the 25.2M of Residual Flows. For ImageNet32, i-DenseNet utilizes 47.0M parameters to utilize the 47.1M of the Residual Flow. A numerical architecture of the full i-DenseNets for image data is presented in Table B.1. g consists of several dense layers. The last dense layer h_n is followed by a 1×1 convolution to match the output of size $\mathbb{R}^d$, after which a squeezing layer is applied. The final part of the network consists of a Fully Connected (FC) layer with the number of blocks set to 4 for both datasets. Before the concatenation in the FC layer, a Linear layer of input $\mathbb{R}^d$ to output dimension 64 is applied, followed by the dense layer with for both datasets the FC DenseNet growth of 32, activation CLipSwish and a DenseNet depth of 3. The final part consists of a Linear layer to match the output of size $\mathbb{R}^d$. The large-scale models require approximately 410 seconds for an epoch on 4 NVIDIA TITAN RTX GPUs.

Architecture smaller models

We used a smaller architecture of Residual Flows [24], with an adjustment of number of blocks per scale set to 4 instead of 16. Additionally, for i-DenseNet we also utilized the LipSwish as activation function from [24]. For training, we ensured an equal parameter budget for i-DenseNets. The architecture of i-DenseNets for image data

are presented in Table B.1. For the MNIST dataset all models used 3 scales where the number of blocks per scale is set to 4. Due to the instability of Residual Flows, we set our coefficient that controls the Lipschitz constraint from 0.98 to 0.93. Furthermore, default settings of Residual Flows are used. For i-DenseNets, we used a coefficient controlling the Lipschitz of the concatenated blocks set to 0.98. i-DenseNets use a DenseNet depth and growth of, respectively, 3 and 108 with 5.0M parameters and Residual Flows utilize 5.0M parameters. For the CIFAR10 dataset, all models used 3 scales, with a set number of blocks per scale of 4. Furthermore, Residual Flows are used with default settings. i-DenseNets use a DenseNet depth and growth of 3 and 124, respectively, with 8.7M parameters, and Residual Flows utilize the 8.7M parameters.

B.3 *Samples*

This section of the appendix contains samples of the models trained on CIFAR10 and ImageNet32, along with samples of the hybrid models.

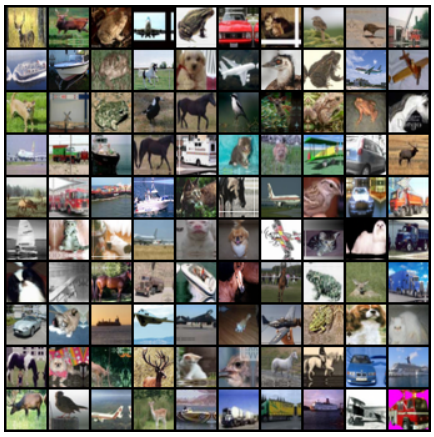
Model samples

Figure B.2 shows real images and samples of the models trained on CIFAR10 and ImageNet32. Figure B.2a shows the real images and Figure B.2b shows samples of i-DenseNet trained on CIFAR10. Figure B.2c shows the real images and Figure B.2d shows samples of i-DenseNet trained on ImageNet32.

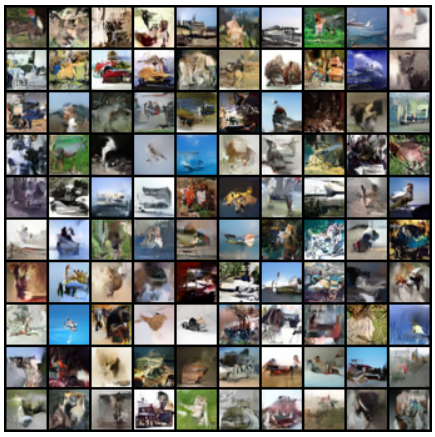
Hybrid modeling samples

Figure B.3 shows samples of the hybrid models trained on CIFAR10. The model trained with a scaling factor of $\lambda = \frac{1}{D}$ can be found in Figure B.3a. We notice that the samples tend to show a lot of red and brown colors and that the images tend to look noisy. This is probably due to the scaling factor where the generative part and classifier part have an equal focus for the likelihood objective, while there are $D = 32 \times 32$ features per image.

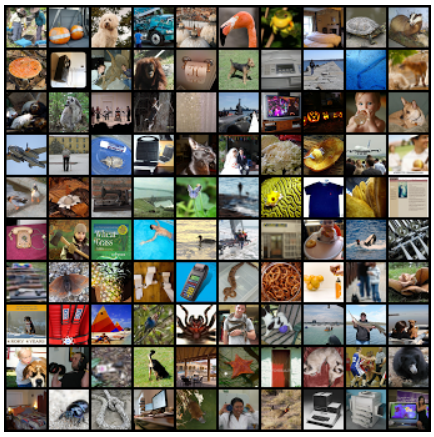
The model trained with $\lambda = 1$ can be found in Figure B.3b. The samples tend to look like the samples in Figure B.2b, only with less definition. This is probably due to the extra part, namely, the classifier part. Comparing the bits per dimension of the hybrid model with i-DenseNet trained for density estimation only, we find a difference of 0.06bpd.



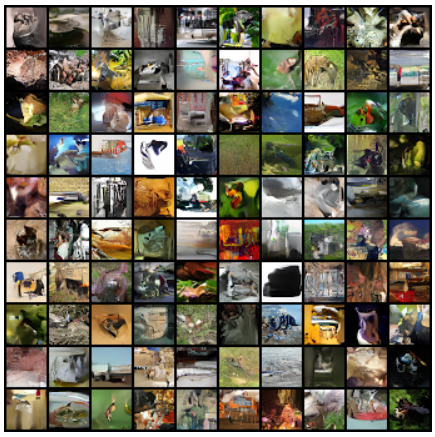
(a) Real images CIFAR10.



(b) Samples CIFAR10 i-DenseNet.



(c) Real images ImageNet32.

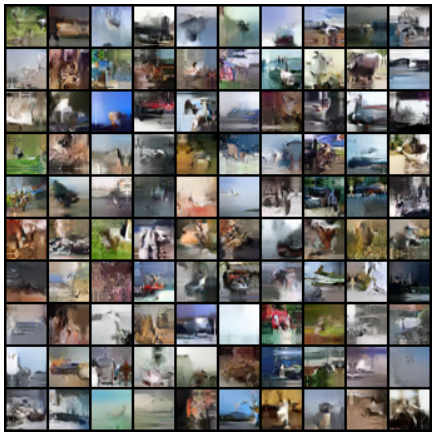


(d) Samples ImageNet32 i-DenseNet.

Figure B.2: Real images and samples from i-DenseNet trained on CIFAR10 and ImageNet32.

4

(a) Hybrid model trained with $\lambda = \frac{1}{D}$



(b) Hybrid model trained with $\lambda = 1$

Figure B.3: Hybrid modeling results with Dense Blocks trained on CIFAR10.

B.4 *Additional experiments*

In this appendix, we perform additional experiments. First, we analyze the preservation of signal for the activations functions with data points that are sampled from a distribution with larger parameter values. Furthermore, we analyze the running time of the models. Next, we examine the importance of the concatenated representation for i-DenseNets that are learned with learnable weighted concatenation. Finally, we analyze a Residual Flow where we extend the width and depth of the ResNet connections modeled in $g(x)$ such that it matches the size of i-DenseNet.

4

Preservation of signal

To further analyze the expressive power for the activation functions with a larger range, we sample 100,000 datapoints from distribution: $v, w \sim \mathcal{N}(0, 5)$ with dimension set to $D = \{1, 128, 1024\}$. We compute the mean and maximum of the sampled ratios with: $\ell_2(\phi(v), \phi(w)) / \ell_2(v, w)$ and analyze the expressive power of each function. Table B.2 shows the results. We find that CLipSwish for all dimensions preserves even more expressive power than data points sampled from $\mathcal{N}(0, 1)$, while sigmoid loses a considerable amount of signal with mean values close to zero instead of 0.25.

Table B.2: The mean and maximum ratio for different dimensions with sample size set to 100,000.

Activation\ Measure	$D = 1$		$D = 128$		$D = 1024$	
	Mean	Max	Mean	Max	Mean	Max
Sigmoid	0.09	0.25	0.08	0.10	0.08	0.09
LipSwish	0.47	1.0	0.54	0.69	0.54	0.59
CLipSwish	0.83	1.0	0.76	0.83	0.76	0.78
Identity	1.0	1.0	1.0	1.0	1.0	1.0

Running time

Table B.3 shows the forward pass, train time and sampling time, expressed in percentage faster or slower than Residual Flow, for each activation function. We find that the forward pass of i-DenseNet, for all activation functions, is faster than Residual Flow. The train time is slower and during sampling i-DenseNet is faster. Note that in comparison to the preliminary results in the rebuttal the times has changed somewhat, since these results have been obtained on a clean system with multiple runs. An interesting observation is that the LeakyLSwish with Residual Flows is much slower than the DenseNet variant, which indicates that fewer fixed-point iterations are needed for i-DenseNets to converge.

Table B.3: i-DenseNet approximate running times in percentage (%) compared to Residual Flow. Faster than Residual Flow is indicated with ↑ and slower ↓.

Activation Function	Forward pass (GPU)	Train time (GPU)	Sampling time (CPU)
LipSwish	↑ 1.3%	↓ 43%	↑ 8.8%
LeakyLSwish	↑ 1.3%	↓ 28%	↑ 231.9%
CLipSwish	↑ 0.6%	↓ 145%	↑ 11%

4

Importance of concatenated representation

Trained on CIFAR10, the smaller architecture with CLipSwish activation and a DenseNet depth of 3 and growth of 178, run for 200 epochs with CLipSwish obtains the best performance score with 3.37bpd. To analyze the importance of the concatenated representation after training, Figure B.4 shows the heatmap for parameter $\hat{\eta}_1$ (Figure B.4a) and parameter $\hat{\eta}_2$ (Figure B.4b). Every scale level 1, 2, and 3 contains 4 DenseBlocks, which each contains 3 dense layers with convolutional layers. The final level FC indicates that fully connected layers are used. The letters 'a', 'b', and 'c' index the dense layers per block.

Remarkably, all scale levels for the last layers h_{ic} give little importance to the input signal. The input signals for these layers are in most cases multiplied with $\hat{\eta}_1$ (close to) zero, while the transformed signal uses almost all the information when multiplied with $\hat{\eta}_2$, which is close to one. This indicates that the transformed signal is of more importance for the network than the input signal. For the fully connected part, this difference is not that pronounced. Instead of 4 DenseBlocks, the full i-DenseNet model utilizes 16 DenseBlocks (CIFAR10) and 32 (ImageNet32) for every scale; these are not included due to the size.

Matching architectures

The Residual Flow architecture with LipSwish activation and 3 scale levels set to 4 Flow blocks has 8.7M parameters. To utilize a similar number of parameters for i-DenseNet with LipSwish activation, we set DenseNets depth to 3 and growth to 124. To go a step further, we also examine modeling ResNet connections matching the size of i-DenseNet. Therefore, we use the same 3×3 kernels as each dense layer uses and, as a final layer, a 1×1 kernel to match the input size. Instead of concatenation, we use the growth size of 124 plus the input size to imitate the dense layers of i-DenseNet but then with convolutional connections. We repeat this process for the Fully Connected layer. Note that this puts the Residual Flow at a considerable advantage as it uses 19.1M parameters instead of the 8.7M of the original flow. We do the same experiment for toy data that uses only linear connections instead of convolutions.

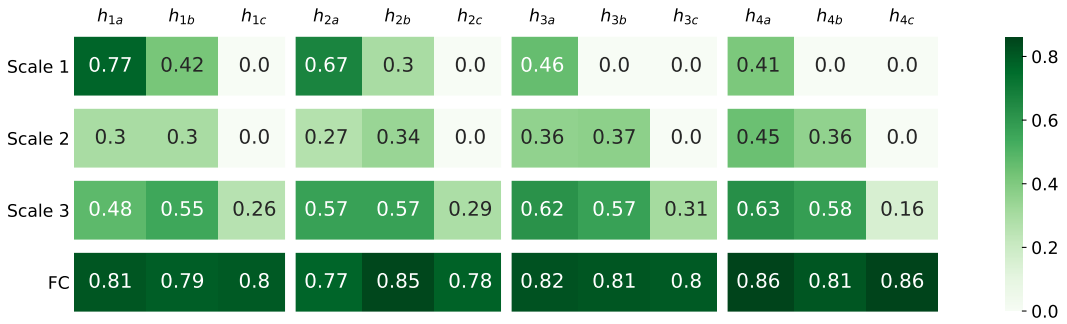
In Table B.4 the results are shown. On toy data, the extended Residual Flow performs slightly better in terms of nats compared to the original Residual Flow without

extended width and depth. Yet, i-DenseNet obtains the lowest (better) scores. On high-dimensional CIFAR10 data, the extended Residual Flow obtains 7.02bpd which is worse than i-DenseNet with 3.39bpd. Yet, the model also scores more than double as high (worse) in terms of bpd than the original Residual Flow with 3.42bpd.

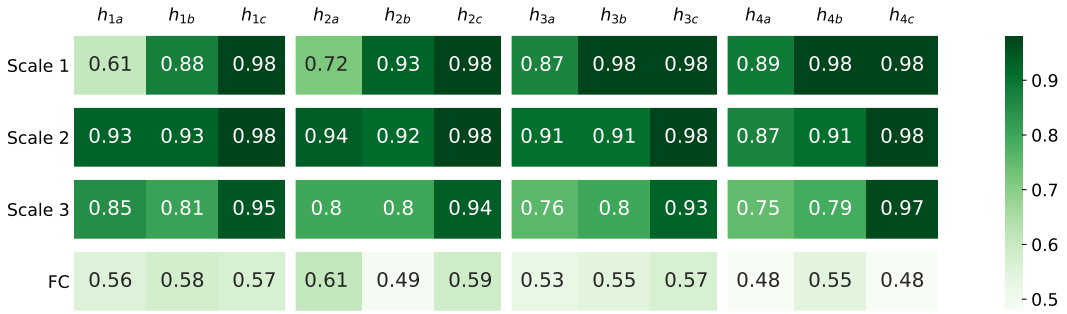
Table B.4: The negative log-likelihood results on test data in nats (toy data) and bpd (CIFAR10), where ↓ lower is better. i-DenseNets with LC are compared with the original Residual Flow and Residual Flow with equal width and depth as i-DenseNet.

Model (LipSwish)	CIFAR10 *bpd* ↓	2 circles *nats* ↓	checkerboard *nats* ↓	2 moons *nats* ↓
Residual Flows	3.42	3.44	3.81	2.60
+ extended width, equal depth	7.02	3.36	3.78	2.52
i-DenseNets+LC	**3.39**	**3.30**	**3.66**	**2.39**

The main difference in architecture of the toy and CIFAR10 is the linear layer for toy data whereas mainly convolutional layers are used for CIFAR10. A possible explanation of this phenomenon is that by forcing more convolutional layers to be 1-Lipschitz that *gradient norm attenuation* problems increase and in practice become less expressive. In conclusion, even though the model utilizes more than double the number of parameters, it performs worse than i-DenseNet with similar architecture and even worse than the original Residual Flow architecture, indicating that modeling a DenseNet in $g(x)$ indeed is an important difference that gives better performance.



(a) Heatmap of $\hat{\eta}_1$ corresponding to the input.



(b) Heatmap of $\hat{\eta}_2$ corresponding to the transformed input.

Figure B.4: Heatmaps of the normalized η_1 and η_2 after training for 200 epochs on CIFAR10. Best viewed electronically.

4

Part III

COMPRESSION

5

ROBUSTLY OVERFITTING LATENTS FOR FLEXIBLE NEURAL IMAGE COMPRESSION

Abstract

Although neural image compression models have proven to be successful in practice by out-performing traditional methods, they still lead to sub-optimal compression results due to imperfect optimization and limitations capacities. Recent work introduces a procedure that improves the compression performance for pre-trained neural image compression models while keeping the network weights fixed. This procedure uses the Stochastic Gumbel Annealing (SGA) method to refine the latents of pre-trained neural image compression models. We extend this idea by introducing SGA+, which contains three different methods that build upon SGA. Further, we give a detailed analysis of our proposed methods, show how they improve performance, and show that they are less sensitive to hyperparameter choices. Besides, we show how each method can be extended to three- instead of two-class rounding. Finally, we show how refinement of the latents with our best-performing method improves the compression performance on the Tecnick dataset and how it can be deployed to partly move along the rate-distortion curve.

Based on [1]:
Yura Perugachi-Diaz, Arwin Gansekoele, Sandjai Bhulai
Robustly overfitting latents for flexible neural image compression
Arxiv: 2401.17789 (2024)

5.1 INTRODUCTION

Neural image compression is an active and successful field [9, 82, 95]. In practice, these models outperform traditional methods [12, 120, 134]. Even though these models are promising, they are less practical for real-time applications due to the requirement of significant computational resources. Furthermore, these models have limited capacity for optimization. Due to the high-dimensional parameter space of a model that needs to be optimized and various hyperparameter settings that need to be set, they lead to challenges in optimization, which in return may lead to sub-optimal results.

Image compression allows efficient sending of an image between systems by reducing their size. There are two types of compression: *lossless* and *lossy*. Lossless image compression sends images perfectly without losing any quality and can thus be restored in their original format, such as the PNG format. Lossy compression, such as BPG [12], JPEG [134] or JPEG2000 [120], loses some quality of the compressed image. Lossy compression aims to preserve as much of the quality of the reconstructed image as possible, compared to its original format, while allowing a significantly larger reduction in required storage.

Traditional methods [120, 134], especially lossless methods, can lead to limited compression ratios or degradation in quality. With the rise of deep learning, neural image compression is becoming a popular method [126, 128]. In contrast with traditional methods, neural image compression methods have been shown to achieve higher compression ratios and less degradation in image quality [9, 82, 95]. Additionally, neural compression techniques have shown improvements compared to traditional codecs for other data domains, such as video. [2, 51, 91].

In practice, neural lossy compression methods have proven to be successful and achieve state-of-the-art performance [9, 82, 95]. These models are frequently based on variational autoencoders (VAEs) with an encoder-decoder structure [67]. The models are trained to minimize the expected rate-distortion (R-D) cost: $R + \lambda D$. Intuitively, one learns a mapping that encodes an image into a compressible latent representation. The latent representation is sent to a decoder and is decoded into a reconstructed image. The aim is to train the compression model in such way that it finds a latent representation that represents the best trade-off between the length of the bitstream for an image and the quality of the reconstructed image. Even though these models have proven to be successful in practice, they do have limited capacity when it comes to optimization and generalization. For example, the encoder's capacity is limited, which makes the latent representation sub-optimal [32]. Recent work [22, 49, 144] proposes procedures to refine the encoder or latents, which leads to better compression performance. Furthermore, in neural video compression, other work focuses on adapting the encoder [7, 90] or finetuning a full compression model after training to improve the video compression performance [132].

The advantage of refining latents [22, 144] is that improved compression results per image are achieved while the model does not need to be modified. Instead, the latent representations for each individual image undergo a refining procedure. This

results in a latent representation that obtains an improved bitstream and image quality over its original state from the pre-trained model. As mentioned in [144], the refining procedure for stochastic rounding with Stochastic Gradient Gumbel Annealing (SGA) considerably improves performance. In this chapter, we introduce SGA+, an extension of SGA that further improves compression performance and is less sensitive to hyperparameter choices. The main contributions are:

- We show how changing the probability space with more natural methods instead of SGA boosts the compression performance.
- We propose the sigmoid scaled logit (SSL), which can smoothly interpolate between the approximate atanh, linear, cosine, and round. We also show how this function finds even better compression performance.
- We demonstrate a generalization to rounding to three classes that contains the two classes as a special case.

Further, we show how SSL outperforms baselines on the Kodak dataset in terms of true loss curves. We show how our method generalizes to the Tecnick dataset in terms of peak signal-to-noise ratio (PSNR) versus the bits per pixel (BPP) in an R-D plot. In addition, we analyze the stability of all functions and show the effect of interpolation between different methods with SSL. Lastly, we propose a refining procedure at compression time that allows moving along the R-D curve when refining the latents with another λ than a pre-trained model is trained on. The code of our proposed methods is publicly available at: https://github.com/yperugachidiaz/flexible_neural_image_compression.

5.2 RELATED WORK

In lossy compression, the aim is to find a mapping of image x where the distortion of the reconstructed image $\hat{x}$ is as little as possible compared to the original one while using as little storage as possible. Therefore, training a lossy neural image compression model presents a trade-off between minimizing the length of the bitstream for an image and minimizing the distortion of the reconstructed image [8, 82, 95, 126].

Neural image compression models from [8, 95, 126], also known as hyperpriors, accomplish this kind of mapping with latent variables. An image x is encoded onto a latent representation $y = g_a(x)$, where $g_a(\cdot)$ is the encoder. Next, y is quantized $Q(y) = \hat{y}$ into a discrete variable that is sent losslessly to the decoder. The reconstructed image is given by: $\hat{x} = g_s(\hat{y})$, where $g_s(\cdot)$ represents the decoder. The rate-distortion objective that needs to be minimized for this specific problem is given by:

$$\begin{aligned}\mathcal{L} &= R + \lambda D \\ &= \underbrace{\mathbb{E}_{x\sim p_x}\left[-\log_2 p_{\hat{y}}(\hat{y})\right]}_{\text{rate}} + \lambda \underbrace{\mathbb{E}_{x\sim p_x}\left[d(x,\hat{x})\right]}_{\text{distortion}},\end{aligned} \tag{5.2.1}$$

where λ is a Lagrange multiplier determining the rate-distortion trade-off, R is the expected bitstream length to encode $\hat{y}$ and D is the metric to measure the distortion of the reconstructed image $\hat{x}$ compared to the original one x. Specifically for the rate, p_x is the (unknown) image distribution, and $p_{\hat{y}}$ represents the entropy model that is learned over the data distribution p_x. A frequently used distortion measure for $d(x,\hat{x})$, is the mean squared error (MSE) or PSNR.

In practice, the latent variable y often consists of multiple levels in neural compression. Namely, a smaller one named z, which is modeled with a relatively simple distribution $p(z)$, and a larger variable, which is modeled by a distribution for which the parameters are predicted with a neural network using z, the distribution $p(y|z)$. We typically combine these two variables into a single symbol y for brevity. Furthermore, a frequent method of quantizing $Q(\cdot)$ used to train hyperpriors consists of adding uniform noise to the latent variable.

5.2.1 *Latent optimization*

Neural image compression models have been trained over a huge set of images to find an optimal encoding. Yet, due to difficulties in optimization or due to constraints on the model capacity, model performance is sub-optimal. To overcome these issues, another type of optimizing compression performance is proposed in [22, 144], where they show how to find better compression results by utilizing pre-trained networks and keeping the encoder and decoder fixed but only adapting the latents. In these methods, a latent variable y is iteratively adapted using differentiable operations at test time. The aim is to find a more optimal discrete latent representation $\hat{y}$. Therefore, the following minimization problem needs to be solved for an image x:

$$\arg\min_{\hat{y}} \left[-\log_2 p_{\hat{y}}(\hat{y}) + \lambda d(x,\hat{x}) \right]. \tag{5.2.2}$$

This is a powerful method that can fit to a test image x directly without the need to train an entire compression model further.

