FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Improving prediction of storm structure by spectral methods
Student:	Adam Barla
Supervisor:	Mgr. Petr Šimánek
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2023/2024

Instructions

Deep learning methods are very successful in the short-term prediction of rain and storms. The current methods are very good at predicting the motion of the storms but usually, the predicted storm lacks the internal structure of a realistic storm. This is problematic for meteorologists and their approach to analyzing the weather. This issue can be attributed to spectral bias [1]. We want to improve current methods (like U-Net) by Guided Upsampling as described in [2] or by FREA-Unet [3].

1) Survey current literature on spectral bias and weather nowcasting.
2) Understand and describe the dataset provided by Meteopress.
3) Understand and describe one method to improve UNet (Guided Upsampling or FREA-UNet).
4) Choose one method, and implement it in PyTorch.
5) Train standard Unet and improved Unet on the provided dataset.
6) Compare, analyze and describe the results in detail. The important metrics are MSE, MAE and SSIM.

[1] On the Spectral Bias of Neural Networks, Rahaman, et al.
[2] Spectrally consistent UNet for high fidelity image transformations, Mernerides, et al.
[3] FREA-Unet: Frequency-aware U-net for Modality Transfer, Emami, et al.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 31 October 2022 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Improving prediction of storm structure by spectral methods

Adam Barla

Department of Applied Mathematics
Supervisor: Mgr. Petr Šimánek

May 12, 2023

Acknowledgements

I thank Mgr. Petr Šimánek for guiding me, offering insightful suggestions, and providing consultation during this project. Additionally, I extend my appreciation to all those who devoted their time to proofreading my thesis, including my supportive parents and my friend Aleš Sršen.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 12, 2023 ……

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Barla, Adam. *Improving prediction of storm structure by spectral methods*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023. Also available from: ⟨`https://github.com/barlaada/GUNet-nowcasting`⟩.

Abstract

Short-term weather forecasting (nowcasting) is crucial in predicting extreme weather events. This thesis focuses on structural biases that can be introduced into convolutional neural network predictions.

I investigated whether guided upsampling, used instead of transposed convolution, can improve the accuracy and reliability of weather forecasts. To this end, I trained UNet and Guided UNet (GUNet) models on radar images from a network of meteorological radars created by the OPERA radar program. I analyzed both models using performance indicators such as mean squared error (MSE), mean absolute error (MAE), and the structural similarity index (SSIM).

The results showed that the GUNet model slightly outperformed the UNet model regarding average MAE and MSE and demonstrated a better ability to capture higher frequencies in the Fourier spectrum of radar images. Moreover, the GUNet model achieved marginally better results on images with higher radar echo intensity, essential for predicting severe weather events.

The study suggests that the GUNet model can improve short-term weather predictions, and the results provide a basis for further research in this area.

Keywords weather nowcasting, convolutional neural networks, transposed convolution, UNet, guided UNet, guided filter, guided upsampling

Abstrakt

Krátkodobá predpoveď počasia (nowcasting) zohráva kľúčovú úlohu pri predvídaní extrémnych výkyvov počasia. V tejto práci sa zaoberám chybami, ktoré môžu byť vnesené do predikcí konvolučných neurónových sietí.

V mojom výskume som skúmal, či aplikácia usmerneného prevzorkovania (Guided Upsampling) namiesto transponovanej konvolúcie môže zlepšiť presnosť a spoľahlivosť predpovedí počasia. Na tento účel som vytrénoval modely UNet a GUNet (Guided UNet) na radarových snímkoch zo siete meteorologických radarov vytvorených radarovým programom OPERA. Na analýzu oboch modelov som použili ukazovatele výkonnosti, ako sú stredná kvadratická chyba (MSE), stredná absolútna chyba (MAE) a index štrukturálnej podobnosti (SSIM).

Výsledky ukázali, že model GUNet mierne prekonal UNet z hľadiska priemernej strednej absolútnej chyby a strednej kvadratickej chyby. Navyše preukázal lepšiu schopnosť zachytiť vyššie frekvencie vo Fourierovom spektre radarových snímkov. Okrem toho GUNet dosahoval sčasti lepšie výsledky na snímkach s vyššou intenzitou radarového echa, čo je významné pre predpovedanie závažných poveternostných udalostí.

Na základe týchto výsledkov možno konštatovať, že model GUNet má potenciál na zlepšenie krátkodobých predpovedí počasia. Výsledky mojej práce poskytujú priestor pre ďalší výskum v tejto oblasti.

Kľúčové slová krátkodobá predpoveď počasia, konvolučné neurónové siete, transponovaná konvolúcia, UNet, Guided UNet, guided filter, guided upsampling

Contents

List of Figures

Introduction

Weather forecasting has long been a critical aspect of human life, with accurate predictions allowing us to plan and protect ourselves from the potential impact of severe weather events. While long-term weather predictions offer valuable insights into general trends and patterns, they need more precision to address the immediate threats of severe storms and other fast-changing conditions.