5.2.2 *Stochastic Gumbel Annealing*

[22] proposes to optimize the latents by iteratively adding uniform noise and updating its latents. While this method proves to be effective, there is still a difference between the true rate-distortion loss ($\hat{\mathcal{L}}$) for the method and its discrete representation $\hat{y}$. This difference is also known as the *discretization gap*. Therefore, [144] propose the SGA method to optimize latents and show how it obtains a smaller discretization gap. SGA is a soft-to-hard quantization method that quantizes a continuous variable v into the discrete representation for which gradients can be computed. A variable v is quantized as follows. First, a vector $\mathbf{v}_r = (\lfloor v \rfloor, \lceil v \rceil)$ is created that stacks the floor and ceil of the variable, also indicating the rounding direction. Next, the variable v is centered between $(0,1)$ where for the flooring: $v_L = v - \lfloor v \rfloor$ and ceiling: $v_R = \lceil v \rceil - v$. With a temperature rate $\tau \in (0,1)$, that is decreasing over time, this variable determines the soft-to-hardness where 1 indicates training with a fully continuous variable v and 0 indicates training while fully rounding variable

v. To obtain unnormalized log probabilities, the inverse hyperbolic tangent (atanh) function is used as follows:

$$\log(\pi) = (-\text{atanh}(v_L)/\tau, -\text{atanh}(v_R)/\tau). \tag{5.2.3}$$

Next, samples are drawn $\mathbf{y} \sim$ Gumbel-Softmax(π, τ) [64] and are multiplied and summed with the vector $\mathbf{v}_r$ to obtain the quantized representation: $\hat{v} = \sum_i (v_{r,i} * y_i)$. As SGA aids the discretization gap, this method may not have optimal performance and may not be as robust to changes in its temperature rate τ.

Besides SGA, [144] propose deterministic annealing [1], which follows almost the same procedure as SGA, but instead of sampling stochastically from the Gumbel Softmax, this method uses a deterministic approach by computing probabilities with the Softmax from $\log(\pi)$. In practice, this method has been shown to suffer from unstable optimization behavior.

5.2.3 *Other methods*

While methods such as SGA aim to optimize the latent variables for neural image compression at inference time, other approaches have been explored in recent research. [50] proposed a soft-then-hard strategy alongside a learned scaling factor for the uniform noise to achieve better compression and a smoother latent. These methods are used to fine-tune network parameters but not the latents directly. [151] proposed using Swin-transformer-based coding instead of ConvNet-based coding. They showed that these transforms can achieve better compression with fewer parameters and shorter decoding times. [132] proposed to also fine-tune the decoder alongside the latent for video compression. While accommodating the additional cost of saving the model update, they demonstrated a gain of $\sim 1dB$. [147] and [39] proposed using implicit neural representations for video and image compression, respectively. [55] proposed an improved context model (SCCTX) and a change to the main transform (ELIC) that achieve strong compression results together. [40] revisited vector quantization for neural image compression and demonstrated it performs on par with hyperprior-based methods. While these approaches change the training process, our work differs in that we only consider the inference process.

5.3 METHODS

As literature has shown, refining the latents of pre-trained compression models with SGA leads to improved compression performance [144]. In this section, we extend SGA by introducing SGA+ containing three other methods for the computation of the unnormalized log probabilities $\log(\pi)$ to overcome issues from its predecessor. We show how these methods behave in probability space. Furthermore, we show how the methods can be extended to three-class rounding.

5.3.1 *Two-class rounding*

Recall from SGA that a variable v is quantized to indicate the rounding direction to two classes and is centered between (0,1). Computation of the unnormalized

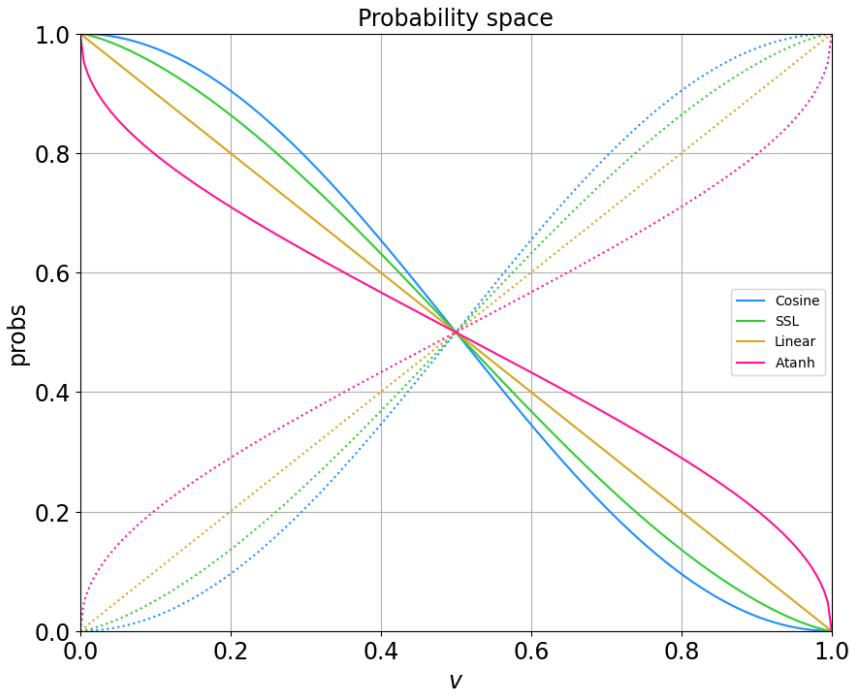


Figure 5.3.1: The probability space for two-class rounding for different functions of SGA+: linear, cosine and SSL ($a = \frac{4}{3}$), along with atanh. Solid lines denote the probability of flooring $\lfloor v \rfloor$ and dotted lines the probability of ceiling $\lceil v \rceil$.

log probabilities is obtained with atanh from Equation (5.2.3). When looking at the probability space from this function, see Figure 5.3.1, the atanh function can lead to sub-optimal performance when used to determine rounding probabilities. The problem is that gradients tend to infinity when the function approaches the limits of 0 and 1. This is not ideal, as these limits are usually achieved when the discretization gap is minimal. In addition, the gradients may become larger towards the end of optimization. Further analyzing the probability space, we find that there are a lot of possibilities in choosing probabilities for rounding to two classes. However, there are some constraints: the probabilities need to be monotonic functions, and the probabilities for rounding down (flooring) and up (ceiling) need to sum up to one. Therefore, we introduce SGA+ and propose three methods that satisfy the above constraints and can be used to overcome the sub-optimality that the atanh function suffers from.

We will denote the probability that v is rounded down by:

$$p(y = \lfloor v \rfloor), \tag{5.3.1}$$

where y represents the random variable whose outcome can be either rounded down

or up. The probability that v is rounded up is conversely: $p(y = \lceil v \rceil) = 1 - p(y = \lfloor v \rfloor)$.

LINEAR PROBABILITIES To prevent gradient saturation or vanishing gradients completely, the most natural case would be to model a probability that linearly increases or decreases and has a gradient of one everywhere. Therefore, we define the linear:

$$p(y = \lfloor v \rfloor) = 1 - (v - \lfloor v \rfloor). \tag{5.3.2}$$

It is easy to see that $p(y = \lceil v \rceil) = v - \lfloor v \rfloor$. In Figure 5.3.1, the linear probability is shown.

COSINE PROBABILITIES As can be seen in Figure 5.3.1, the atanh tends to have gradients that go to infinity for v close to the corners. Subsequently, a method that has low gradients in that area is by modeling the cosine probability as follows:

$$p(y = \lfloor v \rfloor) = \cos^2\left(\frac{(v - \lfloor v \rfloor)\pi}{2}\right). \tag{5.3.3}$$

This method aids the compression performance compared to the atanh since there is less probability of overshooting the rounding value.

SIGMOID SCALED LOGIT There are a lot of possibilities in choosing probabilities for two-class rounding. We introduced two probabilities that overcome suboptimality issues from atanh: the linear probability from Equation (5.3.2), which has equal gradients everywhere, and the cosine from Equation (5.3.3) that has little gradients at the corners. Besides these two functions, the optimal probability might follow a different function from the ones already mentioned. Therefore, we introduce the sigmoid scaled logit (SSL), which can interpolate between different probabilities with its hyperparameter a and is defined as follows:

$$p(y = \lfloor v \rfloor) = \sigma(-a\sigma^{-1}(v - \lfloor v \rfloor)), \tag{5.3.4}$$

where a is the factor determining the shape of the function. SSL resembles exactly the linear for $a = 1$. For $a = 1.6$ and $a = 0.65$ SSL roughly resembles the cosine and atanh. For $a \to \infty$ the function tends to shape to (reversed) rounding.

5.3.2 *Three-class rounding*

As described in the previous section, the values for v can either be floored or ceiled. However, there are cases where it may help to round to an integer further away. Therefore, we introduce three-class rounding and show three extensions that build on top of the linear probability Equation (5.3.2), cosine probability Equation (5.3.3), and SSL from Equation (5.3.4).

The probability that v is rounded is denoted by: $p(y = \lfloor v \rceil) \propto f_{3c}(w|r,n)$, where $w = v - \lfloor v \rceil$ is centered around zero. Further, we define the probability that v is rounded $+1$ and rounded -1 is respectively given by: $p(y = \lfloor v \rceil - 1) \propto f_{3c}(w-1|r,n)$

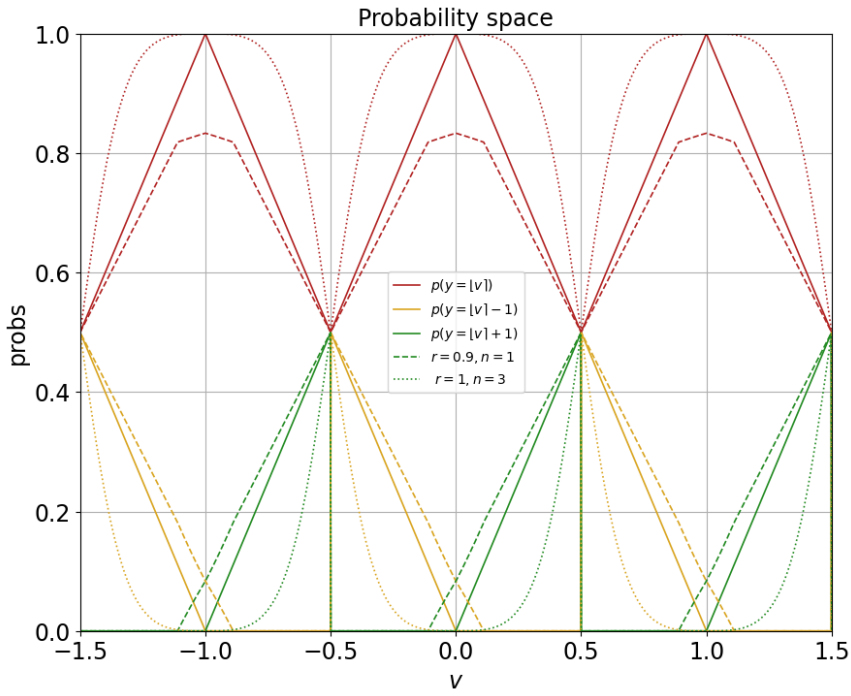


Figure 5.3.2: Three-class rounding is used for the extended version of the linear. Solid lines denote the two-class rounding with $r = 1$ and $n = 1$. Dashed lines denote three-class rounding with $r = 0.9$ and $n = 1$, and the dotted lines indicate the smoothness for $r = 1$ and $n = 3$.

and $p(y = \lfloor v \rceil + 1) \propto f_{3c}(w+1|r,n)$. The general notation for the three-class functions is given by:

$$f_{3c}(w|r,n) = f(\text{clip}(w \cdot r))^n, \tag{5.3.5}$$

where clip$(\cdot)$ clips the value at 0 and 1, r is the factor determining the height and steepness of the function, and power n controls the peakedness of the function.

EXTENDED LINEAR Recall that the linear probability can now be extended to three-class rounding as follows:

$$f_{linear}(w) = |w|. \tag{5.3.6}$$

A special case is $f_{3c,linear}(w|r = 1, n = 1)$, where the function is equivalent to the linear of the two-class rounding from Equation (5.3.2). For $r < 1$, this function rounds to three classes, and for $n \neq 1$, this function is not linear anymore.

In Figure 5.3.2, the three-class rounding for the extension of Equation (5.3.2) can be found. As can be seen, the solid lines denote the special case of two-class rounding with $r = 1$ and $n = 1$, the dashed lines denote three-class rounding with $r = 0.9$ and

$n = 1$ and the dotted lines denote the two-class rounding with $r = 1$ and $n = 3$, which shows a less peaked function. We can now compare an example of two- versus three-class rounding. Consider the case where we have variable $v = -0.95$. For two-class rounding there is only the chance of rounding to -1 with $p(y = \lfloor v \rceil)$ (red solid line), a chance to round to 0 with $p(y = \lfloor v \rceil + 1)$ (green solid line) and zero chance to round to -2 with $p(y = \lfloor v \rceil - 1)$ (yellow solid line). Now for three-class rounding, with $r = 0.9$ and $n = 1$, when $v = -0.95$, we find that there is a high chance to round to -1 with $p(y = \lfloor v \rceil)$ (red dashed line) and a small chance to round to 0 with $p(y = \lfloor v \rceil + 1)$ (green dashed line) and a tiny chance to round to -2 with $p(y = \lfloor v \rceil - 1)$ (yellow dashed line).

EXTENDED COSINE Similarly, we can transform the cosine probability from Equation (5.3.3) to three-class rounding:

$$f_{cosine}(w) = \cos\left(\frac{|w|\pi}{2}\right). \tag{5.3.7}$$

When $f_{3c,cosine}(w|r = 1, n = 2)$, this function exactly resembles the cosine for two-class rounding, and for $r < 1$ this function rounds to three classes.

EXTENDED SSL Additionally, SSL from Equation (5.3.4) can be transformed to three-class rounding as follows:

$$f_{SSL}(w) = \sigma\left(-a\sigma^{-1}(|w|)\right), \tag{5.3.8}$$

where a is the factor determining the shape of the function. When $f_{3c,SSL}(w|r = 1, n = 1)$, this function exactly resembles the two-class rounding case, and for $r < 1$, the function rounds to three classes. Recall that this function is capable of exactly resembling the linear function and approximates the cosine from two-class rounding for $a = 1$ and $a = 1.6$, respectively.

5.4 EXPERIMENTS

In this section, we evaluate our best-performing method and compare it to the baselines with the true R-D loss performance ($\hat{\mathcal{L}}$), the difference between the method loss and true loss ($\mathcal{L} - \hat{\mathcal{L}}$), and corresponding PSNR and BPP plots that respectively express the image quality and cost over t training steps. Next, an in-depth analysis of the stability of each proposed method is shown, followed by an experiment that expresses changes in the true R-D loss performance when one interpolates between functions. Additionally, we evaluate the three-class rounding for each of the proposed methods. Finally, we show how our best-performing method performs on the Tecnick dataset and how the method can be deployed to partly move along the rate-distortion curve.

Following [144], we run all experiments with temperature schedule:

$$\tau(t) = \min(\exp\{-ct\}, \tau_{max}), \tag{5.4.1}$$

where c is the temperature rate determining how fast temperature τ is decreasing over time, t is the number of train steps for the refinement of the latents and $\tau_{max} \in (0,1)$ determines how soft the latents start the refining procedure. Additionally, we refine the latents for $t = 2000$ train iterations, unless specified otherwise. For more hyperparameter settings, see Section 5.4.6. Further, the experimental results are obtained from a pre-trained model trained with $\lambda = 0.01$.

5.4.1 *Implementation details*

The pre-trained mean-scale hyperpriors are trained from scratch on the full-size CLIC 2020 Mobile dataset [127], mixed with the ImageNet 2012 dataset [114] with randomly cropped image patches taken of size 256×256. For ImageNet, only images with a size larger than 256 for height and width are used to prevent bilinear upsampling that negatively affects the model performance. We use the architecture of [95], except for the autoregressive part as a context model. Instead, we use the regular convolutional architecture of [9]. The model package for the mean-scale hyperprior is from CompressAI [10]. During training, each model is evaluated on the Kodak dataset [71]. We ran all models and methods on a single NVIDIA A100 GPU.

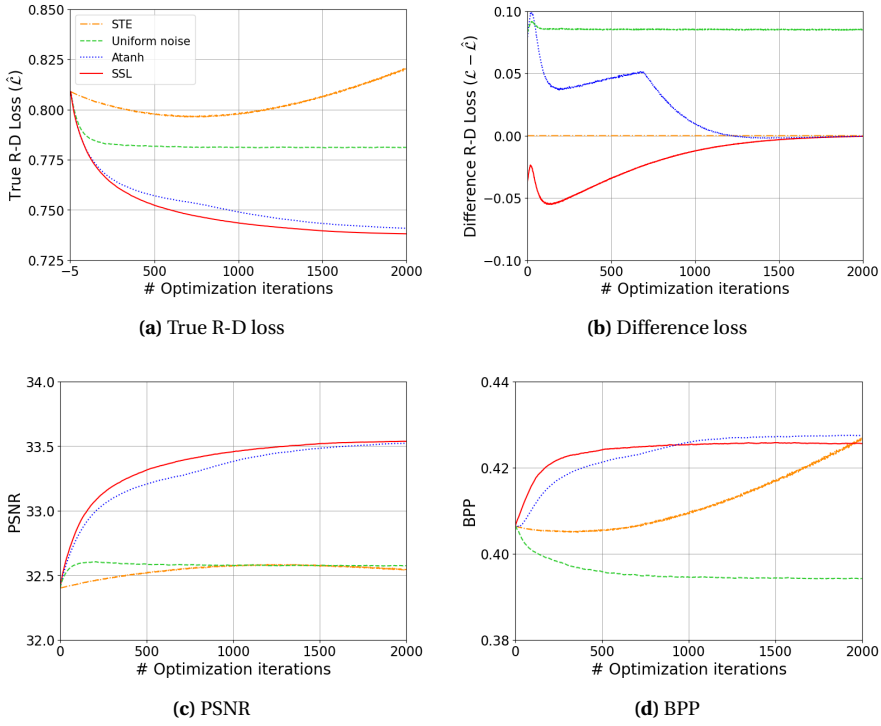
The models are trained with $\lambda = \{0.001, 0.0025, 0.005, 0.01, 0.02, 0.04, 0.08\}$, with a batch size of 32 and Adam optimizer with a learning rate set to $1e^{-4}$. The models are trained for 2M steps, except for model $\lambda = 0.001$, which is trained for 1M steps, and model $\lambda = 0.08$, which is trained for 3M steps. Further, the models for $\lambda = \{0.04, 0.08\}$ are trained with 256 hidden channels, and the model for $\lambda = 0.001$ is trained with 128 hidden channels. The remaining models are trained with hidden channels set to 192.

5.4.2 *Baseline methods*

We compare our methods against the methods that already exist in the literature. The **Straight-Through Estimator** (STE) is a method to round up or down to the nearest integer with a rounding bound set to a half. This rounding is noted as $\lfloor \cdot \rceil$. The derivative of STE for the backward pass is equal to 1 [13, 131, 146]. The **Uniform Noise** quantization method adds uniform noise from $u \sim U(-\frac{1}{2}, \frac{1}{2})$ to latent variable y. Thus: $\hat{y} = y + u$. In this manner $\hat{y}$ becomes differentiable [8]. As discussed in Section 5.2.2, we compare against SGA, which is a soft-to-hard quantization method that quantizes a continuous variable v into a discrete representation for which gradients can be computed.

5.4.3 *Experimental results*

OVERALL PERFORMANCE Figure 5.4.1 shows the overall performance for the refinement of the latents on the Kodak dataset with method: STE, uniform noise, atanh and SSL. The true R-D loss in Figure 5.4.1a shows that STE has trouble converging and uniform noise quickly converges compared to atanh and SSL. We find that SSL outperforms all other methods, including atanh, in terms of the lowest true R-D



(a) True R-D loss

(b) Difference loss

(c) PSNR

(d) BPP

Figure 5.4.1: Performance plots of (a) True R-D Loss (b) Difference in loss (c) PSNR (d) BPP.

loss. Looking at the difference between the method loss and true loss Figure 5.4.1b, we find that SSL quickly and smoothly finds a balance between the method and true loss, while atanh has a big difference in the beginning and more peaks before eventually converging. Additionally, uniform noise shows a big difference between the method loss and true loss, indicating that adding uniform noise overestimates its method loss compared to the true loss. For R-D curves, we refer to Appendix C.2.

Table 5.4.1: True R-D loss for different τ_{max} settings of: atanh(v), linear, cosine and SSL with $a = \frac{4}{3}$. The lowest R-D loss per column is marked with: ↓. Note that the function containing atanh is unnormalized.

Function $\backslash \tau_{max}$	**0.2**	**0.4**	**0.6**	**0.8**	**1.0**
$\exp \operatorname{atanh}(v)$	0.7445 ↓	0.7408	0.7412	0.7416	0.7418
$1 - v$ (linear)	0.7458	0.7406 ↓	0.7390 ↓	0.7386	0.7386
$\cos^2(\frac{v\pi}{2})$	0.7496	0.7414	0.7393	0.7387	0.7384
$\sigma(-a\sigma^{-1}(v))$	0.7578	0.7409	0.7391	0.7383 ↓	**0.7380** ↓
$\exp \operatorname{atanh}(v)$	0	0.0002	0.0022	0.0033	0.0038
$1 - v$ (linear)	0.0013	0	0	0.0003	0.0006

TEMPERATURE SENSITIVITY Now that we have assessed the overall performance, we analyze the stability of each method. Table 5.4.1 represents the stability of atanh and the SGA+ methods, expressed in true R-D loss, for different τ_{max} settings for the temperature schedule. As can be seen, the optimal setting is with $\tau_{max} = 1$ for each of the SGA+ methods. atanh obtains equal loss for $\tau_{max} \in [0.4, 0.5]$. In general, we find that the linear method of SGA+ is least sensitive to changes in τ_{max} and has equal loss between $\tau_{max} \in [0.7, 1]$. To further examine the stability of the linear function compared to atanh, we subtract the best τ_{max}, column-wise, from the linear and atanh of that column. We now see that the linear function is not only least sensitive to changes in τ_{max}, but overall varies little compared to the best τ_{max} settings of the other methods. While the SSL has the largest drop in performance when reducing τ_{max}, it achieves the highest performance overall for higher values of τ_{max}.

INTERPOLATION Table 5.4.2 represents the interpolation between different functions, expressed in true R-D loss. Values for $a < 1$ indicate methods that tend to have (extremely) large gradients for v close to the corners, while high values of a represent a method that tends to a (reversed) step function. We find that SLL with small $a = \{0.01, 0.3\}$ diverges and results in large loss values compared to the rest. Additionally, the loss curves show unstable behavior for these functions, which can be found in Appendix C.1. For $a \in \{0.8, \ldots, 2.25\}$, the loss shows stable behavior in terms of its curves and final loss at $t = 2000$. Furthermore, for $a = 1.65$ (approximately atanh), the method shows non-optimal behavior due to a slightly unstable loss curve, see Appendix C.1, this may be due to setting $\tau_{max} = 1$ instead of optimal

Table 5.4.2: True R-D loss results for the interpolation between different functions by changing a of the SSL.

a	R-D Loss
0.01	1.1500
0.3	0.7528
0.65 (approx atanh)	0.7410
0.8	0.7396
1 (linear)	0.7386
1.33	0.7380 ↓
1.6 (approx cosine)	0.7382
2.25	0.7388
5	0.7415

setting $\tau_{max} = [0.4, 0.5]$. Note that the difference in loss 0.7410 for SSL $a = 0.65$ and 0.7418 for atanh with $\tau_{max} = 1$ (see Table 5.4.1). This difference may be because SSL is an approximation to the atanh function and not exactly equal.

THREE-CLASS ROUNDING In Table 5.4.3, the true R-D loss for two versus three-class rounding can be found at iteration $t = 500$ and in brackets $t = 2000$ iterations. For each method, we performed a grid search over the hyperparameters r and n. Additionally, for the extended SSL, we also performed a grid search over a and found the best setting to be $a = 1.4$. As can be seen in the table, the extended version of the linear of SGA+ has the most impact in terms of the difference between the two versus three-class rounding at iteration $t = 500$ with loss difference 0.0035 and $t = 2000$ with 0.0006 difference. There is a small difference at $t = 500$ for the extended cosine version. In general, we find that running models longer results in convergence to similar values. SSL converges to equal values for two- and three-class rounding. This makes three-class rounding attractive under a constraint optimization budget, possibly because it is easier to jump between classes.

Table 5.4.3: True R-D loss of two versus three-class rounding for SGA+ with the extended version of the linear, cosine, and SSL method at iteration 500 and in brackets after 2000 iterations.

Function \Rounding	Two	Three
$f_{3c,\text{linear}}(w\|r = 0.98, n = 1.5)$	0.7552 (0.7386)	0.7517 (0.7380)
$f_{3c,\text{cosine}}(w\|r = 0.98, n = 2)$	0.7512 (0.7384)	0.7513 (0.7379)
$f_{3c,\text{sigmoidlogit}}(w\|r = 0.93, n = 1.5)$	0.7524 (0.7380)	0.7504 (0.7380)

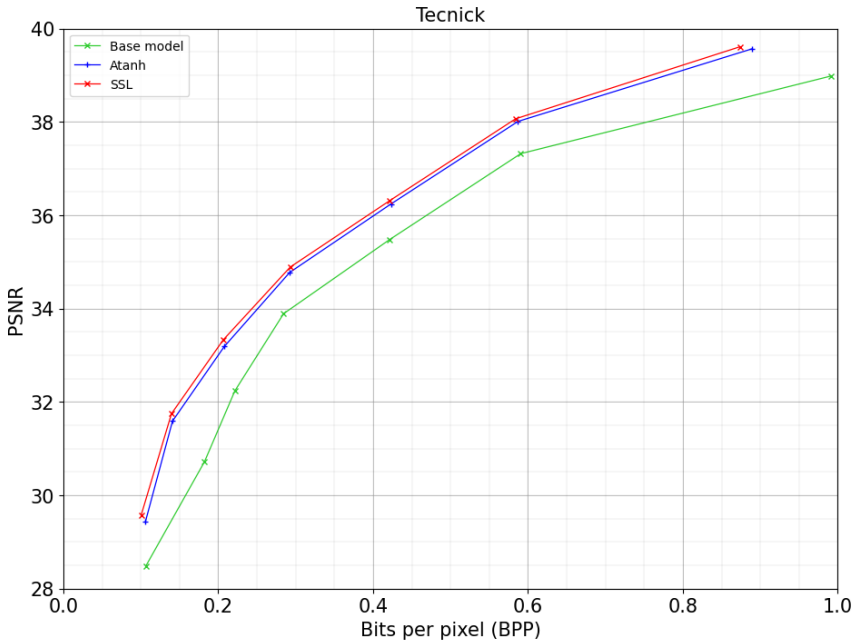


Figure 5.4.2: Rate-distortion performance on the Tecnick dataset at iteration 500 for the base model, atanh and SSL.

5.4.4 *Tecnick*

To test how our method performs on another dataset, we use the Tecnick dataset [6]. We run baselines atanh and the base model and compare against SSL with $a = \frac{4}{3}$. Figure 5.4.2 shows the corresponding R-D curves, using image quality metric PSNR versus BPP, after $t = 500$ iterations. We find that both refining methods greatly improve the compression performance in terms of the R-D trade-off. Additionally, our proposed method outperforms the baselines and shows how it boosts performance, especially for the smallest and highest $\lambda \in \{0.001, 0.08\}$. The R-D plot after $t = 2000$ iterations can be found in Appendix C.2.

5.4.5 *Semi-multi-rate behavior*

An interesting observation is that one does not need to use the same λ during refinement of the latents, as used during training. As a consequence of this approach, we can optimize to a neighborhood of the R-D curve without the need to train a new model from scratch. For instance, we only need three out of seven models to have better performance over the entire curve compared to the base model. We experimented with different values for λ. For every pre-trained model, we ran our best SSL approach using $\lambda \in \{0.0001, 0.0005, 0.001, 0.0025, 0.005, 0.01, 0.02, 0.04, 0.08, 0.16, 0.32\}$ to obtain a semi-multi-rate curve. In Figure 5.4.3, we have plotted the R-D

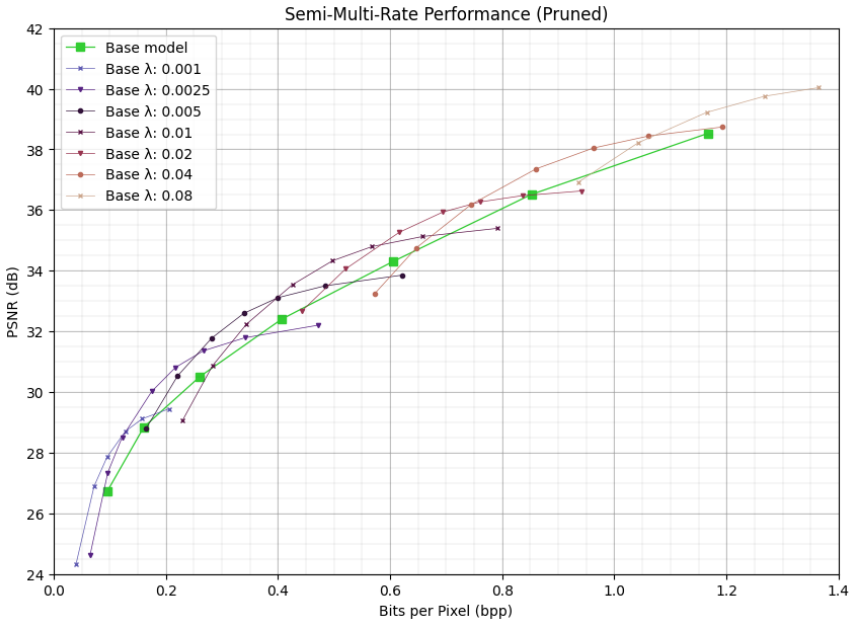


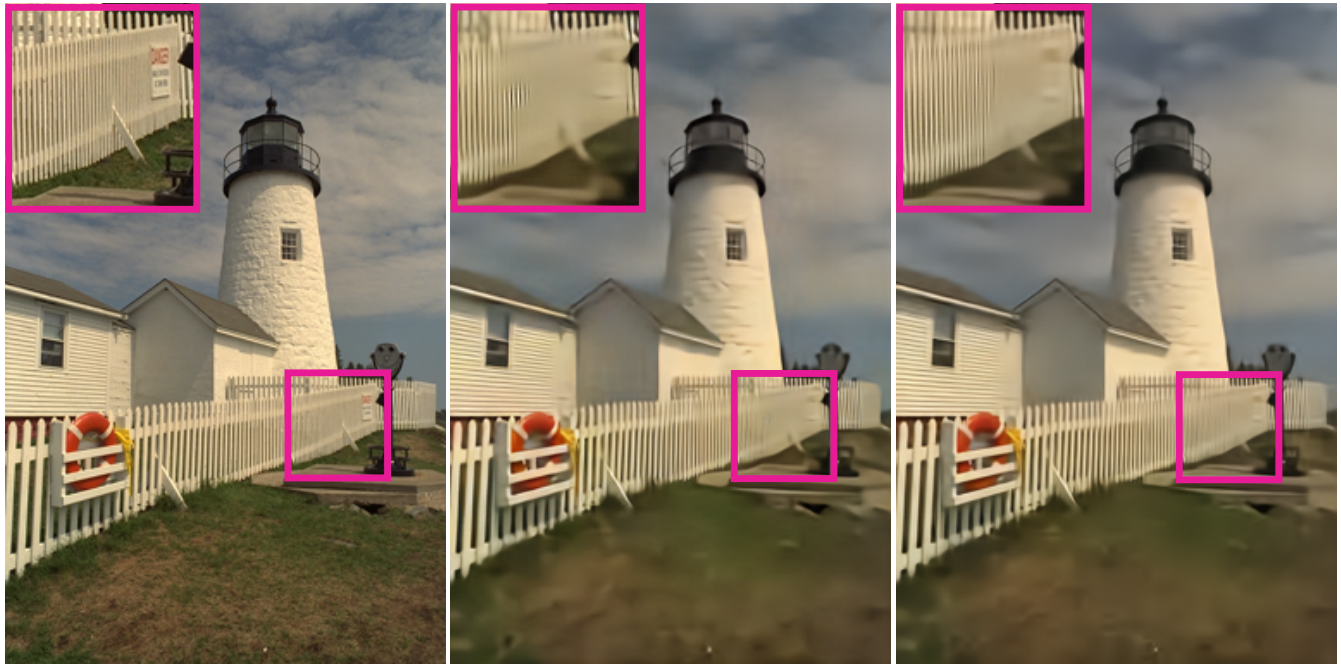
Figure 5.4.3: RD-plot with semi-multi-rate behavior.

curve of the base model (i.e. the lime green line) and its corresponding R-D curves, obtained when refining the latents with the proposed λ's. We pruned away points that fell too far below the base curve and refer to Appendix C.3 for the unpruned version.

QUALITATIVE RESULTS In Figure 5.4.4, we demonstrate the effects of using the semi-multi-rate strategy. We compare the original image, the compressed image using the base model for $\lambda = 0.001$, and the compressed image using SSL with a base $\lambda = 0.0025$ and a target $\lambda = 0.0005$. We have chosen the latter two to compare as their BPP values are similar, while the base models are different. For the image compressed by SSL, we observe that there are less artifacts visible. For instance, looking at the fence, we see more texture compared to the base model. Additionally, looking at the sky we find less artifacts compared to the base model.

5.4.6 *Hyperparameter settings*

Similar to [144], we find an optimal learning rate of 0.005 for atanh and find the best performance for STE with a lower learning rate of 0.0001, yet STE still has trouble converging. Additionally, we find the best performance for atanh with $\tau_{max} \in [0.4, 0.5]$. For SGA+, we use optimal convergence settings, which are a fixed learning rate of 0.005, a temperature rate of $c = 0.001$, and $\tau_{max} = 1$. Furthermore, experimentally, we find approximately equal and best performance for SLL with $a \in [1.3, 1.4]$. For



(a) Original Kodak Image $\lambda = 0.001$ (0.08 BPP vs 27.08 PSNR)

(b) Base Model $\lambda = 0.001$ (0.08 BPP vs 27.08 PSNR)

(c) Base model $\lambda = 0.0025$, SSL trained with $\lambda = 0.0005$ (0.08 BPP vs 27.73 PSNR)

Figure 5.4.4: Qualitative comparison of an Kodak image. Base model for $\lambda = 0.001$ compared to base model $\lambda = 0.0025$ tuned with $\lambda = 0.0005$ during the refinement procedure for SSL. Best viewed electronically.

SSL we choose $a = \frac{4}{3}$, see corresponding probability in Figure 5.3.1.

5.5 SOCIETAL IMPACT

The development and improvement of neural compression techniques play an important role in our increasingly data-driven society. Being able to learn compression codecs as opposed to developing them saves time and effort. There is a rich array of signal-based data types available that benefit from compression. A smaller data footprint reduces the requirements for storage, networking, and computing resources. It thus also means smaller investments in terms of raw resources, manufacturing, energy, and labor.

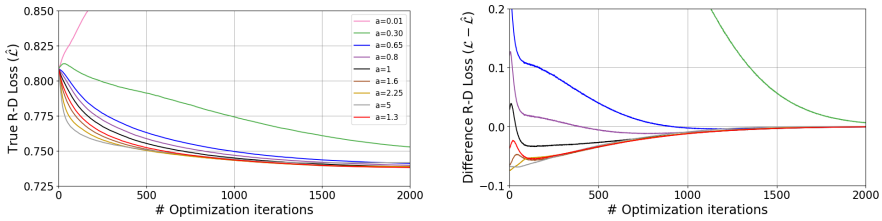
Neural image compression methods still require a large amount of computation to be trained. This process has tremendous adverse effects on the environment, as established earlier. Our work aims to make these models more flexible through the proposal of methods that improve on these already trained models. Improved flexibility can result in improved reusability, which may reduce the environmental impact of neural image codecs.

5.6 CONCLUSION

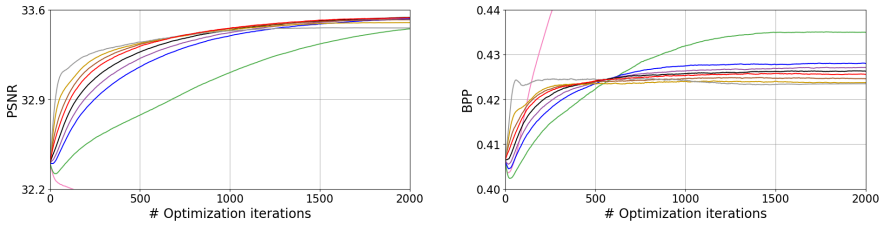
Training neural image compression models is a time-consuming and difficult task. In practice, finding optimal encoding comes with difficulties and can lead to sub-optimal performance of these networks. Refining latents of a pre-trained network has been shown to improve the compression performance by adding uniform noise or using a stochastic method called SGA [22, 144].

We extend this idea and propose SGA+. We show how SGA+ has nicer properties, which aids the compression performance. We introduced SSL that can approximately interpolate between all of the proposed methods. Further, we show how our best-performing method, SSL, outperforms the baselines and that it is more stable under varying conditions. We give a general notation and demonstrate how the extension to three-class rounding improves the convergence of the SGA+ methods but comes with the cost of fine-tuning extra hyperparameters. Finally, we show how our proposed method performs when refining latents on the Tecnick dataset and how we can use refining of the latents to obtain semi-multi-rate behavior.

In general, it applies for each method that as the temperature rate has reached a stable setting, the longer one trains, the closer together the performance will be. However, when under a constrained optimization budget, running SGA+ shorter already gives good results. The best results are obtained while tuning the hyperparameter of SSL. Further, our experiments show that the linear version for SGA+ is least sensitive to hyperparameter changes.



(a) True R-D loss **(b)** Difference loss



(c) PSNR **(d)** BPP

Figure C.1: Interpolation performance plots of different a settings for SLL (a) True R-D Loss, (b) Difference in loss (c) PSNR, and (d) BPP.

C APPENDIX FOR IMAGE COMPRESSION

In this appendix, additional experimental results can be found.

C.1 *Interpolation*

Table 5.4.2 presents the true R-D loss results for the interpolation with different a settings for SSL. In Figure C.1, the corresponding overall performance of the methods can be found. As can be seen in Figure C.1a, for $a = \{0.01, 0.30\}$, the functions diverge, resulting in large loss values. For $a = 0.65$, we find that the loss curve is slightly unstable at the beginning of training, which can be seen in the bending of the curve, indicating non-optimal settings. This may be due to the fact that we run all methods with the same $\tau_{max} = 1$ for a fair comparison. Additionally, note that SSL with $a = 0.65$ obtains a true R-D loss of 0.7410 compared to 0.7418 for atanh with the same settings. This is due to the fact that SSL, especially in the tails of the probability, is slightly more straight-curved compared to the atanh when looking at its probability space.

Remarkably, for $a \geq 1$, the difference in losses starts close to zero (see Figure C.1b). SSL with $a = 5$ results in the fastest convergence and quickly finds a stable point but ends at a higher loss than most methods.

C.2 *Rate-distortion performance*

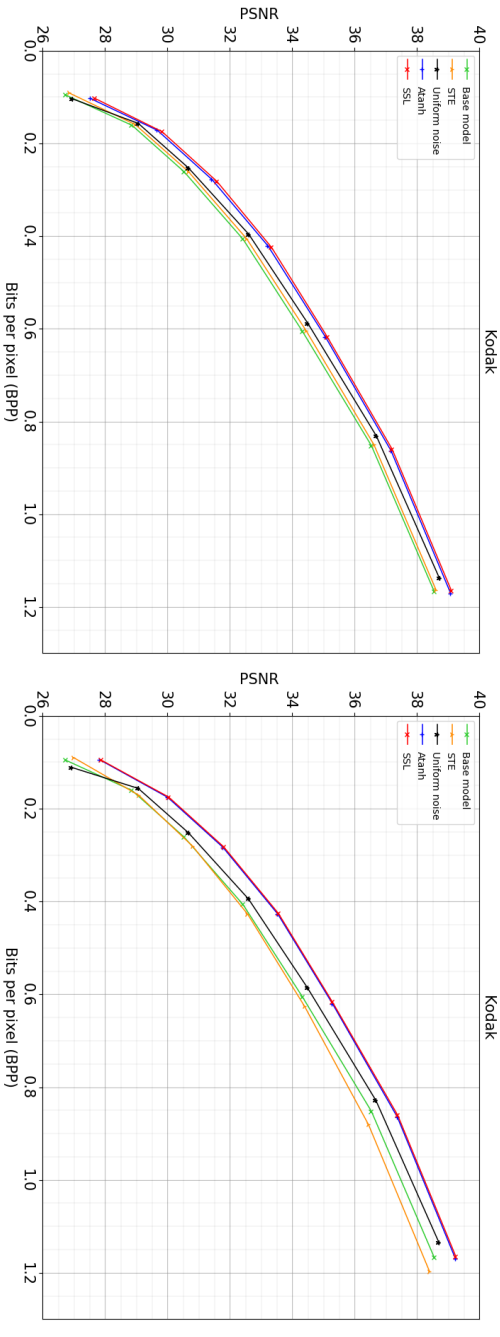
We evaluate our best performing method SSL with $a = \frac{4}{3}$ on the Kodak and Tecnick datasets, by computing the R-D performance, average over each of the datasets. The R-D curves use image quality metric PSNR versus BPP on the Kodak and Tecnick dataset. Recall that as a base model, we use the pre-trained mean-scale hyperprior, trained with $\lambda = \{0.001, 0.0025, 0.005, 0.01, 0.02, 0.04, 0.08\}$.

KODAK Figure C.2 shows the R-D curve for refining the latents, evaluated on Kodak. We compare SLL against baselines: STE, uniform noise, atanh and the base model at iteration $t = 500$ (see Figure C.2a) and after full convergence at $t = 2000$ (see Figure C.2b). As can be seen in Figure C.2a, STE performs slightly better than the base model, while after $t = 2000$ iterations the method performs worse, this also reflected in the corresponding true loss curve for $\lambda = 0.01$ (see Figure 5.4.1a), which diverges. Remarkably, for the smallest $\lambda = 0.001$, STE performs better than at $t = 500$. Adding uniform noise results in better performance when running the method longer. Further, SSL outperforms atanh slightly, while the difference is more pronounced when running for $t = 500$ iterations.

TECNICK Figure C.3 shows the R-D curve when refining latents on the Tecnick dataset, after $t = 2000$ iterations. As can be seen in the plot, we find that the longer the methods run, the closer the performances lie to each other. When there is a limited budget available, one can run the refinement process for $t = 500$ iterations.

C.3 *Semi multi-rate behavior*

Figure C.4 shows the different R-D curves when refining the latents using different values for λ. For each model trained using $\lambda \in \{0.001, 0.0025, 0.005, 0.01, 0.02, 0.04, 0.08\}$, we run SSL with $a = \frac{4}{3}$ for $t = 2000$ iterations for all $\lambda \in \{0.0001, 0.0005, 0.001, 0.0025, 0.005, 0.01, 0.02, 0.04, 0.08, 0.16, 0.32\}$. We depicted the base curve alongside the curves for each base model. We observe that using SGA+ in this manner gives samples diverse enough such that there is always a point on the resulting curve that is better than the base curve. As a result, latent fine-tuning allows for semi-multi-rate behavior. We also observed that when the values for λ used between training and SGA+ differ too much, the resulting BPP versus PSNR point becomes worse than the base model.



(a) R-D curve at iteration $t = 500$

(b) R-D curve at iteration $t = 2000$

Figure C.2: R-D performance on Kodak of the base model, STE, Uniform noise, SGA atanh and SSL with $a = \frac{4}{3}$. Best viewed electronically.

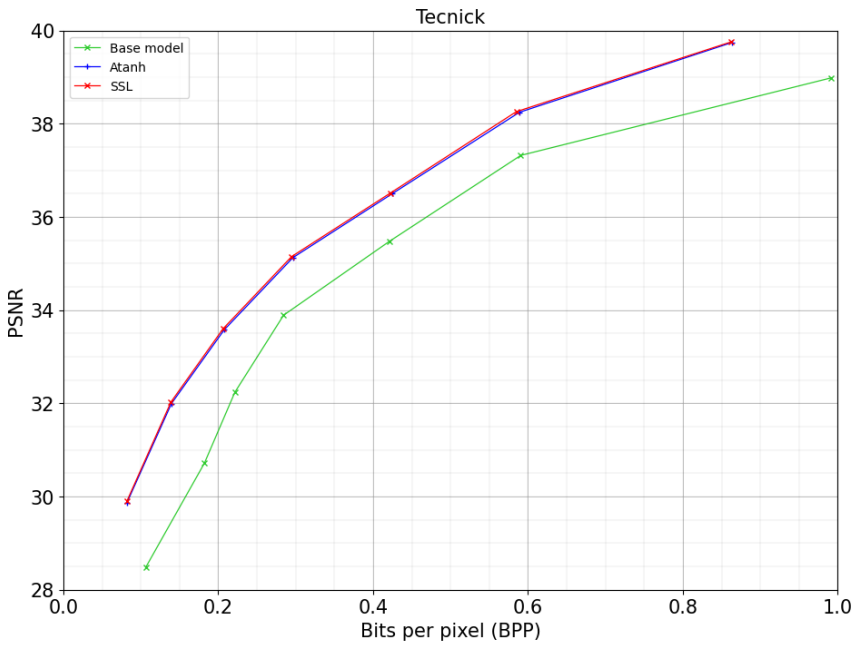


Figure C.3: R-D performance on Tecnick after $t = 2000$ iterations, which compares SSL with the baselines: base model and atanh. Best viewed electronically.

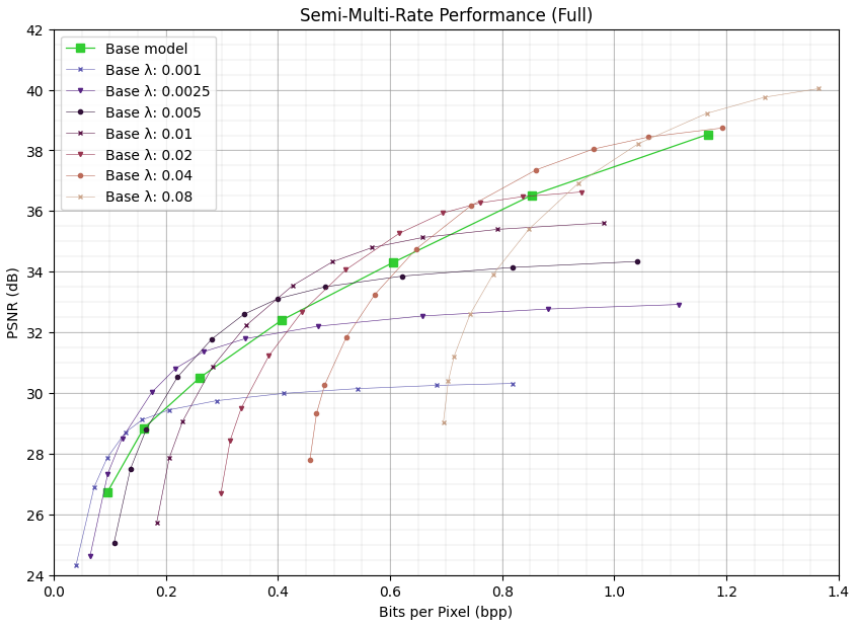
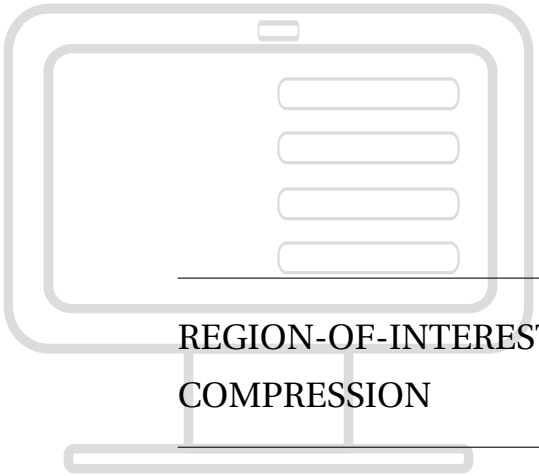


Figure C.4: R-D performance with semi-multi-rate behavior. For each base model, we ran SGA+ with $\lambda \in \{0.0001, 0.0005, \ldots, 0.16, 0.32\}$ and depicted the resulting BPP/PSNR point.



6

REGION-OF-INTEREST BASED NEURAL VIDEO COMPRESSION

Abstract

Recently, several neural codecs have been introduced for video compression, yet they operate uniformly over all spatial locations, lacking the capability of ROI-based processing. We introduce two models for ROI-based neural video coding. First, we propose an implicit model that is fed with a binary ROI mask and it is trained by de-emphasizing the distortion of the background. Secondly, we design an explicit latent scaling method, that allows control over the quantization binwidth for different spatial regions of latent variables, conditioned on the ROI mask. By extensive experiments, we show that our methods outperform all baselines in terms of Rate-Distortion performance in the ROI. Moreover, they can generalize to different datasets and ROI specifications at inference time. Finally, they do not require expensive pixel-level annotations during training, as synthetic ROI masks can be used with little to no degradation in performance. To the best of our knowledge, our proposals are the first solutions that integrate ROI-based capabilities into neural video compression models.

Based on [2]:
Yura Perugachi-Diaz, Guillaume Sautière, Davide Abati, Yang Yang, Amirhossein Habibian, Taco Cohen
Region-of-interest based neural video compression
33rd British Machine Vision Conference BMVC, London, UK, November 21-24. 2022, pages 1–21.

6.1 INTRODUCTION

Deep learning for neural image compression has proven to be successful in practice and outperforms traditional methods [9, 82, 95]. Besides being successful for image compression, deep learning for neural video compression is rapidly gaining ground [2, 51, 111]. Furthermore, in neural image compression, there are several models that not only learn how to encode an entire image but also learn how to encode Regions-Of-Interest (ROIs) [21, 85, 141] with better compression ratios than non-ROIs. ROI-based coding is useful for the user to compress only user-interesting parts with high quality and non-ROIs with lower quality. Although neural video compression models have been successfully applied to compress entire video frames, there are no solutions that integrate ROI-based compression.

The most common approach in neural lossy video compression is to rely on variational autoencoders to minimize the expected rate-distortion (R-D) objective, $D + \beta R$ [2, 51, 89, 95, 111]. Although this approach has proven to be successful, a model trained to minimize the expected rate-distortion tradeoff uniformly over all pixels may allocate too few bits to salient regions of a specific video. This clashes with the model of the human visual system, which is space-variant and has the highest spatial resolution at the foveation point [62, 135]. Exploiting this phenomenon, e.g. by encoding ROIs with higher fidelity, can significantly contribute to the subjective quality under a low bitrate regime. The key idea of traditional *ROI-based codecs* [21, 27, 54, 85, 87, 117, 141] is to allocate different bitrate budgets for objects or regions of interest, and therefore to allow for non-uniform reconstruction qualities. For instance, traditional codecs like JPEG2000 [120] and MPEG-4 [133] were used as basis to build object-based coding methods [27, 54]. However, these ideas lacked widespread adoption due to their complexity and block-based nature, which limited their capability to deal with arbitrary ROI shapes.

More recently, some works have developed ROI-based neural *image* codecs, either by implicitly identifying the ROI as part of the encoding process [21, 85], or by relying on external algorithms for its extraction [141]. Under both approaches, the R-D objective can be spatially weighted, and additionally, the latent variables can be masked before the quantization step to reduce their entropy [21, 51, 141]. Nevertheless, existing neural ROI-based codecs have the following limitations: (i) they only work for images, (ii) they use intricate masking schemes to spatially control the rate, without exploiting the Gaussian structure of the latent prior distribution and (iii) the encoding operations are tightly coupled with ROI prediction, which makes it hard for the codecs to be adapted to different ROI requirements.

In this chapter, we present the first two neural codecs capable of ROI-based compression. The first *implicit* model is fed with the ROI mask and is trained with an ROI-aware loss, where the distortion of the background is de-emphasized. Secondly, the *latent-scaling* model extends the implicit model by exploiting a recent technique originally developed for variable rate coding [25, 26, 33, 92]. We extend its design by introducing an auxiliary autoencoder (AE) being fed with the ROI map, and regressing a gain tensor explicitly controlling the quantization binwidth for different spatial

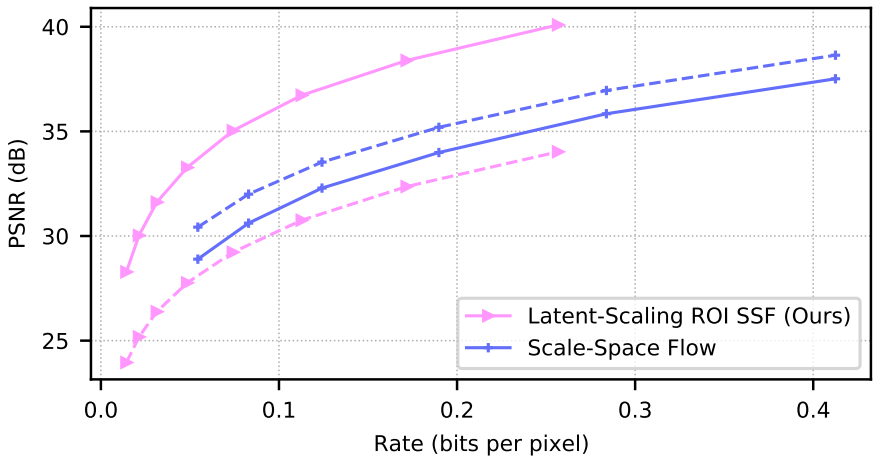


Figure 6.1.1: R-D improvements on the DAVIS dataset [105]. The solid line denotes ROI-PSNR, while dashed non-ROI PSNR. The improvement in ROI-PSNR is equivalent of 69.3% saving in bitrate measured by BD-rate gain [16].

regions. This can be seen as the continuous equivalent of the masking scheme used in conjunction with scalar quantization [21, 85, 94]. We describe our solution in the context of a Scale Space Flow (SSF) [2] architecture; however, we argue that they are in principle compatible with most state-of-the-art models based on hyperpriors [89, 106, 111]

We show that our methods outperform all our baselines on the DAVIS dataset [105] in terms of R-D performance, as measured in PSNR in the ROI (Figure 6.1.1). Moreover, further analyses show that they generalize to any arbitrary ROI that can be specified by the user at inference time and that expensive pixel-dense annotations are not required during training, as synthetic ROI can be used with little to no degradation in performance.

6.2 RELATED WORK

6.2.1 *Non-uniform coding*

The literature on spatially variant image encoding mainly focuses on two separate problems: (i) how to estimate the ROI and (ii) how to exploit it to improve coding. Most traditional block-based methods [27, 53, 54] fall under the former category and simply exploit non-uniform coding capabilities of standard codecs such as JPEG2000 [120] and MPEG-4 [133]. These solutions are limited in their capabilities due to their block-based approach to compression, which hinders the encoding of arbitrarily shaped objects and does not allow for pixel-level bit allocation optimization [87, 117].

In contrast, recent work in neural image coding tackles both the above mention problems and target pixel-level ROI [3, 4, 21, 37, 38, 85, 141]. Among these, Li et al. [85] and Cai et al. [21] learn the ROI implicitly by spatially masking out the latents before scalar quantization, whereas Xia et al. [141] use the down-scaled output of the DeepLab [23] segmentation network to mask out foreground from background, before sending each stream to a separate hyper-codec for quantization. Similar to these works, our work focuses on how to use a given ROI to enable non-uniform coding, whilst delegating its extraction to some external automatic model such as [23, 76, 78, 136–138, 149]. However, our approach extends neural ROI-coding to the case of video inputs.

6.2.2 *Neural video compression*

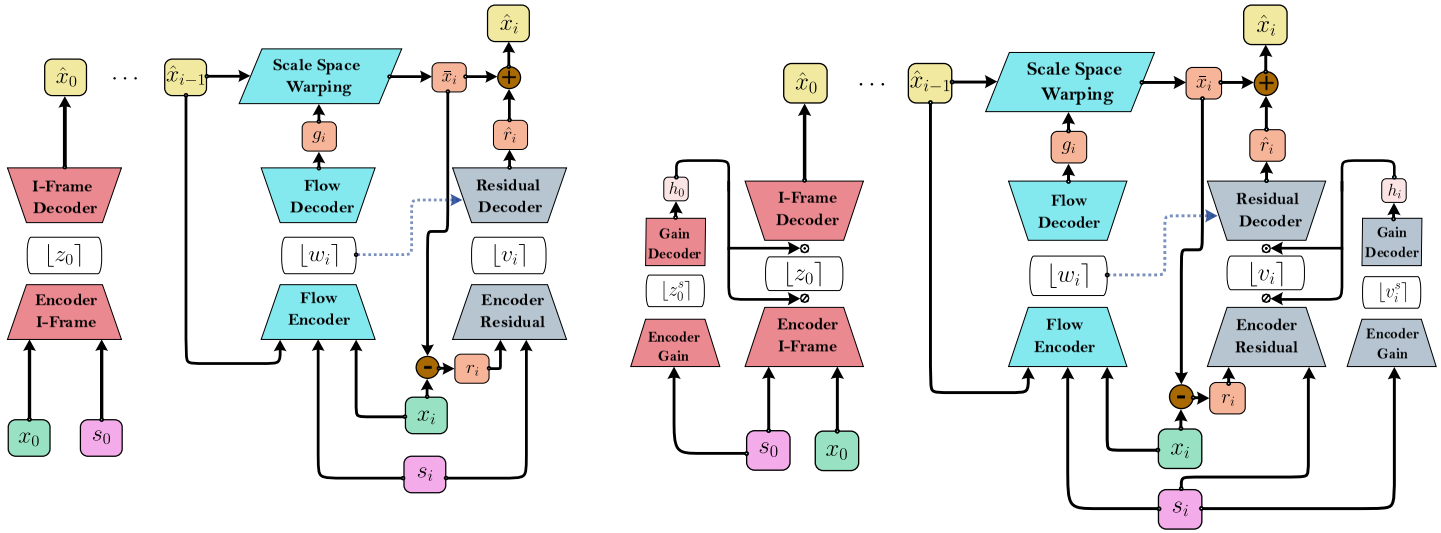
Compressing videos with neural networks has been an active field of research recently [2, 44, 51, 59, 89, 91, 106, 111, 112, 140]. While varying in their choice of architecture and quantization strategy, neural video codecs generally follow the DVC [91] framework where an I-frame codec compresses the first frame and a P-frame codec uses motion estimation and a residual network to model the subsequent ones. Recently, Agustsson et al. [2] proposed to use a Scale-Space Flow which addresses uncertainties in motion estimation via interpolation through a Gaussian pyramid. This allows blurring of the warped frames in regions where optical flow prediction is uncertain or ill-posed, like chaotic motions and obstructed objects. Our work is established in the same SSF framework, and enables ROI-based coding by means of latent scaling [25, 26, 33, 92], a technique originally introduced for variable bitrate coding. Differently from these works, that scale the latents globally with a single scalar value, we adjust the quantization step size for every spatial location, thus controlling the levels of distortion and entropy in foreground and background regions.

In summary, we are the first work to learn ROI coding end-to-end for video inputs (as opposed to images) and extend latent scaling spatially to be used in an ROI-based context. Additionally, other works either learn implicitly the ROI using a subnetwork [21, 38, 85] or tie themselves to a restricted set of semantic classes [3, 4, 37], which would require re-training if testing on unseen classes. In contrast, we explicitly take the ROI as input, which provides the user evaluation time flexibility similar to H.264 and H.265 ROI mode.

6.3 ROI-BASED NEURAL VIDEO COMPRESSION

In this section, we first present the neural video codec we use as the backbone for our work, Scale Space Flow (SSF) [2]. Next, we extend SSF to be an ROI-based codec by proposing two models: the *Implicit* and *Latent-scaling* ROI SSF. Lastly, we will describe the optimization for SSF and the ROI-aware methods.

We define a video frame $x_i \in \mathbb{R}^{H\times W\times 3}$ at time step i, where H and W represent its height and width respectively. Then, a video sequence is denoted as $\mathbf{x} = \{x_0, x_1, \ldots,$



(a) Implicit ROI Scale-Space Flow.

(b) Latent-scaling ROI Scale-Space Flow.

Figure 6.3.1: Illustration of the proposed ROI-based neural video compression models.

$x_T\}$, with $T+1$ frames. The sequence of binary ROI masks corresponding to the video sequence is defined as $\mathbf{s} = \{s_0, s_1, \ldots, s_T\}$, where $s_i \in \{0,1\}^{H \times W}$. The neural video codec SSF consists of an I-frame codec and a P-frame codec. The I-frame codec is a mean-scale hyperprior AE [95] which encodes a first frame x_0 independently to produce a reconstruction $\hat{x}_0$. The P-frame codec is comprised of two hyperprior AEs. The first, the *P-frame flow hyperprior AE*, estimates a scale-space flow g_i from the previous reconstruction $\hat{x}_{i-1}$ and current frame x_i, which is used to warp the previous reconstruction into $\bar{x}_i$. The second hyperprior AE, *the P-frame residual hyperprior AE*, encodes the residual $r_i = \bar{x}_i - x_i$. The final reconstruction $\hat{x}_i$ is obtained by adding the warped prediction $\bar{x}_i$ and the estimated residual $\hat{r}_i$. The latent codes of each hyperprior AE are denoted by z_0, w_i and v_i and are rounded to integer values then entropy coded using the prior parameters estimated by their respective hyper-decoder. We omit hyper latent codes for ease of exposition, and we refer to [2] for further details.

6.3.1 *Implicit ROI Scale-Space Flow*

An immediate extension to SSF to make it ROI-aware is to provide the ROI mask s_i as input to each of the three hyperpriors, see Figure 6.3.1a. Note that the ROI mask is not fed to the decoder, meaning we expect the encoders to implicitly store the relevant ROI information inside the existing latent codes. Since the decoder does not require the ROI mask, we do not need to transmit a representation of the mask itself. Feeding information about the mask along with the video frame, in combination with the use of an ROI-aware loss, encourages the model to focus on important aspects for the user. Albeit simple, we show the effectiveness of this approach when paired with an ROI-aware loss in Section 6.4.

6.3.2 *Latent-scaling ROI Scale-Space Flow*

Inspired by methods like [21, 85] which introduce a mechanism to explicitly control the spatial bit allocation, we adapted a recent technique called latent-scaling [26, 33]. Albeit similar in its motivation, it differs from the masking approach of [33] by exploiting the Gaussian prior structure of mean-scale hyperprior AE. The key idea is to apply a scaling factor to the latent which changes the quantization step size, leading to different trade-offs between rate and distortion in ROI and non-ROI areas. By using ROI-based information to control the scale of latents, the quantization grid can be explicitly adjusted. Our model can, therefore, learn that foreground regions require finer quantization than background regions. For ease of exposition, we will describe in the next paragraphs latent-scaling for the I-frame hyperprior AE, but the same method is applied to the P-frame residual hyperprior AE. We do not apply it to the P-frame flow hyperprior AE as initial studies showed the flow code w_i only accounts for a small fraction of the total rate. For similar reasons, we only apply latent-scaling latents, leaving hyper-latents, which are cheap to encode, unaffected.

We introduce a new hyperprior-like network called *gain hyperprior AE* (see leftmost autoencoder in Figure 6.3.1b). This network encodes the ROI mask s_0 into a latent

code z_0^s, which is decoded to a gain variable h_0, which shares the same dimensions as the latent variable z_0, both spatially and channel-wise (previous latent-scaling [26, 33, 92] work only use channel-wise gain). We scale the latent z_0 with the inverse of the estimated spatial gain variable h_0, where we restrict $h_0 \geq 1$. Such a procedure is akin to making the quantization range larger, depending on the value of h_0. We further denote the mean μ and scale σ as the prior parameters estimated by the I-frame hyper-decoder. In the quantization step, we choose to center the scaled latent $z_0 \oslash h_0$ by its prior mean $\mu \oslash h_0$, where $\oslash$ is an elementwise division. Next, we apply the rounding operator $\lfloor \cdot \rceil$ on $(z_0 - \mu) \oslash h_0$ such that the estimated mean μ learned by the hyper-encoder is on the grid, and then add the offset $\mu \oslash h_0$ back. The dequantized latent $\hat{z}_0(h_0)$ is obtained by multiplying by h_0 after the quantization block. More precisely:

$$\hat{z}_0(h_0) = \lfloor (z_0 - \mu) \oslash h_0 \rceil \odot h_0 + \mu, \tag{6.3.1}$$

where $\odot$ denotes elementwise multiplication. After the dequantized latent $\hat{z}_0(h_0)$ is obtained, it is passed to the decoder to obtain the reconstructed frame $\hat{x}_0$. The whole procedure is illustrated in Figure 6.3.2a. For rate computation and entropy coding, we use the modified probability $\mathbb{P}$ of $\hat{z}_0(h_0)$ as follows:

$$\mathbb{P}(\hat{z}_0(h_0)) = \int_{\hat{z}_0(h_0) - h_0/2}^{\hat{z}_0(h_0) + h_0/2} \mathcal{N}(x - \mu | 0, \sigma)\, dx \tag{6.3.2}$$

$$= \int_{\hat{z}_0(h_0)/h_0 - 1/2}^{\hat{z}_0(h_0)/h_0 + 1/2} \mathcal{N}\left(x - \frac{\mu}{h_0} \middle| 0, \frac{\sigma}{h_0}\right) dx. \tag{6.3.3}$$

As shown in Figure 6.3.2b and in Equation (6.3.2), latent-scaling can be interpreted as effectively changing the quantization grid / binwidth. In practice, for entropy coding we do not change the quantization grid and round to the integer grid and scale the prior appropriately, as in Figure 6.3.2a and b (middle plot) and Equation (6.3.3). As stated above, the same procedure is applied to the P-frame residual latent code v_t, as shown in Figure 6.3.1b.

6.3.3 *ROI-aware Rate-Distortion Loss*

We modify the regular R-D loss from SSF to take into account the ROI mask. We sum the rate and distortion for all T frames in the video sequence $\mathbf{x}$ with corresponding ROI masks $\mathbf{s}$:

$$\mathcal{L} = \beta \mathcal{L}_R + \sum_{i=0}^{T} \mathcal{L}_{D,i}, \tag{6.3.4}$$

where β is the rate-distortion trade-off variable. $\mathcal{L}_D$ represents the distortion loss which is a modified mean squared error (MSE) involving the binary ROI mask:

$$\mathcal{L}_{D,i} = \frac{1}{HWC} \sum_{j=1}^{H} \sum_{k=1}^{W} \sum_{l=1}^{C} \left(s_i \odot \epsilon_i + \frac{1}{\gamma} \cdot (1 - s_i) \odot \epsilon_i \right)_{jkl}, \tag{6.3.5}$$

where H, W and C denote the image dimensions, γ is a penalty hyperparameter for the non-ROI, $\epsilon_i = (x_i - \hat{x}_i)^2$ is the squared error and s_i is broadcasted over the

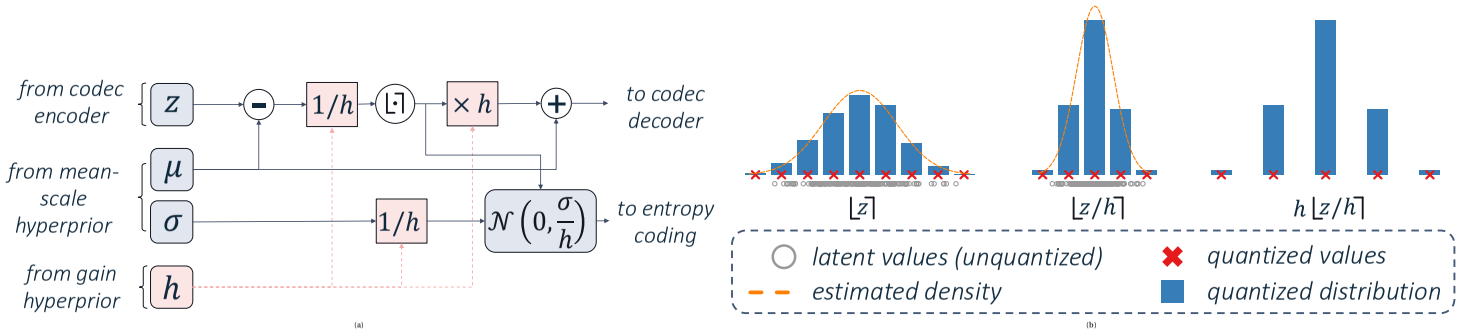


Figure 6.3.2: Illustration of the latent scaling mechanism for ROI-based coding. (a) shows how latent-scaling variable h affects latent z and prior parameters μ and σ. (b) shows intuition on why scaling the prior is necessary for entropy coding.

channel dimension. Note that the distortion loss of the original SSF corresponds to the special case where s_i equals one everywhere. Further, the rate loss $\mathcal{L}_R$ is computed with the estimated cross-entropy $\mathcal{H}(\cdot)$ by the hyperprior of each latent variable present in the model. For the implicit ROI SSF the rate loss $\mathcal{L}_{I,R}$ is equal to:

$$\mathcal{L}_{I,R} = \mathcal{H}(z_0) + \sum_{i=1}^{T} \left[\mathcal{H}(v_i) + \mathcal{H}(w_i)\right]. \tag{6.3.6}$$

The rate loss $\mathcal{L}_{LS,R}$ of the latent-scaling ROI SSF also includes latent variables z_0^s for the latent scaling of the I-frame hyperprior AE and v_i^s for the latent scaling of the P-frame residual hyperprior AE. As such, it is given by:

$$\begin{aligned}\mathcal{L}_{LS,R} =& \mathcal{H}(z_0^s) + \mathcal{H}(z_0) \\ &+ \sum_{i=1}^{T} \left[\mathcal{H}(v_i^s) + \mathcal{H}(v_i) + \mathcal{H}(w_i)\right].\end{aligned} \tag{6.3.7}$$

In practice, we found that the two extra rate contributions from the ROI masks $\mathcal{H}(z_0^s)$ and $\mathcal{H}(v_i^s)$ are only a small fraction compared to the standard rate components $\mathcal{H}(z_0)$ and $\mathcal{H}(v_i)$ of the model. Please note that in both Equations 6.3.6 and 6.3.7 we omit the rate of the hyper latent codes to avoid notational clutter.

6.4 EXPERIMENTS

6.4.1 *Datasets*

As standard video compression benchmarks [17, 130, 142] do not come with ROI annotations, we hereby introduce a benchmark for ROI-based codecs, by utilizing publicly available video segmentation datasets and deriving ROI maps from their pixel-level ground truth labels. More specifically, we rely on DAVIS [105] and Cityscapes [31] for training and evaluation of our models. DAVIS is composed of 90 diverse and short video sequences, for which ground truth segmentation of salient objects is provided. To create binary ROI masks, we consider all labeled objects as foreground, whereas the rest of the frame is labeled as background. We use 60 sequences for training and 30 for validation, comprising 4,209 and 1,999 frames respectively. Cityscapes is composed of 2,120 video sequences from dashcam of vehicles driving around German cities. 1,885 sequences are used for training and 235 for validation, or 89,248 and 15,000 frames respectively. As ground truth segmentation labels are provided only at 1 fps, we extract semantic labels automatically for every frame by running the state-of-the-art segmentation model in [124]. The dataset provides a categorization of every pixel into one of 19 classes. We select pixels of "*vehicle*", "*road*", "*pedestrian*", "*bicycle*", "*motorcycle*" as belonging to the ROI, and mark other classes as non-ROI. To reduce compression artifacts, we resize the frames from both datasets to 720p using Pillow [29].

As an alternative to ground-truth ROI masks, in some experiments (see Section 6.4.6) we rely on synthetic ROI masks generated using Perlin noise [104] (only during training). The masks contain blobs that evolve continuously over time to cover each of the video frames.

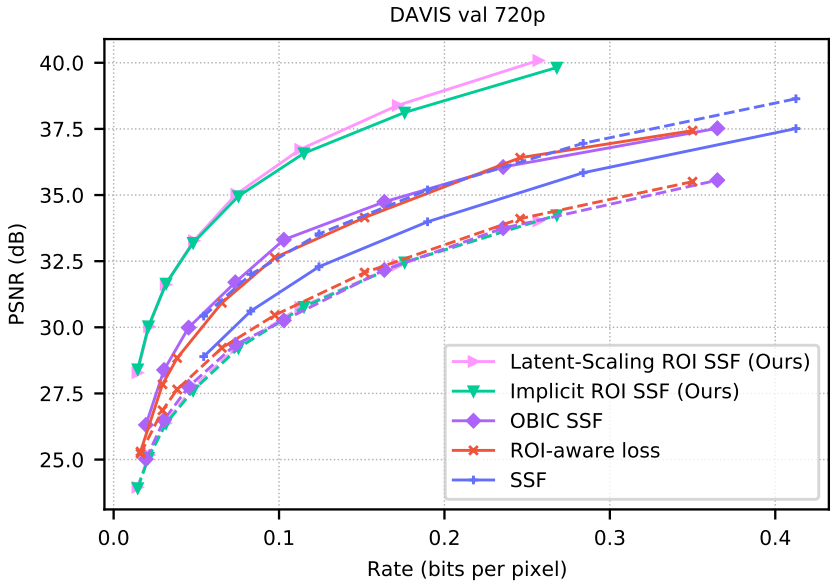


Figure 6.4.1: All ROI-based neural video compression approaches vs SSF. The solid line denotes ROI PSNR, while dashed non-ROI PSNR.

6.4.2 *Implementation details*

We optimize all methods but SSF with the ROI-aware MSE as distortion metric (Equation (6.3.5)), and use $\gamma = 30$ as penalty for the non-ROI areas. Following the training scheme from [2, 106], all models are warm-started from an SSF pre-trained on the Vimeo-90k dataset [143] for 1M steps, then fine-tuned on the dataset of interest for 300K steps. We trained all models at various rate-distortion tradeoffs with $\beta = 2^{\alpha} \times 10^{-4} : \alpha \in 0, 1, ..., 7$. We use Adam optimizer with a learning rate of 10^{-4} with batch size 8. Each example in the batch is comprised of 3 frames (I-P-P), randomly cropped to 256×384. The models take about three days to train on a single NVIDIA V100 GPU. We report video quality in terms of PSNR in ROI and non-ROI, where both are first calculated per frame in the RGB color space, then averaged over all the frames of each video and finally averaged over all the videos of a dataset. The results we report are based on a Group-of-Picture of size of 12 for consistency with other neural compression works [2, 89, 91, 106]. See Appendix D.4 for architecture details, along with information about the computational complexity of the models.

6.4.3 *Compared methods*

We compare our method to the plain SSF and two further ROI-based baselines. The first, dubbed *ROI-aware loss*, consists of SSF trained with our ROI-aware loss as described in Equation (6.3.4). While the codec is blind to the ROI, it is expected to

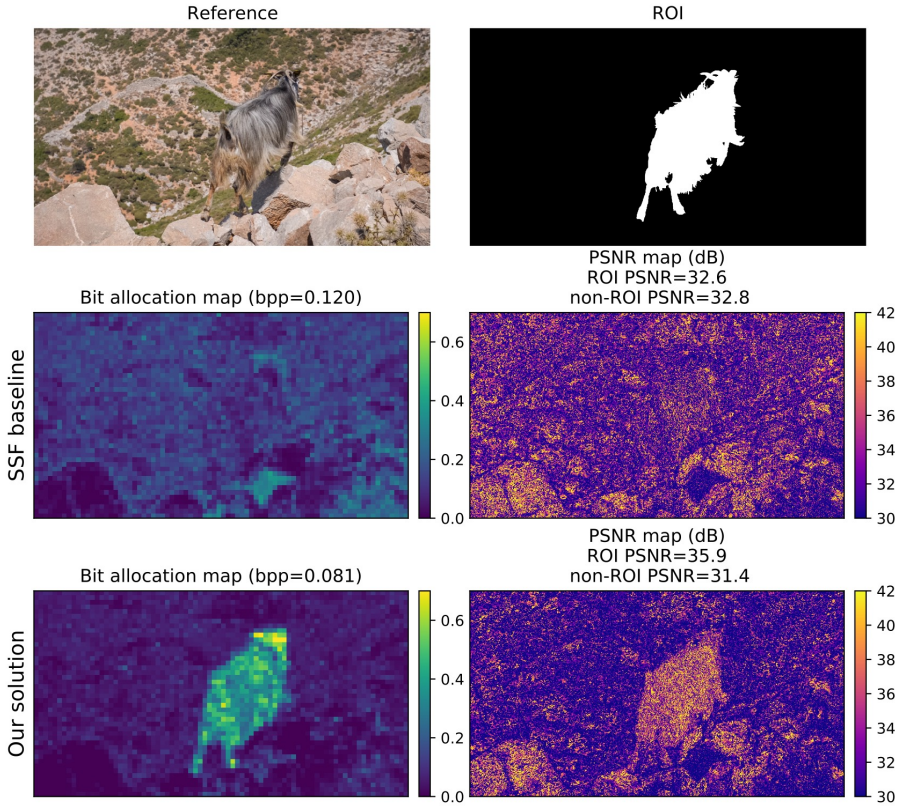


Figure 6.4.2: Bitrate and PSNR allocation maps for SSF and our proposed ROI-based codec, latent-scaling ROI SSF. We hereby report frame 5 of DAVIS "`goat`" sequence.

Figure 6.4.3: Qualitative results of SSF and our proposed ROI-based codecs, implicit ROI SSF and latent-scaling ROI SSF, on the sequences "goat, camel, parkour" sequence of DAVIS Val 2017. We hereby report frames 5, 11, 31 respectively.

implicitly learn it through the training objective, in a similar fashion as the semantic models in Habibian et al. [51]. The second method, dubbed *OBIC SSF*, is based on a recent ROI-based neural image codec [141]. To enable a fair comparison, we train this architecture using our ROI-aware loss, which is slightly different from the formulation in [141].

6.4.4 *ROI-based coding*

In Figure 6.4.1, we report the RD-plots of Implicit ROI SSF and Latent-scaling ROI SSF. We compare our proposed models to the described ROI-aware loss and OBIC SSF baselines, as well as to a plain SSF model that does not involve any ROI-based compression. For all compared models, solid lines and dashed lines correspond to RD curves in ROI and non-ROI regions respectively. The figure shows several insights. First, the plain SSF shows better compression results on non-ROI regions, that are seemingly easier to compress than ROI areas on DAVIS. This result - that we hypothesize is due to the high degree of motion affecting foreground objects on the dataset - underlines that such a codec might be suboptimal. The ROI-based baselines we consider, namely ROI-aware loss and OBIC SSF, succeed in delivering a better tradeoff for foreground regions. Overall, their performances seem comparable across the rate spectrum. Interestingly, the separate hyperprior models envisioned by OBIC SSF for foreground and background barely outperform a simple ROI-aware loss in our experiments. Finally, the figure clearly shows the superiority of the proposed implicit and latent-scaling ROI SSF. Indeed, their RD-curves perform on par with the mentioned baselines on background regions while achieving a superior tradeoff for ROI regions. In this respect, our latent-scaling based model seems to slightly outperform the implicit model in ROI areas, especially at higher bpps (> 0.1).

Furthermore, we investigate the behavior of the proposed Latent Scaling ROI SSF codec in terms of spatial bit allocation and reconstruction quality. Figure 6.4.2 shows, on a reference validation frame from DAVIS, the pixel-wise bpp and PSNR as compared to the ones achieved by SSF. For SSF, bit allocation and reconstruction quality are roughly uniformly distributed over the image. Differently, Latent Scaling ROI SSF model focuses both bpp and PSNR on the region of interest. Moreover, it is worth noting how, despite the fact latent scaling operates at the reduced resolution of the latents (resulting in block-wise bpp allocation), the PSNR of the reconstructed frame properly aligns with the ROI at pixel level. Finally, in Figure 6.4.3, we show a few qualitative compression results of our model, compared to SSF. More qualitative results can be found in Appendix D.3.

6.4.5 *Generalization*

We investigate the generalization capability of our proposed latent-scaling ROI SSF model to different data and regions of interest. To do so, train a model on DAVIS and measure its performance on Cityscapes. We expect (at least) two main sources of the generalization gap. First, the videos in the two datasets depict very different content (*data gap*), and differences in the acquisition settings may generate discrepancies in

low-level image statistics and global motion. For instance, in Cityscapes the motion is dominated by the ego-motion of the camera, which is car-mounted. Moreover, the ROI specification described above might impact training (*ROI gap*). To monitor both effects, we plot in Figure 6.4.4 the RD curves of our latent scaling model and plain SSF, trained either on DAVIS or on Cityscapes, and evaluated on Cityscapes. By considering the gap between the SSF model (blue lines) trained on DAVIS and the one trained on Cityscapes, we notice how the former performs slightly worse than the latter, both for ROI and non-ROI areas. This gives a sense of the severity of the data gap alone, as no ROI was employed during training whatsoever. In order to assess the effect of the ROI gap, we examine the margin between the two trainings of Latent Scaling ROI SSF (pink lines). Interestingly, we observe a similar edge as the one observed for plain SSF. The fact that the performance gap does not increase significantly suggests that most of the discrepancy is still due to the data gap, and that our codec is barely susceptible to the nature of ROIs used during training. Finally, we observe that, when evaluated on Cityscapes ROI areas, the ROI-based model trained on DAVIS outperforms the SSF model. This observation suggests that, when interested in ROI-based compression on a target dataset, our codec trained on a different dataset might still be a better choice than its non ROI-based counterpart, even when the latter is trained on the target dataset itself.

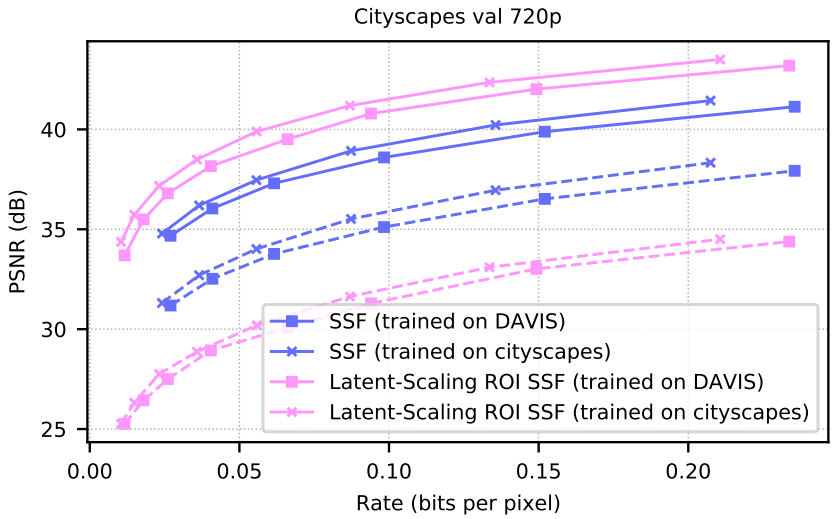


Figure 6.4.4: Latent-scaling ROI SSF tested on Cityscapes. The solid line denotes ROI PSNR, while the dashed non-ROI PSNR.

6.4.6 *Synthetic ROI masks*

In order to further investigate the sensitivity of our latent scaling based codec to the nature of ROIs used during training, we carry out an experiment where we train it

using synthetically generated masks. Specifically, we rely on the DAVIS dataset, and we generate the ROI for every training clip randomly by taking advantage of Perlin noise [104]. The resulting masks are temporally smooth, but do not correlate with the content of the video itself. In Figure 6.4.5, we plot the performance of such a model (in purple) against a model trained on regular semantic masks obtained by manual annotation (in pink). We emphasize that both models are tested on the validation set on regular semantic masks of ROI objects. Thus, we expect the model trained on realistic ROI masks to trace an upper bound RD-curve for the model trained on synthetic. Interestingly, results show that a gap exists between the two models, but it is almost negligible, confirming the intuition that our model is minimally affected by the nature of training ROIs. The close performance represented in RD-curves suggests that, although in the case ROI masks are available at training time their use is worthwhile, their lack does not represent a serious impediment for optimizing the model, as the use of synthetic masks yields similar performance on realistic use cases. In Appendix D.2 we repeat the experiment for the Implicit ROI SSF and for a different value of γ, reaching similar conclusions.

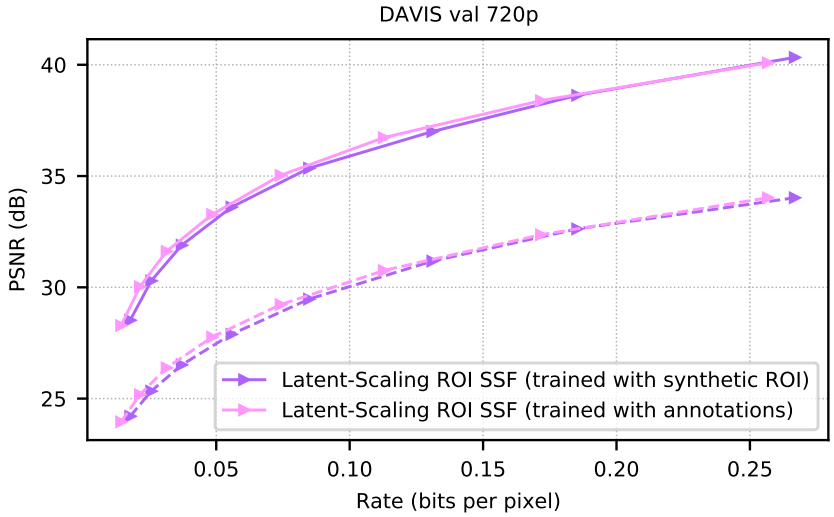


Figure 6.4.5: Effect of training with synthetic ROI instead of ground-truth annotation for the binary ROI mask. The solid line denotes ROI PSNR, while dashed non-ROI PSNR.

6.5 SOCIETAL IMPACT

Our main motivation for the Latent Scaling ROI Scale-Space Flow was to allow for inference-time single model multirate behavior for the largest rate model, without the need to re-train or to adapt the training scheme like in [33, 111] (similar to what was demonstrated in [92] for image compression). This would make our ROI codec more practical to deploy by drastically reducing the number of parameters and

allowing fine-grain control of the rate. However, it does not allow for a fully multi-rate model (*i.e.* a single model covering the whole rate spectrum), and it comes with an increase in implementation complexity with minor performance benefits over the simpler implicit ROI approach.

In addition, visual assessments highlighted how, in their current implementation, both ROI-based models can sometimes produce sharp quality transitions between ROI and non-ROI regions. The problem would probably be exacerbated if the ROI masks suffered both in terms of quality and in temporal consistency. Both of these issues may be overcome by using smooth masks during training and/or inference.

Finally, a user study would benefit the evaluation of the quality of the compressed videos as quantitative quality metrics were shown to correlate poorly with human judgment [88]. Such an analysis, based on subjective metrics such as Mean Opinion Scores (MOS), would further confirm that higher fidelity in the ROI at the cost of fidelity in the non-ROI can lead to a net boost in perceptual quality.

Concerning societal impact, we do not see immediate harmful applications of our method that might negatively affect any public. Note that because the ROI codecs depend on an ROI retrieval algorithm, the methods may suffer from (and potentially amplify) its biases and shortcomings.

6.6 CONCLUSION

We introduced two methods for ROI-based neural video compression, capable of allocating more bits to pre-specified regions of interest in order to increase their fidelity. More specifically, we introduced an implicit model being fed with the ROI, as well as a latent scaling model explicitly controlling the quantization bitwidth of the latent variables in a spatial variant fashion. Both models are optimized by an ROI-aware rate-distortion objective. We showed that our methods outperform all baselines in terms of Rate-Distortion performance in the regions of interest, and that they can generalize to different datasets at inference time. Finally, they do not require expensive pixel-level annotations during training, as synthetic ROI masks can be used with little to no degradation in performance.

D APPENDIX FOR VIDEO COMPRESSION

D.1 *ROI creation*

In Section 6.4 we explained how we created binary ROI mask from ground-truth annotations. In Figure D.1 we show visual examples of this process for the DAVIS (top) and Cityscapes (bottom) datasets.

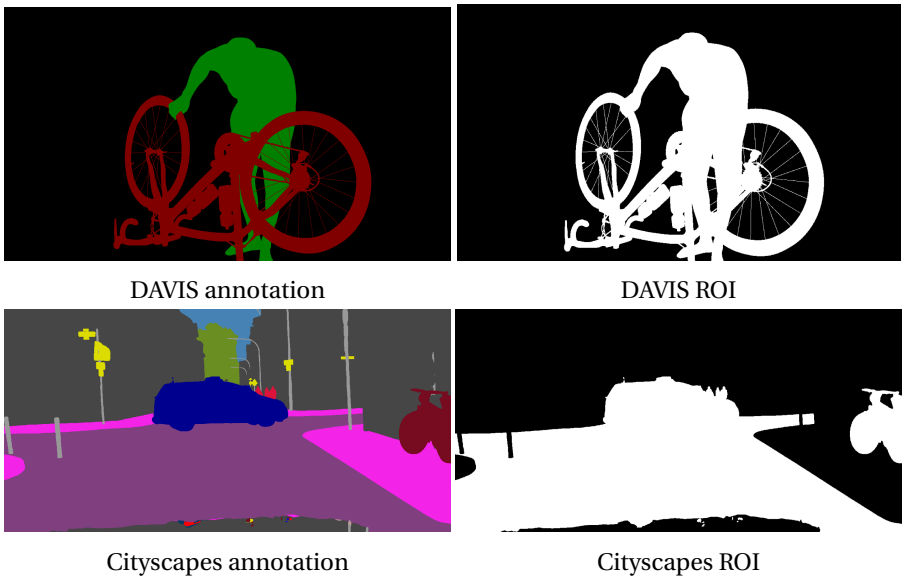


Figure D.1: Example of ROI creation for the DAVIS and Cityscapes datasets.

D.2 *Additional results*

Quantitative results

During our research we tested two different penalty term γ for non-ROI distortion, as defined in Equation (6.3.5), namely $\gamma = \{10, 30\}$. In Section 6.4, all results are shared with $\gamma = 30$ for ease of exposition. In this section, we provide additional results with $\gamma = 10$. We allow side-by-side comparison for all experiments of Section 6.4 for each penalty γ. Finally, we provide an additional multirate analysis.

ROI-BASED CODING In Figure D.2 we show all ROI-based models trained with $\gamma = \{10, 30\}$ on DAVIS and evaluated on DAVIS val, with SSF as reference. As expected from our loss formulation, a smaller penalty γ results in a smaller performance gap between ROI and non-ROI across all ROI-based methods. Interestingly, both the ROI-aware loss and OBIC SSF baselines, which are blind to the ROI mask, seem to only allow higher PSNR in the ROI than in the non-ROI at low bitrate, namely ≤ 0.15 bpp. For $\gamma = 30$, the ROI PSNR is consistently better than non-ROI PSNR across the

entire rate spectrum. The two methods may perform similarly as they are both blind to the ROI mask, *i.e.* the encoding operation does not get the ROI mask as input, although OBIC SSF foreground and background hypercodecs do get ROI information as their input is the ROI masked latent. We hypothesize that it may be insufficient for the hyper-codec network to implicitly learn to scale the prior parameters, and does not allow the encoder to scale the latent.

GENERALIZATION In Figure D.3 we show the SSF and latent-scaling ROI SSF models trained on either DAVIS or Cityscapes and evaluated on Cityscapes val for both values of $\gamma = \{10, 30\}$. As expected from our loss formulation, for $\gamma = 10$ latent-scaling ROI SSF exhibits a smaller gap between ROI PSNR and non-ROI PSNR than with $\gamma = 30$. Yet, irrespective of γ, the same observation can be made: the ROI PSNR of latent-scaling ROI SSF trained on DAVIS is higher than SSF trained on Cityscapes. This indicates that when interested in ROI-based compression on a target dataset, our codec trained on a different dataset might still be a better choice than its non ROI-based counterpart, even when the latter is trained on the target dataset itself.

SYNTHETIC ROI MASKS In Figure D.4 we show the effect of using synthetic ROI mask during training instead of ground-truth annotations, for $\gamma = \{10, 30\}$. In addition to the experiment in the main text, we not only show latent-scaling ROI SSF but also implicit ROI SSF. We find that for each of our proposed models, training with synthetically generated masks results only in a minor performance drop, albeit slightly larger for the implicit model. Since the performance of our proposed ROI-based models seems to be minimally affected by the type of ROI masks used during training, one could train them without requiring expensive pixel-wise annotations. This allows training on a target dataset of interest, which may be different from a dataset with available annotations like DAVIS. Consider, for instance, cartoons instead of natural videos.

INFERENCE TIME ROI SELECTION We then evaluate the capability of our model to adapt to different ROI specifications when compressed videos are presented. We remark that this trait is appealing as it would elect our model as general purpose, as the same trained model could be deployed for ROI-based compression in disparate use cases. We also notice how this feature is lacking in current works for neural codecs [3, 51], as they typically commit to specific semantic classes during optimization and are trained such that their encoder would implicitly recognize and favor important regions. On the contrary, our model is explicitly fed with a mask specifying the desired (non-)ROI areas, allowing us to compress the same video differently, depending on the desired ROI specifications.

We select several sequences from the DAVIS validation set (`dogs-jump`, `pigs` and `gold-fish`), being labeled with more than one instance. Instead of merging all instances into a single ROI mask (as we do in all other experiments), we compress the video multiple times by considering different instances as ROI in different runs. We consistently monitor PSNR on all instances and observe that it is consistently higher in the region considered as ROI. We represent these results color-coded in

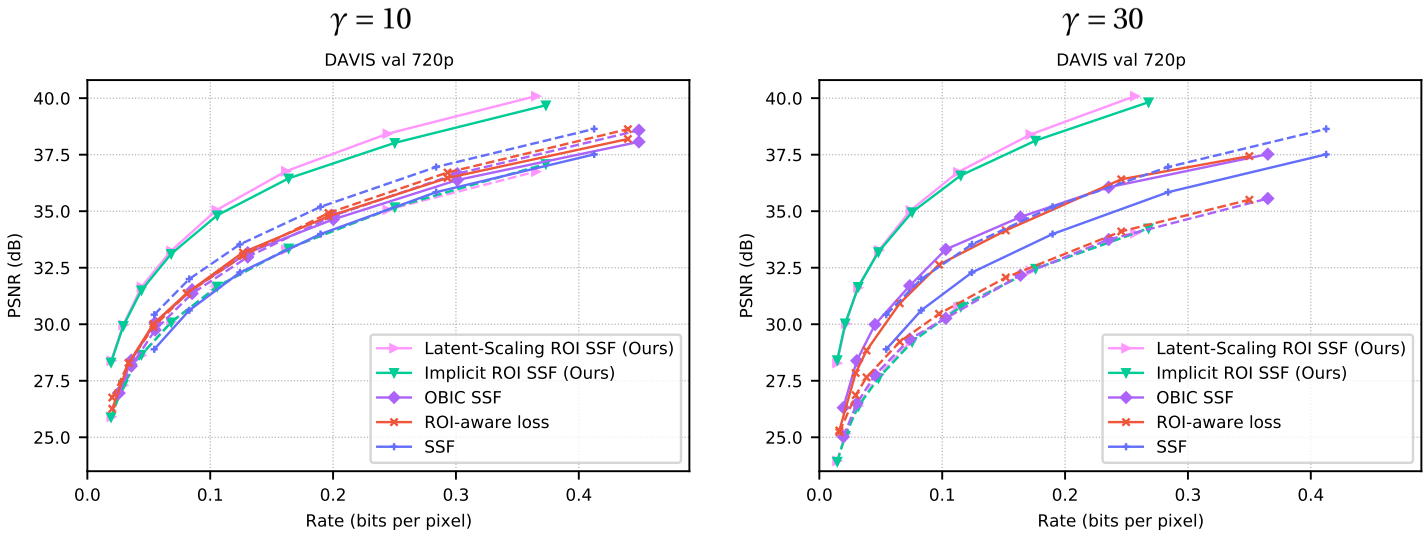


Figure D.2: All ROI-based neural video compression approaches vs SSF, trained on DAVIS and evaluated on DAVIS val. ROI-based models are trained with $\gamma = \{10, 30\}$, left and right plots, respectively. The right plot is Figure 6.4.1 in the main text. The solid line denotes ROI-PSNR, while dashed non-ROI PSNR.

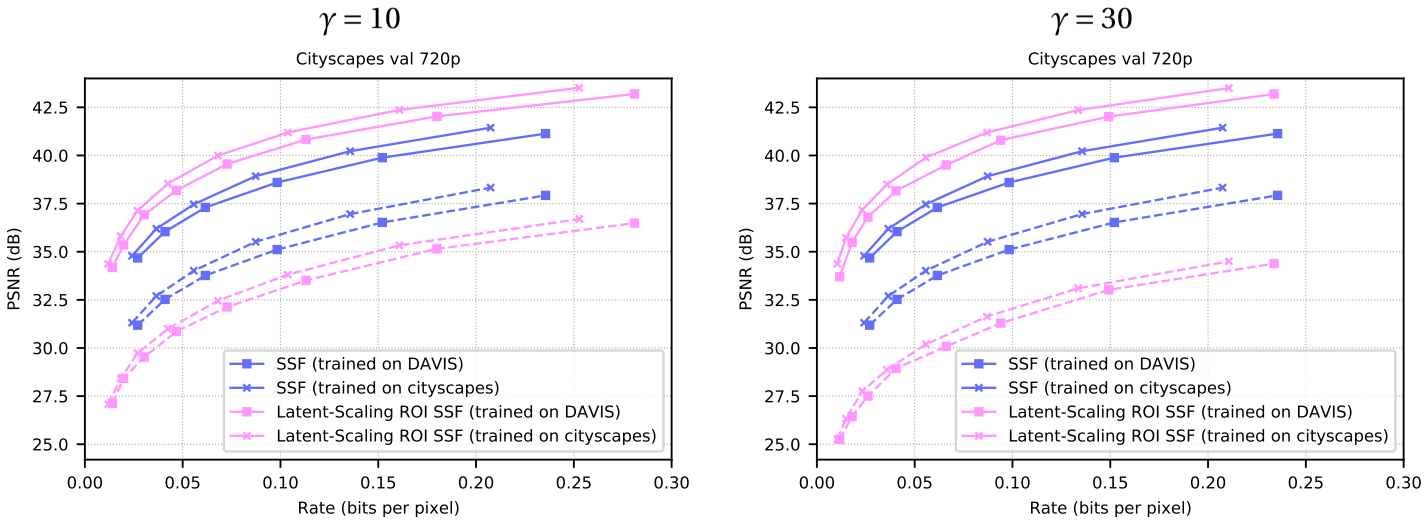


Figure D.3: SSF and Latent-scaling ROI SSF trained on either DAVIS or Cityscapes with ground-truth annotations and evaluated on Cityscapes val. Our models are trained with $\gamma = \{10, 30\}$, left and right plots, respectively. Right plot is Figure 6.4.4 in the main text. The solid line denotes ROI-PSNR, while the dashed non-ROI PSNR.

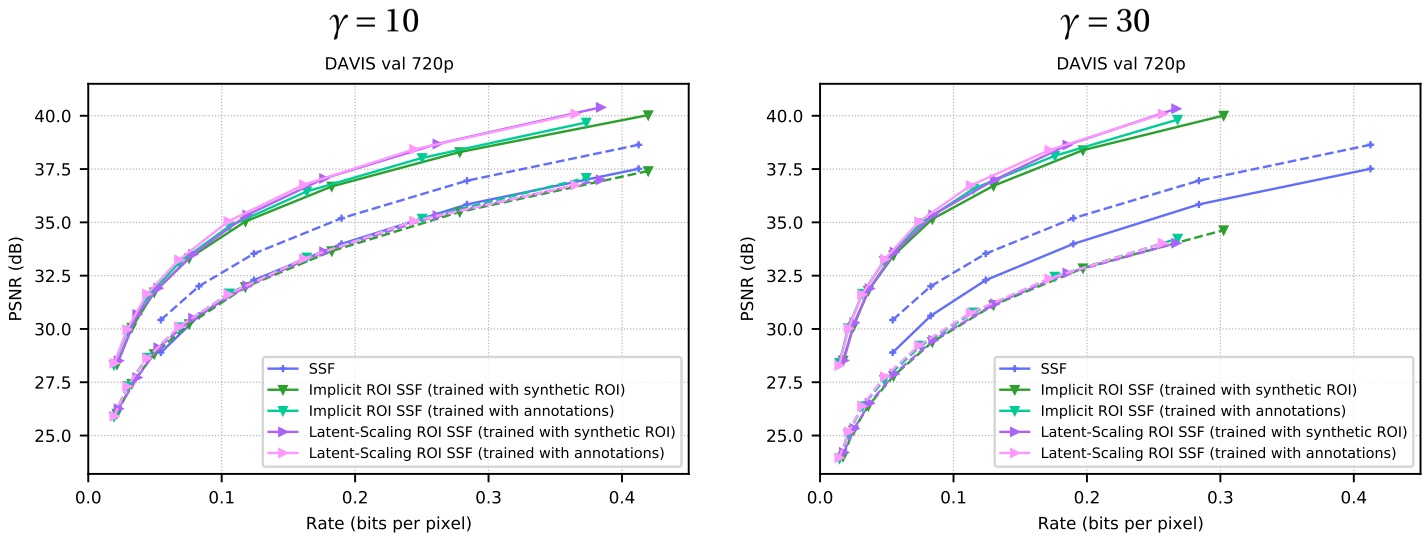


Figure D.4: Effect of training with synthetic ROI masks instead of ground-truth annotations on ROI PSNR R-D performance for DAVIS validation dataset. We show the implicit and latent-scaling ROI SSF versus the original SSF. Our models are trained with $\gamma = \{10, 30\}$, left and right plots, respectively. Right plot is a modified version of Figure 6.4.5 in the main text. The solid line denotes ROI-PSNR, while the dashed non-ROI PSNR.

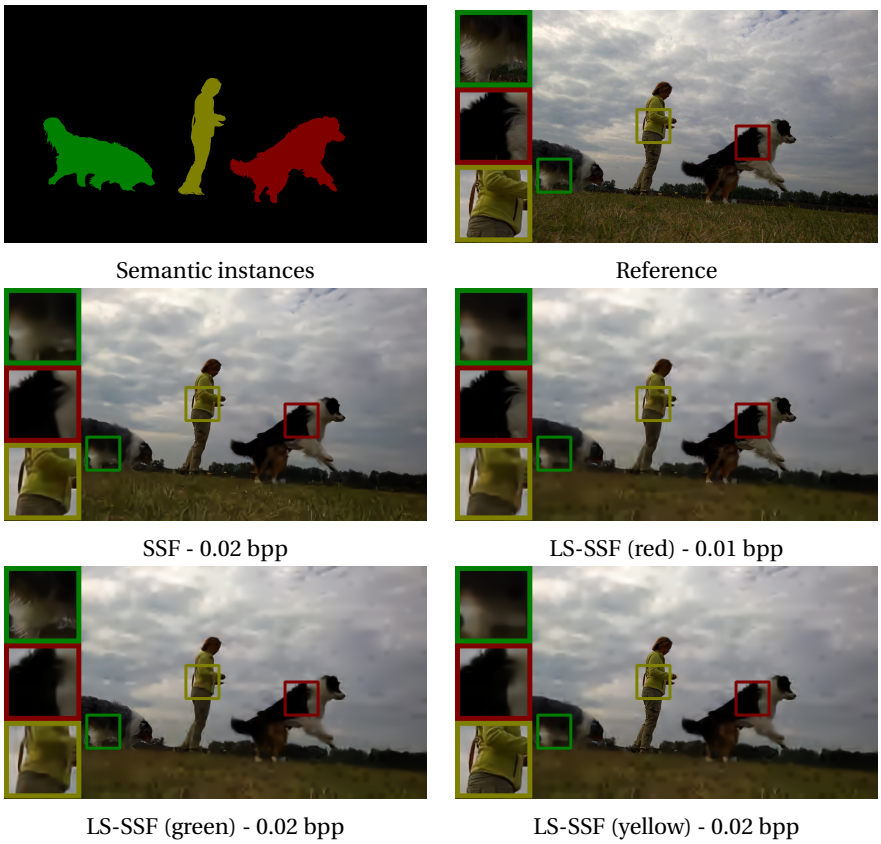


Figure D.5: ROI-coding of different foreground instances (red, green and yellow) in the 37-th frame of the "`dogs-jump`" sequence in the DAVIS validation set. The same pretrained latent-scaling ROI SSF model can be conditioned to achieve a higher ROI PSNR on different ROIs at eval time.

the barplots in Figure D.6. In all videos being considered, the instance considered as ROI benefits a boost of 5dB or more in PSNR. This result clearly shows that our codec can be used at approximately the same bitrate to improve reconstruction quality in any ROI of choice. A qualitative representation of such a feature is represented, for the `dogs-jump` sequence, in Figure D.5.

MULTIRATE CAPABILITIES We experimented with the "naive" latent-scaling technique described in Lu et al. [92]. With the use of a gain amplifier ga, it allows navigating different R-D tradeoffs with a single trained model during evaluation. The gain variable h output by the gain hyperprior AE is transformed using

$$\tilde{h} = (h-1)\cdot \text{ga} + 1 \tag{D.1}$$

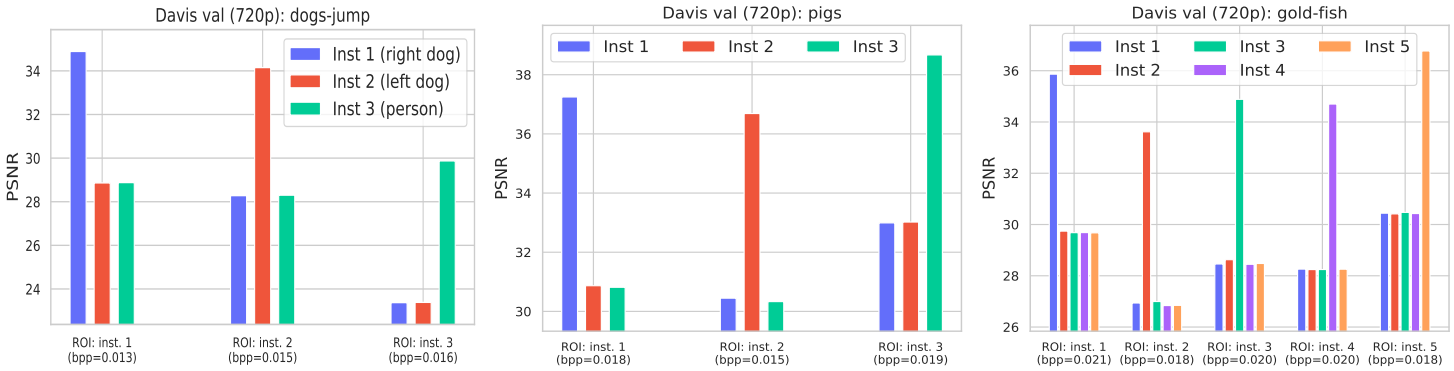


Figure D.6: PSNR of each instance when ROI-coding the different foreground instances in the "`dogs-jump`", "`pigs`" and "`gold-fish`" sequences in the DAVIS validation set.

before being used to scale the prior parameters and latent code, see Section 6.3 for details. Note that the higher the ga value, the coarser the quantization grid becomes, which in return is cheaper to encode.

In Figure D.7 we show the latent-scaling ROI SSF for different rate regularization coefficients β with gain amplifier ga = 1 in pink. In addition we select three trained models ($\beta \in \{0.0001, 0.0008, 0.0064\}$) and sweep the gain amplifier ga $\in \{1, 2, 4, 8, 16, 32, 64\}$; such curves are represented in red, purple and brown, and marked as "MR" (multirate) in the plot. The figure shows how, in general, the multi-rate curves can follow the baseline curve for several values of the gain amplifier before falling below it. This allows us to cover the target bpp range with 3 trained models instead of the 8 originally achieved by separate trainings. More specifically, for high bpps (β = 0.0001), we observe favorable performance for low values of the gain amplifier, with a severe drop as ga increases. We, however, appreciate that for higher compression rates ($\beta \in \{0.0008, 0.0064\}$), the MR curves closely resemble the one achieved by separate training. This shows promise for training a single model to support multiple bitrates by following training schemes as proposed in Cui et al. [33].

6

D.3 *Qualitative results*

In this Appendix, we provide additional visual results for several variants of the proposed ROI-based methods.

DIFFERENT BACKGROUND PENALTY In Figure D.8, we report for frames from the DAVIS validation set the ROI-based encodings achieved by Implicit ROI-SSF and Latent Scaling ROI-SSF at different values of the background penalty γ (Equation (6.3.5)). Such a hyperparameter controls to which extent background distortion can be de-emphasized to achieve (under rate constraints) a better quality in ROI regions.

TRAINING ON SYNTHETIC ROI MASKS As validated in Figure 6.4.5 and Figure D.4, our models can be trained even in the absence of pixel-level ROI masks, as synthetically generated ones can be used instead, with similar validation performances. In Figure D.9, we report some examples of encodings for comparable models when trained either on synthetic or ground truth masks. The visual quality of the resulting encoded frames appears comparable, confirming quantitative measurements.

Runtime performance

In table D.1, we benchmarked the runtime of SSF and Latent-Scaling ROI SSF on HD 720p inputs on an NVIDIA Tesla V100 and Intel CPU E5-1620 v4 @ 3.50GHz. We show timings in frames-per-second (FPS) for encode and decode operations: neural-network execution only, and together with entropy coding on CPU including data transfer, for I-frame and P-frame codec separately.

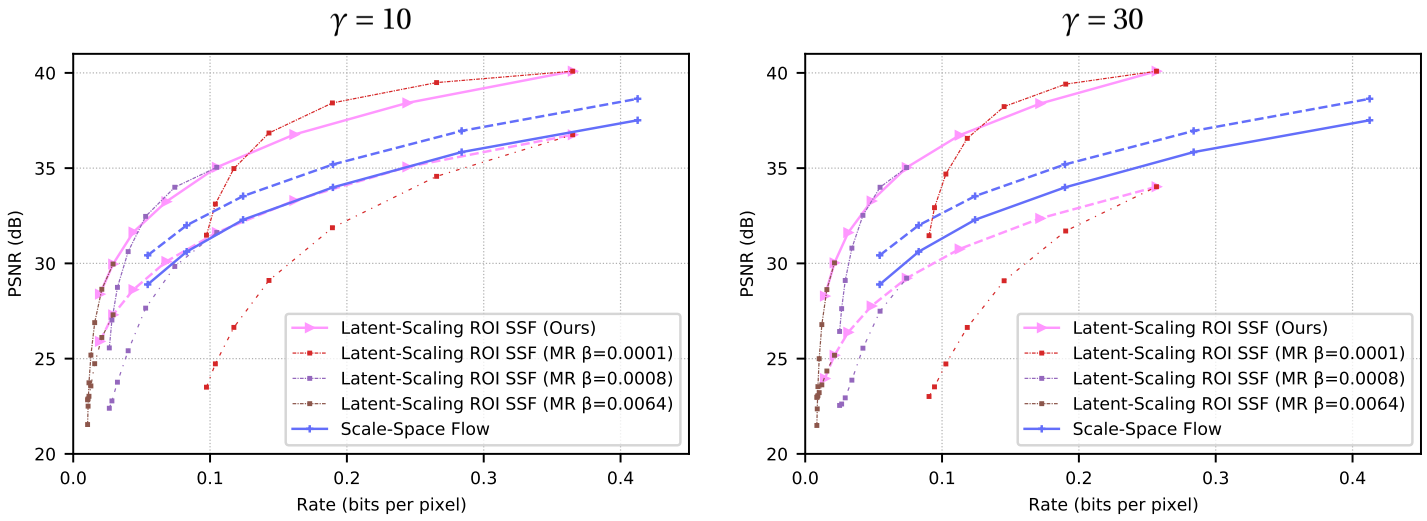


Figure D.7: Latent-scaling ROI SSF trained on DAVIS with ground-truth ROI masks evaluated on DAVIS val. ROI-based models are trained with $\gamma = \{10, 30\}$, left and right plots, respectively. The right plot is a modified version of Figure 6.1.1 in the main text, with additional curves obtained by multi-rate (MR). The solid line denotes ROI-PSNR, while the dashed non-ROI PSNR. The dashed-dotted lines, marked with "MR" in the legend, are obtained using a model trained for a single β, and then varying gain amplifier ga as outlined in Equation (D.1).

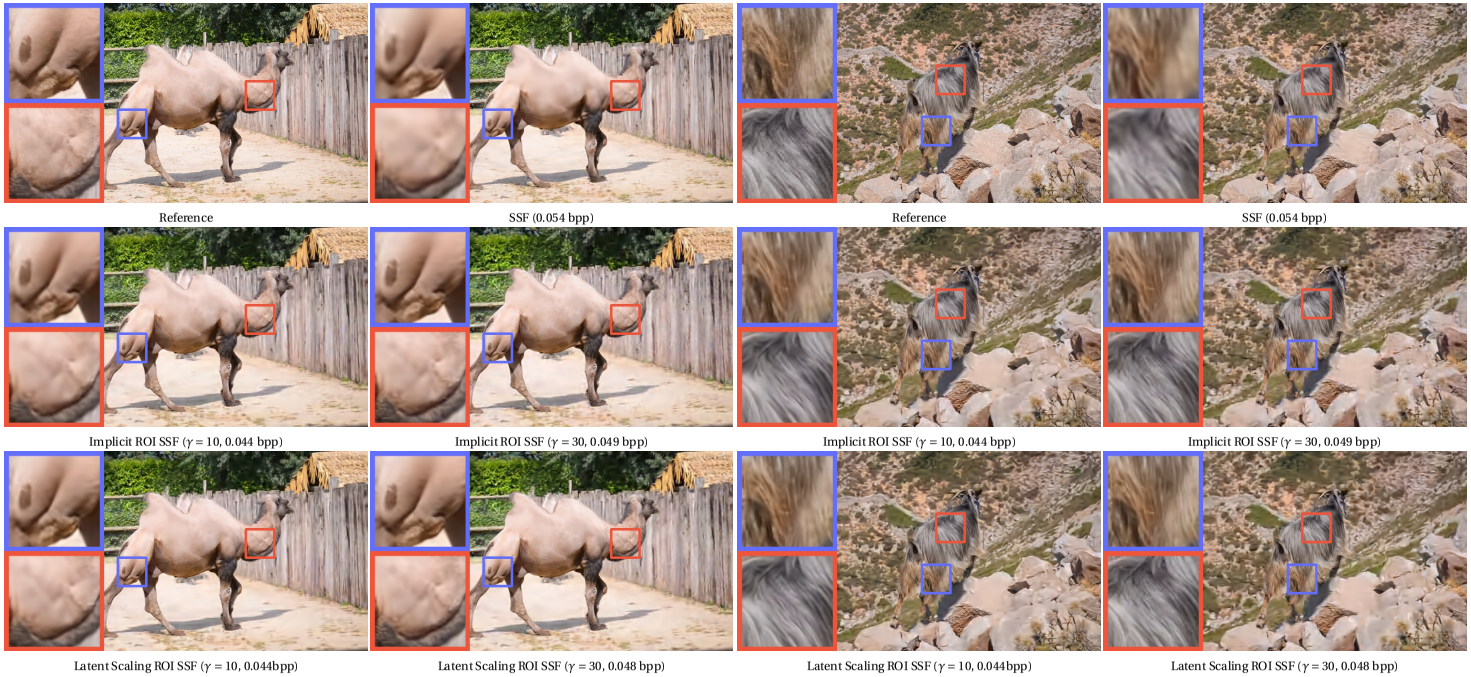


Figure D.8: Qualitative results of the Implicit and Latent-Scaling ROI SSF for $\gamma = \{10, 30\}$. Benchmarked against the SSF and the reference frame. We use the "`camel`" and "`goat`" sequences from the DAVIS validation set, at frames 11 and 5 respectively.

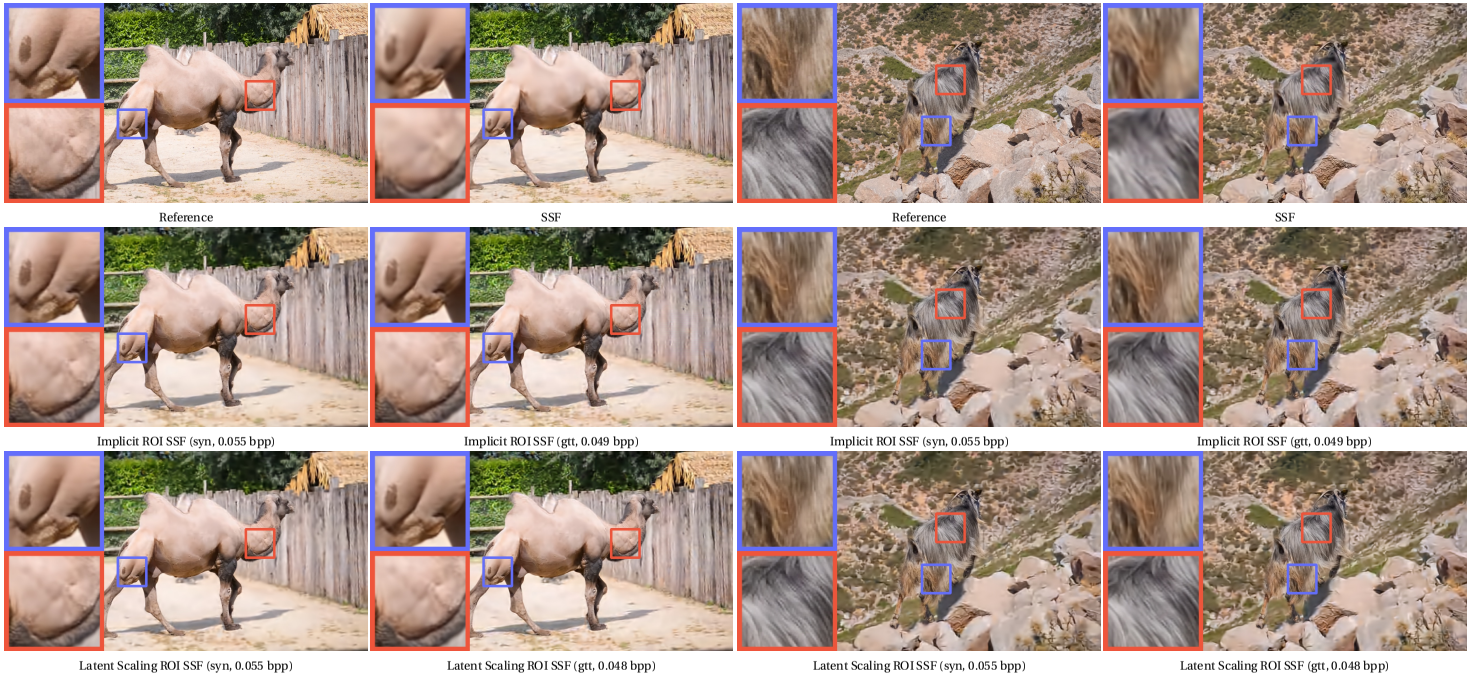


Figure D.9: Qualitative results of the Implicit and Latent-Scaling ROI SSF when trained using synthetic ROI maps (syn) or ground-truth ROI maps (gtt). We use the "`camel`" and "`goat`" sequences from the DAVIS validation set, at frames 11 and 5 respectively.

Note that the computational complexity of the Implicit ROI SSF is negligibly higher than that of the original SSF, as it only adds an input channel to each autoencoder.

Table D.1: Comparison of runtime (FPS) for 720p inputs of SSF and LS ROI SSF I/P-frame codecs on NVIDIA V100.

		Encode		Decode		Encode (no EC)		Decode (no EC)	
		I-frame	P-frame	I-frame	P-frame	I-frame	P-frame	I-frame	P-frame
SSF	FPS	3.5	1.7	3.8	1.8	378	192	682	340
LS ROI SSF	FPS	2.9	1.5	3.2	1.7	247	156	410	259
	FPS drop	-17%	-12%	-16%	-6%	-35%	-19%	-40%	-24%

D.4 *Architecture details*

We use the same SSF architecture as described in Pourreza and Cohen [106], Appendix A.1, except we share the hyperdecoder for mean and scale, and the last layer outputs twice as many channels. Our gain hyperprior autoencoder follows a similar architecture, except for the codec decoder which does not upsample and replaces transpose convolutions with regular convolutions with stride 1, see details in Figure D.10 for the codec and Figure D.11 for the hyper-codec.

We adopt the quantization strategy in Guo et al. [50]. Calling y the latent, we apply additive uniform noise ($\tilde{y} = y + u$ with $u \sim \mathcal{U}(-0.5, 0.5)$) and rounding with straight-through gradient estimation ($\bar{y} = \lfloor y \rceil$). During training, we use the noisy $\tilde{y}$ for the entropy computation in the prior, whereas we feed the decoder with the rounded latent $\bar{y}$. The same strategy holds for the hyper-latents.

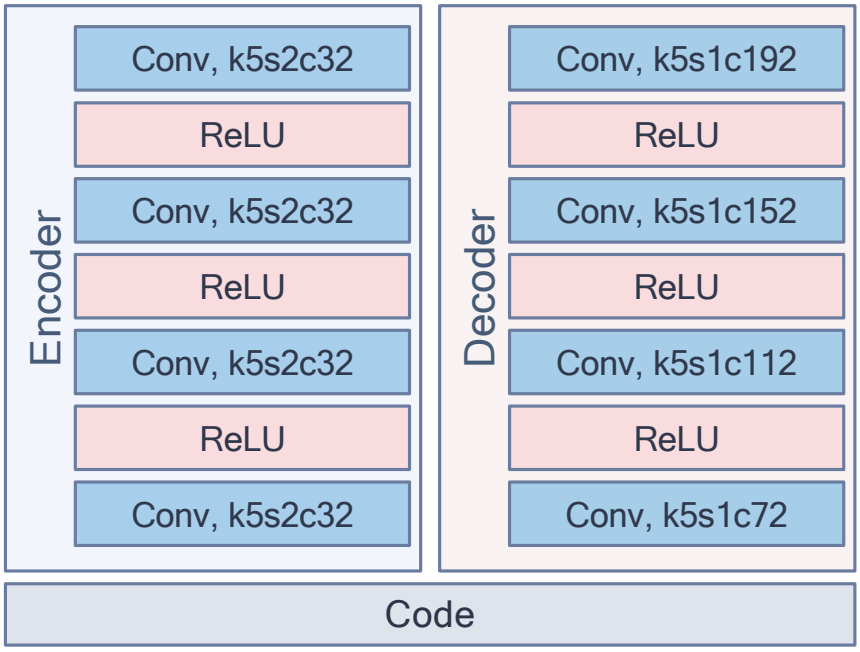


Figure D.10: Gain hyperprior codec details. k, s, and c denote kernel size, stride, and the number of output channels, respectively.

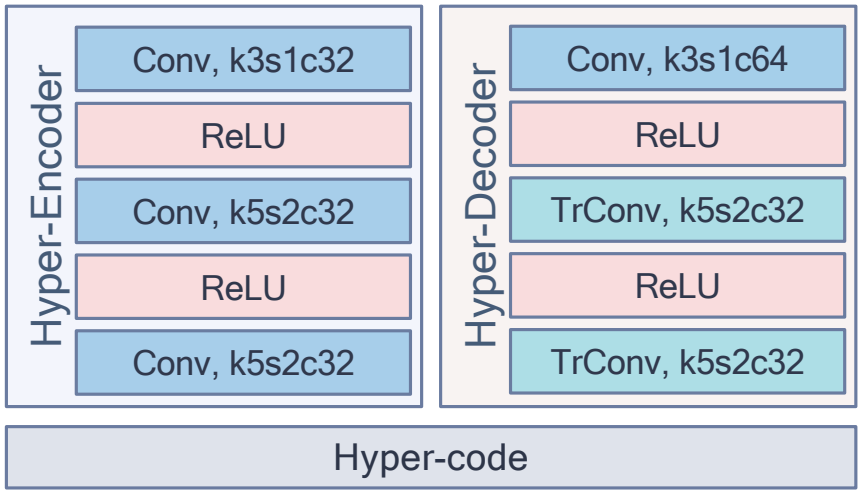


Figure D.11: Gain hyperprior hyper-codec details. k, s, and c denote kernel size, stride and the number of output channels, respectively.

BIBLIOGRAPHY

[1] E. Agustsson, F. Mentzer, M. Tschannen, L. Cavigelli, R. Timofte, L. Benini and L. V. Gool. 'Soft-to-Hard Vector Quantization for End-to-End Learning Compressible Representations'. In: *Proceedings of Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, NeurIPS, December 4-9, Long Beach, CA, USA*. 2017, pages 1141–1151.

[2] E. Agustsson, D. Minnen, N. Johnston, J. Balle, S. J. Hwang and G. Toderici. 'Scale-space flow for end-to-end optimized video compression'. In: *Proceedings of IEEE conference on Computer Vision and Pattern Recognition*. 2020.

[3] E. Agustsson, M. Tschannen, F. Mentzer, R. Timofte and L. V. Gool. 'Generative Adversarial Networks for Extreme Learned Image Compression'. In: *Proceedings of IEEE conference on Computer Vision and Pattern Recognition*. 2019.

[4] M. Akbari, J. Liang and J. Han. 'DSSLIC: Deep semantic segmentation-based layered image compression'. In: *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*. 2019.

[5] C. Anil, J. Lucas and R. B. Grosse. 'Sorting Out Lipschitz Function Approximation'. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June, Long Beach, California, USA*. Volume 97. Proceedings of Machine Learning Research. PMLR, 2019, pages 291–301.

[6] N. Asuni and A. Giachetti. 'TESTIMAGES: A large-scale archive for testing visual devices and basic image processing algorithms (SAMPLING 1200 RGB set)'. In: *Proceedings of STAG: Smart Tools and Apps for Graphics*. 2014.

[7] C. Aytekin, X. Ni, F. Cricri, J. Lainema, E. Aksu and M. Hannuksela. 'Block-optimized variable bit rate neural image compression'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pages 2551–2554.

[8] J. Ballé, V. Laparra and E. P. Simoncelli. 'End-to-end Optimized Image Compression'. In: *Proceedings of the 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, Conference Track Proceedings*. OpenReview.net, 2017.

[9] J. Ballé, D. Minnen, S. Singh, S. J. Hwang and N. Johnston. 'Variational image compression with a scale hyperprior'. In: *arXiv preprint arXiv:1802.01436* (2018).

[10] J. Bégaint, F. Racapé, S. Feltman and A. Pushparaja. 'CompressAI: a PyTorch library and evaluation platform for end-to-end compression research'. In: *arXiv preprint arXiv:2011.03029* (2020).

[11] J. Behrmann, W. Grathwohl, R. T. Chen, D. Duvenaud and J.-H. Jacobsen. 'Invertible residual networks'. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June, Long Beach, California, USA*. Volume 97. Proceedings of Machine Learning Research. PMLR, 2019, pages 573–582.

[12] F. Bellard. *BPG Specification*. (accessed June 3, 2020). 2014.

[13] Y. Bengio, N. Léonard and A. Courville. 'Estimating or propagating gradients through stochastic neurons for conditional computation'. In: *arXiv preprint arXiv:1308.3432* (2013).

[14] R. v. d. Berg, L. Hasenclever, J. M. Tomczak and M. Welling. 'Sylvester normalizing flows for variational inference'. In: *arXiv preprint arXiv:1803.05649* (2018).

[15] C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[16] G. Bjontegaard. 'Calculation of average PSNR differences between RD-curves'. In: *VCEG-M33* (2001).

[17] F. Bossen. *Common test conditions and software reference configurations*. JCTVC-F900. 2011.

[18] L. Bottou. 'Large-Scale Machine Learning with Stochastic Gradient Descent'. In: *19th International Conference on Computational Statistics, COMPSTAT 2010, Paris, France, August 22-27, 2010 - Keynote, Invited and Contributed Papers*. Physica-Verlag, 2010, pages 177–186.

[19] C. D. Brown and H. T. Davis. 'Receiver operating characteristics curves and related decision measures: A tutorial'. In: *Chemometrics and Intelligent Laboratory Systems* 80.1 (2006), pages 24–38.

[20] M. Buda, A. Maki and M. A. Mazurowski. 'A systematic study of the class imbalance problem in convolutional neural networks'. In: *Neural Networks* 106 (2018), pages 249–259.

[21] C. Cai, L. Chen, X. Zhang and Z. Gao. 'End-to-End Optimized ROI Image Compression'. In: *IEEE Transactions on Image Processing* 29 (2020), pages 3442–3457.

[22] J. Campos, S. Meierhans, A. Djelouah and C. Schroers. 'Content adaptive optimization for neural image compression'. In: *arXiv preprint arXiv:1906.01223* (2019).

[23] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille. 'DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs'. In: *IEEE International Conference on Acoustics, Speech and Signal Processing* 40.4 (2018), pages 834–848.

[24] R. T. Q. Chen, J. Behrmann, D. Duvenaud and J. Jacobsen. 'Residual Flows for Invertible Generative Modeling'. In: *Proceedings of Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, Vancouver, BC, Canada*. 2019, pages 9913–9923.

[25] T. Chen, H. Liu, Z. Ma, Q. Shen, X. Cao and Y. Wang. 'Neural image compression via non-local attention optimization and improved context modeling'. In: *arXiv preprint arXiv:1910.06244* (2019).

[26] T. Chen and Z. Ma. 'Variable Bitrate Image Compression with Quality Scaling Factors'. In: *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8*. IEEE, 2020, pages 2163–2167.

[27] Z. Chen, J. Han and K. N. Ngan. 'Dynamic Bit Allocation for Multiple Video Object Coding'. In: *IEEE Transactions on Multimedia* 8.6 (2006), pages 1117–1124.

[28] P. Chrabaszcz, I. Loshchilov and F. Hutter. 'A downsampled variant of imagenet as an alternative to the cifar datasets'. In: *arXiv preprint arXiv:1707.08819* (2017).

[29] A. Clark. *Pillow (PIL Fork) Documentation*. 2015.

[30] D. Claudiu Ciresan, U. Meier, L. M. Gambardella and J. Schmidhuber. 'Deep big simple neural nets excel on handwritten digit recognition'. In: *arXiv preprint arXiv:1003.0358* (2010).

B

[31] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth and B. Schiele. 'The Cityscapes Dataset for Semantic Urban Scene Understanding'. In: *Proceedings of IEEE conference on Computer Vision and Pattern Recognition*. 2016.

[32] C. Cremer, X. Li and D. Duvenaud. 'Inference suboptimality in variational autoencoders'. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15*. Volume 80. Proceedings of Machine Learning Research. PMLR, 2018, pages 1086–1094.

[33] Z. Cui, J. Wang, B. Bai, T. Guo and Y. Feng. 'G-VAE: A continuously variable rate deep image compression framework'. In: *arXiv preprint arXiv:2003.02012* (2020).

[34] N. De Cao, W. Aziz and I. Titov. 'Block neural autoregressive flow'. In: *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25*. Volume 115. Proceedings of Machine Learning Research. AUAI Press, 2019, pages 1263–1273.

[35] L. Dinh, D. Krueger and Y. Bengio. 'NICE: Non-linear Independent Components Estimation'. In: *arXiv:1410.8516* (2015).

[36] L. Dinh, J. Sohl-Dickstein and S. Bengio. 'Density estimation using Real NVP'. In: *arXiv:1605.08803* (2017).

[37] S. Duan, H. Chen and J. Gu. 'JPAD-SE: High-Level Semantics for Joint Perception-Accuracy-Distortion Enhancement in Image Compression'. In: *arXiv preprint arXiv:2005.12810* (2020).

[38] Y. Duan, Y. Zhang, X. Tao, C. Han, M. Xu, C. Yang and J. Lu. 'Content-aware Deep Perceptual Image Compression'. In: *Proceedings of the 11th International Conference on Wireless Communications and Signal Processing, WCSP 2019, Xi'an, China, October 23-25*. IEEE, 2019, pages 1–6.

[39] E. Dupont, A. Golinski, M. Alizadeh, Y. W. Teh and A. Doucet. 'COIN: COmpression with Implicit Neural representations'. In: *CoRR* abs/2103.03123 (2021).

[40] A. El-Nouby, M. J. Muckley, K. Ullrich, I. Laptev, J. Verbeek and H. Jégou. 'Image Compression with Product Quantized Masked Image Modeling'. In: *Trans. Mach. Learn. Res.* 2023 (2023).

[41] K. P. Ferentinos. 'Deep learning models for plant disease detection and diagnosis'. In: *Computers and Electronics in Agriculture* 145 (2018), pages 311–318.

[42] E. Fetaya, J.-H. Jacobsen, W. Grathwohl and R. Zemel. 'Understanding the Limitations of Conditional Generative Models'. In: *arXiv:1906.01171* (2019).

[43] Y. Ganesh, R. P. Singh and G. R. Murthy. 'Pattern classification using quadratic neuron: An experimental study'. In: *Proceedings of the 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE. 2017, pages 1–6.

[44] A. Golinski, R. Pourreza, Y. Yang, G. Sautière and T. S. Cohen. 'Feedback Recurrent Autoencoder for Video Compression'. In: *Proceedings of Computer Vision - ACCV 2020 - 15th Asian Conference on Computer Vision, Kyoto, Japan, November 30 - December 4, Revised Selected Papers, Part IV*. Volume 12625. Lecture Notes in Computer Science. Springer, 2020, pages 591–607.

[45] I. Goodfellow, Y. Bengio and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[46] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio. 'Generative adversarial networks'. In: *Communications of the ACM* 63.11 (2020), pages 139–144.

[47] H. Gouk, E. Frank, B. Pfahringer and M. Cree. 'Regularisation of neural networks by enforcing Lipschitz continuity'. In: *arXiv preprint arXiv:1804.04368* (2018).

[48] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever and D. Duvenaud. 'FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models'. In: *Proceedings of the 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9*. OpenReview.net, 2019.

[49] T. Guo, J. Wang, Z. Cui, Y. Feng, Y. Ge and B. Bai. 'Variable Rate Image Compression with Content Adaptive Optimization'. In: *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19*. Computer Vision Foundation / IEEE, 2020, pages 533–537.

[50] Z. Guo, Z. Zhang, R. Feng and Z. Chen. 'Soft then hard: Rethinking the quantization in neural image compression'. In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July, Virtual Event*. Volume 139. Proceedings of Machine Learning Research. PMLR, 2021, pages 3920–3929.

[51] A. Habibian, T. v. Rozendaal, J. M. Tomczak and T. S. Cohen. 'Video compression with rate-distortion autoencoders'. In: *Proceedings of IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2*. IEEE, 2019, pages 7032–7041.

[52] K. Hajian-Tilaki. 'Receiver operating characteristic (ROC) curve analysis for medical diagnostic test evaluation'. In: *Caspian Journal of Internal Medicine* 4.2 (2013), page 627.

[53] S. Han and N. Vasconcelos. 'Image compression using object-based regions of interest'. In: *2006 International Conference on Image Processing*. IEEE. 2006, pages 3097–3100.

[54] S. Han and N. Vasconcelos. 'Object-Based Regions of Interest for Image Compression'. In: *Proceedings of Data Compression Conference (DCC 2008), 25-27 March 2008, Snowbird, UT, USA*. IEEE Computer Society, 2008, pages 132–141.

[55] D. He, Z. Yang, W. Peng, R. Ma, H. Qin and Y. Wang. 'ELIC: Efficient Learned Image Compression with Unevenly Grouped Space-Channel Contextual Adaptive Coding'. In: *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24*. IEEE, 2022, pages 5708–5717.

[56] K. He, X. Zhang, S. Ren and J. Sun. 'Deep residual learning for image recognition'. In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30*. 2016, pages 770–778.

[57] J. Ho, X. Chen, A. Srinivas, Y. Duan and P. Abbeel. 'Flow++: Improving flow-based generative models with variational dequantization and architecture design'. In: *arXiv preprint arXiv:1902.00275* (2019).

[58] E. Hoogeboom, V. G. Satorras, J. M. Tomczak and M. Welling. 'The Convolution Exponential and Generalized Sylvester Flows'. In: *arXiv preprint arXiv:2006.01910* (2020).

[59] Z. Hu, G. Lu, J. Guo, S. Liu, W. Jiang and D. Xu. 'Coarse-to-fine deep video coding with hyperprior-guided mode prediction'. In: *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24*. IEEE, 2022, pages 5911–5920.

[60] G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger. 'Densely connected convolutional networks'. In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26*. IEEE Computer Society, 2017, pages 2261–2269.

[61] M. F. Hutchinson. 'A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines'. In: *Communications in Statistics-Simulation and Computation* 19.2 (1990), pages 433–450.

[62] L. Itti. 'Automatic foveation for video compression using a neurobiological model of visual attention'. In: *IEEE Transactions on Image Processing* 13.10 (2004), pages 1304–1318.

[63] J.-H. Jacobsen, J. Behrmann, R. Zemel and M. Bethge. 'Excessive Invariance Causes Adversarial Vulnerability'. In: *arXiv:1811.00401* (2018).

[64] E. Jang, S. Gu and B. Poole. 'Categorical reparameterization with gumbel-softmax'. In: *arXiv preprint arXiv:1611.01144* (2016).

[65] H. Kahn. 'Use of different Monte Carlo sampling techniques'. In: *Proceedings of Symposium on Monte Carlo Methods* (1955).

[66] A. Kalisz, J. Kostrzewa, A. Sekara, A. Grabowska and S. Cebula. 'Yield and nutritional quality of several non-heading Chinese cabbage (Brassica rapa var. chinensis) cultivars with different growing period and its modelling'. In: *Horticultural Science & Technology* 30.6 (2012), pages 650–656.

[67] D. P. Kingma and M. Welling. 'Auto-encoding variational Bayes'. In: *arXiv preprint arXiv:1312.6114* (2013).

[68] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2014.

[69] D. P. Kingma and P. Dhariwal. 'Glow: Generative flow with invertible 1x1 convolutions'. In: *Proceedings of Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, Montréal, Canada*. 2018, pages 10236–10245.

[70] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever and M. Welling. 'Improved variational inference with inverse autoregressive flow'. In: *Advances in Neural Information Processing Systems*. 2016, pages 4743–4751.

[71] E. Kodak. *Kodak Lossless True Color Image Suite (PhotoCD PCD0992)*.

[72] A. Krizhevsky. *Learning multiple layers of features from tiny images*. Technical report. University of Toronto, 2009.

[73] A. Krizhevsky, G. Hinton et al. 'Learning multiple layers of features from tiny images'. In: (2009).

[74] A. Krizhevsky, I. Sutskever and G. E. Hinton. 'Imagenet classification with deep convolutional neural networks'. In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems. Proceedings of a meeting held December 3-6, Lake Tahoe, Nevada, United States*. 2012, pages 1106–1114.

[75] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. C. Lopez and J. V. Soares. 'Leafsnap: A computer vision system for automatic plant species identification'. In: *Computer Vision - ECCV 2012 - 12th European Conference on Computer Vision, Florence, Italy, October 7-13, Proceedings, Part II*. Volume 7573. Lecture Notes in Computer Science. Springer, 2012, pages 502–516.

[76] Q. Lai, W. Wang, H. Sun and J. Shen. 'Video Saliency Prediction using Spatiotemporal Residual Attentive Networks'. In: *IEEE Transactions on Image Processing* 29 (2020), pages 1113–1126.

[77] D. C. Lay. *Linear Algebra and its Applications. The Science of Microfabrication*. Pearson, 2006.

[78] T.-N. Le and A. Sugimoto. 'Video salient object detection using spatiotemporal deep features'. In: *IEEE Transactions on Image Processing* 27.10 (2018), pages 5002–5015.

[79] Y. LeCun, Y. Bengio and G. Hinton. 'Deep learning'. In: *Nature* 521.7553 (2015), pages 436–444.

[80] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel. 'Backpropagation applied to handwritten zip code recognition'. In: *Neural computation* 1.4 (1989), pages 541–551.

[81] Y. LeCun, C. Cortes and C. Burges. 'MNIST handwritten digit database'. In: *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist* 2 (2010).

[82] J. Lee, S. Cho and S.-K. Beack. 'Context-adaptive Entropy Model for End-to-end Optimized Image Compression'. In: *Proceedings of the 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9*. OpenReview.net, 2019.

[83] S. H. Lee, C. S. Chan, P. Wilkin and P. Remagnino. 'Deep-Plant: Plant Identification with convolutional neural networks'. In: *Proceedings of IEEE International Conference on Image Processing, ICIP 2015, Quebec City, QC, Canada, September 27-30*. IEEE, 2015, pages 452–456.

[84] J. L. Leevy, T. M. Khoshgoftaar, R. A. Bauder and N. Seliya. 'A survey on addressing high-class imbalance in big data'. In: *Journal of Big Data* 5.42 (2018).

[85] M. Li, W. Zuo, S. Gu, D. Zhao and D. Zhang. 'Learning Convolutional Networks for Content-Weighted Image Compression'. In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22*. Computer Vision Foundation / IEEE Computer Society, 2018, pages 3214–3223.

[86] Q. Li, S. Haque, C. Anil, J. Lucas, R. B. Grosse and J.-H. Jacobsen. 'Preventing gradient attenuation in Lipschitz constrained convolutional networks'. In: *Proceedings of Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS 2019, December 8-14, Vancouver, BC, Canada*. 2019, pages 15364–15376.

[87] S. Li and W. Li. 'Shape-adaptive discrete wavelet transforms for arbitrarily shaped visual object coding'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 10.5 (2000), pages 725–743.

[88] Z. Li, A. Aaron, I. Katsavounidis, A. Moorthy, M. Manohara et al. 'Toward a practical perceptual video quality metric'. In: *The Netflix Tech Blog* 6.2 (2016), page 2.

[89] J. Lin, D. Liu, H. Li and F. Wu. 'M-LVC: Multiple Frames Prediction for Learned Video Compression'. In: *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19*. Computer Vision Foundation / IEEE, 2020, pages 3543–3551.

[90] G. Lu, C. Cai, X. Zhang, L. Chen, W. Ouyang, D. Xu and Z. Gao. 'Content adaptive and error propagation aware deep video compression'. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, Proceedings, Part II 16*. Springer. 2020, pages 456–472.

[91] G. Lu, W. Ouyang, D. Xu, X. Zhang, C. Cai and Z. Gao. 'Dvc: An end-to-end deep video compression framework'. In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20*. Computer Vision Foundation / IEEE, 2019, pages 11006–11015.

[92] Y. Lu, Y. Zhu, Y. Yang, A. Said and T. S. Cohen. 'Progressive Neural Image Compression with Nested Quantization and Latent Ordering'. In: *arXiv preprint arXiv:2102.02913* (2021).

B

[93] D. J. MacKay and M. N. Gibbs. 'Density networks'. In: *Statistics and neural networks: advances at the interface. Oxford University Press, Oxford* (1999), pages 129–144.

[94] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte and L. Van Gool. 'Conditional probability models for deep image compression'. In: *Proceedings of IEEE conference on Computer Vision and Pattern Recognition*. 2018.

[95] D. Minnen, J. Ballé and G. Toderici. 'Joint autoregressive and hierarchical priors for learned image compression'. In: *Neural Information Processing Systems* (2018).

[96] T. Miyato, T. Kataoka, M. Koyama and Y. Yoshida. 'Spectral normalization for generative adversarial networks'. In: *arXiv preprint arXiv:1802.05957* (2018).

[97] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller. 'Playing atari with deep reinforcement learning'. In: *arXiv preprint arXiv:1312.5602* (2013).

[98] E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur and B. Lakshminarayanan. 'Hybrid Models with Deep and Invertible Features'. In: *International Conference on Machine Learning*. 2019.

[99] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior and K. Kavukcuoglu. 'Wavenet: A generative model for raw audio'. In: *arXiv preprint arXiv:1609.03499* (2016).

[100] A. v. d. Oord, N. Kalchbrenner and K. Kavukcuoglu. 'Pixel Recurrent Neural Networks'. In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24*. Volume 48. JMLR Workshop and Conference Proceedings. JMLR.org, 2016, pages 1747–1756.

[101] S. J. Pan and Q. Yang. 'A survey on transfer learning'. In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2009), pages 1345–1359.

[102] G. Papamakarios, T. Pavlakou and I. Murray. 'Masked autoregressive flow for density estimation'. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, NeurIPS 2017, December 4-9, Long Beach, CA, USA*. 2017, pages 2338–2347.

[103] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga and A. Lerer. 'Automatic differentiation in PyTorch'. In: (2017).

[104] K. Perlin. 'An image synthesizer'. In: *ACM Siggraph Computer Graphics* 19.3 (1985), pages 287–296.

[105] J. Pont-Tuset, F. Perazzi, S. Caelles, P. Arbeláez, A. Sorkine-Hornung and L. Van Gool. 'The 2017 DAVIS Challenge on Video Object Segmentation'. In: *arXiv:1704.00675* (2017).

[106] R. Pourreza and T. S. Cohen. 'Extending Neural P-frame Codecs for B-frame Coding'. In: *Proceedings of IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 2021, pages 6660–6669.

[107] M. Prabhakar, Y. G. Prasad, S. Desai, M. Thirupathi, K. Gopika, G. R. Rao and B. Venkateswarlu. 'Hyperspectral remote sensing of yellow mosaic severity and associated pigment losses in Vigna mungo using multinomial logistic regression models'. In: *Crop Protection* 45 (2013), pages 132–140.

[108] D. J. Rezende and S. Mohamed. 'Variational inference with normalizing flows'. In: *arXiv preprint arXiv:1505.05770* (2015).

[109] D. J. Rezende, S. Mohamed and D. Wierstra. 'Stochastic backpropagation and approximate inference in deep generative models'. In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June*. Volume 32. JMLR Workshop and Conference Proceedings. JMLR.org, 2014, pages 1278–1286.

[110] O. Rippel and R. P. Adams. 'High-dimensional probability estimation with deep density models'. In: *arXiv preprint arXiv:1302.5125* (2013).

[111] O. Rippel, A. G. Anderson, K. Tatwawadi, S. Nair, C. Lytle and L. Bourdev. 'ELF-VC: Efficient Learned Flexible-Rate Video Coding'. In: *arXiv preprint arXiv:2104.14335* (2021).

[112] O. Rippel, S. Nair, C. Lew, S. Branson, A. G. Anderson and L. Bourdev. 'Learned Video Compression'. In: *Proceedings of IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2*. IEEE, 2019, pages 3453–3462.

[113] R. Rombach, A. Blattmann, D. Lorenz, P. Esser and B. Ommer. 'High-resolution image synthesis with latent diffusion models'. In: *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24*. IEEE, 2022, pages 10674–10685.

[114] O. Russakovsky et al. 'ImageNet Large Scale Visual Recognition Challenge'. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pages 211–252.

[115] W. Shang, K. Sohn, D. Almeida and H. Lee. 'Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units'. In: *arXiv:1603.05201* (2016).

[116] C. Shorten and T. M. Khoshgoftaar. 'A survey on image data augmentation for deep learning'. In: *Journal of Big Data* 6.1 (2019), page 60.

[117] T. Sikora and B. Makai. 'Shape-adaptive DCT for generic coding of video'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 5.1 (1995), pages 59–62.

[118] K. Simonyan and A. Zisserman. 'Very deep convolutional networks for large-scale image recognition'. In: *arXiv preprint arXiv:1409.1556* (2014).

[119] J. Skilling. 'The eigenvalues of mega-dimensional matrices'. In: *Maximum Entropy and Bayesian Methods*. Springer, 1989, pages 455–466.

[120] A. Skodras, C. Christopoulos and T. Ebrahimi. 'The JPEG 2000 still image compression standard'. In: *IEEE Signal Processing Magazine* 18.5 (2001), pages 36–58.

[121] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich. 'Going deeper with convolutions'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pages 1–9.

[122] E. G. Tabak and C. V. Turner. 'A family of nonparametric density estimation algorithms'. In: *Communications on Pure and Applied Mathematics* 66.2 (2013), pages 145–164.

[123] E. G. Tabak, E. Vanden-Eijnden et al. 'Density estimation by dual ascent of the log-likelihood'. In: *Communications in Mathematical Sciences* 8.1 (2010), pages 217–233.

[124] A. Tao, K. Sapra and B. Catanzaro. 'Hierarchical multi-scale attention for semantic segmentation'. In: *arXiv preprint arXiv:2005.10821* (2020).

[125] N. Teimouri, M. Dyrmann, P. Nielsen, S. Mathiassen, G. Somerville and R. Jørgensen. 'Weed growth stage estimator using deep convolutional neural networks'. In: *Sensors* 18.5 (2018), pages 1–13.

[126] L. Theis, W. Shi, A. Cunningham and F. Huszár. 'Lossy image compression with compressive autoencoders'. In: *arXiv preprint arXiv:1703.00395* (2017).

[127] G. Toderici, W. Shi, R. Timofte, L. Theis, J. Balle, E. Agustsson, N. Johnston and F. Mentzer. *Workshop and Challenge on Learned Image Compression (CLIC2020)*. CVPR, 2020.

[128] G. Toderici, D. Vincent, N. Johnston, S. Jin Hwang, D. Minnen, J. Shor and M. Covell. 'Full resolution image compression with recurrent neural networks'. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pages 5306–5314.

[129] J. M. Tomczak and M. Welling. 'Improving variational auto-encoders using householder flow'. In: *arXiv preprint arXiv:1611.09630* (2016).

[130] Ultra Video Group. *UVG test sequences*. http://ultravideo.cs.tut.fi/. Accessed: 2020-02-21.

[131] A. van den Oord, O. Vinyals and K. Kavukcuoglu. 'Neural Discrete Representation Learning'. In: *Proceedings of Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, NeurIPS 2017, December 4-9, Long Beach, CA, USA*. 2017, pages 6306–6315.

[132] T. van Rozendaal, I. A. M. Huijben and T. Cohen. 'Overfitting for Fun and Profit: Instance-Adaptive Data Compression'. In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7*. OpenReview.net, 2021.

[133] A. Vetro, H. Sun and Y. Wang. 'MPEG-4 rate control for multiple video objects'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 9.1 (1999), pages 186–199.

[134] G. K. Wallace. 'The JPEG still picture compression standard'. In: *IEEE transactions on consumer electronics* 38.1 (1992), pages xviii–xxxiv.

[135] B. A. Wandell. 'Foundations of Vision MA Sunderland'. In: (1995).

[136] J. Wang et al. 'Deep High-Resolution Representation Learning for Visual Recognition'. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing* (2021).

[137] W. Wang, J. Shen, J. Xie, M. Cheng, H. Ling and A. Borji. 'Revisiting Video Saliency Prediction in the Deep Learning Era'. In: *Proceedings of the IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[138] W. Wang, J. Shen, F. Guo, M.-M. Cheng and A. Borji. 'Revisiting Video Saliency: A Large-Scale Benchmark and a New Model'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22*. Computer Vision Foundation / IEEE Computer Society, 2018, pages 4894–4903.

[139] S. C. Wong, A. Gatt, V. Stamatescu and M. D. McDonnell. 'Understanding data augmentation for classification: when to warp?' In: *2016 international conference on Digital Image Computing: Techniques and Applications (DICTA)*. IEEE. 2016, pages 1–6.

B

[140] C.-Y. Wu, N. Singhal and P. Krähenbühl. 'Video Compression Through Image Interpolation'. In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, Proceedings, Part VIII*. Volume 11212. Lecture Notes in Computer Science. Springer, 2018, pages 425–440.

[141] Q. Xia, H. Liu and Z. Ma. 'Object-Based Image Coding: A Learning-Driven Revisit'. In: *Proceedings of the IEEE International Conference on Multimedia and Expo, ICME 2020, London, UK, July 6-10*. IEEE. 2020, pages 1–6.

[142] Xiph.org. *Xiph.org Video Test Media [derf's collection]*. https://media.xiph.org/video/derf/. Accessed: 2020-02-21.

[143] T. Xue, B. Chen, J. Wu, D. Wei and W. T. Freeman. 'Video Enhancement with Task-Oriented Flow'. In: *International Journal of Computer Vision* 127.8 (2019), pages 1106–1125.

[144] Y. Yang, R. Bamler and S. Mandt. 'Improving inference for neural image compression'. In: *Advances in Neural Information Processing Systems* 33 (2020), pages 573–584.

[145] Y. Yang, S. Mandt, L. Theis et al. 'An introduction to neural data compression'. In: *Foundations and Trends® in Computer Graphics and Vision* 15.2 (2023), pages 113–200.

[146] P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi and J. Xin. 'Understanding straight-through estimator in training activation quantized neural nets'. In: *arXiv preprint arXiv:1903.05662* (2019).

[147] Y. Zhang, T. van Rozendaal, J. Brehmer, M. Nagel and T. Cohen. 'Implicit Neural Video Compression'. In: *CoRR* abs/2112.11312 (2021).

[148] Z. Zhang, H. Liu, Z. Meng and J. Chen. 'Deep learning-based automatic recognition network of agricultural machinery images'. In: *Computers and Electronics in Agriculture* 166 (2019).

[149] H. Zhao, J. Shi, X. Qi, X. Wang and J. Jia. 'Pyramid Scene Parsing Network'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26*. IEEE Computer Society, 2017, pages 6230–6239.

[150] Y.-Y. Zheng, J.-L. Kong, X.-B. Jin, X.-Y. Wang, T.-L. Su and M. Zuo. 'CropDeep: the crop vision dataset for deep-learning-based classification and detection in precision agriculture'. In: *Sensors* 19.5 (2019), page 1058.

[151] Y. Zhu, Y. Yang and T. Cohen. 'Transformer-based Transform Coding'. In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29*. OpenReview.net, 2022.

SUMMARY

Deep learning has become a major field with many applications: from face recognition to generating images to compressing data. As a result, deep learning is becoming more and more integrated into our daily lives. In this thesis, we demonstrate how deep learning can be deployed for several applications in three different domains namely, improving business processes for the agriculture, high-dimensional density estimation with generative models, and neural compression of data.

The first domain aims to optimize the business process of a seed breeding company operating in agriculture. Seed breeding ensures the development of seeds for the changing climate conditions, which is essential for the growing world population to guarantee food security and to preserve the biodiversity of the seeds. The main challenge for seed breeding companies is the utilization of deep learning to stay competitive with similar businesses, that aim to deliver high-quality seeds to their customers. In Part I of this thesis we examine a dataset of white cabbage seedling images. The aim is to predict the (un)successfulness of the seedlings based on only an image. Since accurate and early predictions can terminate the seedlings stay in a growth chamber, which provides more space for other seeds to grow. Further, automating the process aids professionals. We show how a particular convolutional neural network, AlexNet, outperforms the other machine learning methods and that the model can accurately determine if a seedling is going to grow (un)successfully. Moreover, we observe that training AlexNet on earlier days generalizes to predictions on later days.

The second domain concerns the utilization of generative modeling for high dimensional density estimation. Generative models may aid the development of drug discovery, anomaly detection, or simulation of complex environmental processes for climate patterns. The challenges lie in the difficulty and instability of training these models, the significant requirement of computational resources, and adaption to different datasets. Another important challenge is how to deal with malicious generation of fake content with deepfake videos which cause a serious concern for society. In Part II, we aid to close the gap in estimating the true data distribution that is modeled with generative models. More concretely, we improve model performance of a generative model, known as the normalizing flow. We based this work on the belief that DenseNets, most of the time, outperform ResNets on classification tasks. Since we are dealing with a special type of normalizing flow, we need to construct concatenations in such a way that they satisfy specific Lipschitz condi-

tions. Therefore, we construct a method that relies on an analysis of the Lipschitz continuity in DenseNets and enforces the invertibility of the network by satisfying the Lipschitz constant. Further, we propose a learnable weighted concatenation for the model and an activation function, which we call Concatenated LipSwish, that enforces the Lipschitz condition. The new architecture is known as i-DenseNet and outperforms its predecessor Residual Flow and other comparable flow-based models on generative and hybrid modeling performance.

Finally, the third domain covers the neural compression process for images and videos. With the growing amount of data worldwide, compression, in general, has become a fundamental part of data storage and transmission. Neural compression aims to contribute to optimizing digital content such as multimedia streaming, internet services, and storing data. Although neural compression contributes significantly, it also faces challenges. These large-scale models rely on large amounts of high-quality datasets and are difficult to efficiently deploy on (mobile) devices since they require a significant amount of computational resources. Part III consists of two chapters that cover the neural compression domain.

In the first chapter of Part III, we examine a neural image compression model, based on the mean-scale hyperprior. The mean-scale hyperprior has proven to be a successful model that outperforms classical methods in the image compression field. Even though these models are effective in practice, they do have limited capacity when it comes to optimization and generalization. Therefore, we introduce a new method that aids the compression performance and results in improved compression results per image. We aim to optimize the latents of an already pre-trained mean-scale hyperprior model, by keeping the networks weights fixed, and only further optimizing its latents with a refinement procedure. We introduce three different methods that built upon the stochastic Gumbel annealing method and show how it can be extended to three-class rounding. Further, we show how our proposed method, known as SGA+, outperforms the baselines on two different datasets. Additionally, we show how SGA+ can be used to move partly along the rate-distortion curve and how it is robust to hyperparameter changes.

Recent work has integrated neural image compression models that are capable of allocating more bits to pre-specified regions of interest. However, until now there was no neural video compression model capable of doing this. In the final chapter of Part III, we introduce a neural video compression model, based on scale-space flow, that allocates more bits to pre-specified regions-of-interest. We introduce two versions that are able to achieve this, namely, an implicit and a latent scaling model. In general, both models out-perform all baselines in terms of the rate-distortion performance in regions of interest and can generalize to different datasets at inference time. The latent scaling model has the best performance and can explicitly control the quantization binwidth of latent variables by only using a single model during evaluation. Further, we find that the models show a negligible performance gap when trained with synthetic region-of-interest masks, which do not correlate with the content of the video, compared to training with pixel-wise annotated masks.

SAMENVATTING

Deep learning is een breed veld met veel toepassingen: van gezichtsherkenning tot het genereren van plaatjes, tot aan het comprimeren van data. Deep learning is onderhand steeds meer geïntegreerd in ons dagelijks leven. Dit proefschrift laat zien hoe deep learning verschillende toepassingsmogelijkheden heeft in drie verschillende domeinen: het verbeteren van bedrijfsprocessen voor de landbouwsector, met generatieve modellen de hoog dimensionaliteit schatten, en met neurale netwerken data compressie toepassen.

In het eerste deel van dit proefschrift streven wij ernaar om bedrijfsprocessen van een zaadveredelingsbedrijf in de landbouwsector te optimaliseren. Zaadveredeling is een belangrijk proces dat probeert de ontwikkeling van zaden voor de klimaatsverandering te waarborgen, verder is het essentieel om de groeiende wereldbevolking te voorzien van voedsel, en beschermt het de biodiversiteit. Zaadveredelingsbedrijven hebben als doel om hoge kwaliteit zaden te leveren aan hun klanten. Om competitief te blijven met concurrerende bedrijven, is het van belang om deep learning te gebruiken voor de verbetering van de bedrijfsprocessen. In het eerste deel van dit proefschrift onderzoeken wij een dataset bestaande uit foto's van witte kool zaden. Het doel van het onderzoek is om de foto's van de groeiende kool zaden te classificeren. Er wordt gekeken of de kool goed of slecht groeit, op basis van alleen het plaatje. Daarbij leiden nauwkeurige en vroege voorspellingen tot extra ruimte in de groeikamer waar de zaadjes verblijven. Het automatiseren van zo'n proces helpt een vakman tijd te besparen. Wij laten zien hoe een specifiek convolutioneel netwerk, genaamd, AlexNet, beter presteert dan andere machine learning methodes. Verder laten wij zien dat het model accuraat kan bepalen of een kool plantje goed of slecht groeit. Daarnaast zien wij dat AlexNet, getraind op eerdere dagen, generaliseert naar voorspellingen op latere dagen.

Het tweede deel bestaat uit het generatief modelleren van de hoog-dimensionale dichtheid van de data. Generatieve modellen zouden kunnen bijdragen aan de ontwikkeling van nieuwe medicijnen, detectie van afwijkingen in data of het simuleren van complexe klimaatveranderingen. De uitdagingen van deze modellen zijn terug te vinden in moeilijkheden en instabiliteit tijdens het trainen van de netwerken. Verder vereisen de modellen een aanzienlijke hoeveelheid computer rekenkracht en passen zij zich moeilijk aan, aan ander soort dataset dan zij op getraind zijn. Een zorgwekkende ontwikkeling voor de maatschappij, die deze modellen met zich meebrengen, is de vraag hoe er omgegaan moet worden met het genereren van on-

NL

juiste en kwaadwillende content, zoals deepfake video's. In het tweede deel van dit proefschrift dragen wij bij aan het nog nauwkeuriger schatten van de data distributie, met generatieve modellen. Daarbij verbeteren wij de prestatie van een generatief model dat bekendstaat onder de naam Normalizing Flow. Dit werk is gebaseerd op het feit dat DenseNets, meestal, ResNets overtreffen in classificatie opdrachten. Omdat wij te maken hebben met een speciaal type normalizing flow, zijn wij verplicht om de aaneenschakelingen van DenseNet blokken zó samen te stellen dat er wordt voldaan aan de Lipschitz constante. Daarnaast stellen wij een activatiefunctie voor, genaamd: Concatenated LipSwish, welke net zoals de aaneenschakelingen, voldoet aan de Lipschitz conditie. Ook laten wij zien hoe een leerbaar en gewogen aaneenschakeling voor de DensNet blokken de model prestaties verbeteren. De nieuwe architectuur noemen wij i-DenseNet. i-DenseNet presteert beter op het generatief en hybride modelleren, dan zijn voorganger: Residual Flow en andere soortgelijke flow-gebaseerde modellen.

Tot slot wordt er in het derde deel gekeken naar het volledige proces van het comprimeren van foto's en video's met deep learning. Met de wereldwijd toenemende hoeveelheid data, maakt het comprimeren van data een fundamenteel deel uit voor data transmissie en opslag. Neuraal comprimeren heeft als doel digitale content zoals, multimedia streaming, internet voorzieningen, en data opslag te optimaliseren. Hoewel neuraal comprimeren aanzienlijk bijdraagt aan de maatschappij, zijn er ook problemen die zich voordoen. Zo hebben de grootschalige modellen hoge kwaliteit data nodig en zijn zij moeilijk om efficiënt te ontwikkelen op (mobiele) apparaten. Daarnaast vereisen zij een aanzienlijke hoeveelheid computerkracht. Het laatste deel van dit proefschrift onderzoekt het neuraal comprimeren domein en bestaat uit twee hoofdstukken.

NL

In het eerste hoofdstuk van het derde deel onderzoeken wij een neuraal foto comprimerend model, gebaseerd op de zogenaamde mean-scale hyperprior. Dit is een succesvol model, dat beter presteert dan klassieke modellen in het foto comprimerend veld. Ondanks dat deze modellen effectief zijn in de praktijk, hebben de modellen een beperkt vermogen als het aankomt op optimaliseren en generalisatie. Om deze reden introduceren wij een nieuwe methode. Deze methode draagt bij aan het verbeteren van de compressie prestatie, wat resulteert in een verbeterd comprimerend vermogen per foto. In foto comprimeren focussen wij op het optimaliseren van de latente variabele die uit een voor getraind mean-scale hyperprior model komen. Het voor getrainde model heeft gewichten die vast staan en al geleerd zijn. Daardoor hoeft alleen de latente variabele verder te worden geoptimaliseerd, met een verfijnings methode. Wij introduceren drie verschillende methodes, die voortbouwen op de bestaande methode: stochastische Gumbel annealing. Wij laten zien hoe deze uitgebreid kan worden naar het afronden van niet twee, maar drie klassen. Verder laten wij zien hoe de methodes, vanaf nu bekend als SGA+, de basismodellen verbeteren op twee verschillende datasets. Wij laten ook zien hoe SGA+ gebruikt kan worden om over de bitrate-distortie lijn kan bewegen en hoe een van de methodes robuust is tegen verschillende settings van hyperparameter waardes.

Recent onderzoek heeft neuraal foto comprimerende modellen ontwikkeld die in staat zijn om meer bits toewijzen aan vooraf gespecificeerde interesse regio's. Hoewel deze modellen voor foto's goed werken, was er tot nu toe geen neuraal video comprimerend model op de markt die hiertoe in staat was. In het laatste hoofdstuk van dit proefschrift laten wij zien hoe een video compressie model, gebaseerd op de scale-space flow, in staat is om meer bits toe te kennen aan vooraf gespecificeerde interessegebieden. Wij introduceren twee versies die hiertoe in staat zijn, een impliciet model en een latent-schalings model. Over het algemeen presteren beide modellen beter dan de baselines, in termen van de bitrate-distortie in de vooraf gespecificeerde regio's, en generaliseren zij naar een andere dataset gedurende inference tijd. Het latente schaalbaarheid model behaald de aller beste performance en heeft het de optie om met een model, expliciet de kwantisatie klassenbreedte te controleren gedurende evaluatie. Als laatst vinden wij dat er nauwelijks verschil in resultaten zichtbaar zijn wanneer de modellen getrained worden met synthetische gegenereerde frames, welke niet gecorreleerd zijn aan de video frames, in vergelijking met het trainen van pixel genoteerde video frames.