Weather nowcasting is the practice of making short-term weather predictions on the scale of minutes to a few hours. It is a vital tool in our efforts to anticipate and respond to these rapidly evolving weather events. It is beneficial for anticipating the evolution of rapidly changing weather phenomena, like storms, thunderstorms, or heavy rainfall, which can significantly impact public safety, infrastructure, and our daily activities.

In recent years, deep learning techniques like Convolutional Neural Networks (CNNs) have significantly advanced the field of weather nowcasting, especially in predicting storm evolution using radar images. However, despite these advancements, limitations in the quality and accuracy of these predictions persist, which is the focus of my thesis.

The UNet architecture, a popular CNN model for image processing tasks, has shown its potential in various applications, including weather nowcasting. Nonetheless, it is known to suffer from structural bias, which can introduce unwanted artifacts and reduce the quality of the predicted storm structures. To enhance the reliability and accuracy of storm structure predictions, I aim to address these issues and develop a more spectrally consistent model.

Inspired by the work of Demetris Marnerides et al. [1], which proposed the Guided UNet (GUNet) architecture for high-fidelity image transformations, this thesis aims to adapt and apply the concept of spectrally consistent models to the problem of weather nowcasting.

The main objective is to investigate and remove the structural biases present in deep learning models, such as UNet, to improve the prediction of storm structures from radar images. By leveraging spectral analysis methods, I aim

to identify and mitigate the adverse effects of artifacts on the predictions and examine the improvement in the prediction quality.

Initially, the thesis provides a comprehensive literature review covering weather nowcasting, convolutional neural networks, the UNet architecture, and its improvement using Guided Upsampling (GU). Then, the methodology chapter provides an introduction to the dataset of radar images produced by OPERA, along with details about the preprocessing and data splitting techniques used on it. The chapter continues with implementation details of the UNet and the GUNet architectures and the model training process.

The evaluation and results chapter follows, comparing the standard and improved UNet models using metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and Structural Similarity Index Measure (SSIM) while also analyzing the impact of GU on the spectral bias. The discussion delves into the interpretation of results, limitations, potential improvements, and the implications for weather nowcasting. Finally, the thesis concludes by summarizing the findings, highlighting contributions to the field, and outlining possible future research directions.

In summary, my thesis strives to advance the field of weather nowcasting by addressing the structural biases present in deep learning models, thereby improving the prediction of storm structures and contributing to the ongoing efforts to develop more accurate and reliable weather forecasts.

Thesis's Objective

The primary objective of this thesis is to enhance the accuracy and reliability of short-term weather predictions, notably storm structure predictions from radar images, by addressing the spectral biases present in deep learning models like the UNet architecture. By adapting and applying the GU technique from the GUNet architecture to weather nowcasting, the thesis aims to develop a more spectrally consistent model, effectively mitigating the adverse effects of artifacts on the predictions and contributing to the ongoing efforts to develop more accurate and reliable weather forecasts.

In the first part of the thesis, I aim to introduce the reader to CNNs and their application in weather nowcasting. I will describe the UNet architecture, present the concept of structural bias, and discuss how it may arise and affect the network's performance. Furthermore, I will present the GU technique and its use in the GUNet architecture. I will also provide an overview of the dataset used for training the Neural Networks (NNs), characterizing its features and relevance to weather predictions.

The second part of the thesis focuses on creating a more spectrally consistent model for weather predictions using the provided dataset of radar images, which covered the area above the Czech Republic. I intend to design and implement two models, GUNet and UNet, in PyTorch and analyze the impact of GU on spectral bias by comparing their performance using metrics such as MSE, MAE, and SSIM.

Chapter 1

Literature Review

1.1 Weather Nowcasting

When asked to predict future rainfall, meteorologists often get it wrong. At first glance, forecasting rainfall should be manageable as it is one of the main focuses of weather predictions, but the opposite turns out to be the case. Rain is tough to forecast on a small scale.

Rainfall forms in two main ways. The first one happens when two fronts of different temperatures meet. Because of that, it is called frontal rain. The warmer air mass is lifted above, the colder as it is less dense. It then cools at a higher altitude, which causes the water trapped in the previously warmer air to condense due to a drop in the temperature. Meteorologists can predict this type of rainfall by studying these air masses using available weather instruments. Frontal rain often lasts longer and amounts to 40 percent of all rainfall.

The other 60 percent comes from a process known as convection. This type of rain happens when the ground warmed by the sun transfers heat to the nearby air. Warming can cause the air to rise to the higher layers of the atmosphere, where the water vapor condenses. Convective rainfall sounds very similar to frontal rain. Predicting precipitation accurately can be challenging due to several factors. These include determining the ground's temperature, whether the heat is enough to make the air rise, and how long the air will stay above the warm ground to absorb the heat. Knowing the amount of water vapor in the air mass is also crucial. The areas this type of rain covers are minor and likely undetectable by methods used for frontal rain predictions. Convective rain is also shorter and, therefore, harder to predict.

Weather nowcasting aims to address predictions in this shorter horizon. Keith Browning originally defined it as *"the description of the current state of the weather in detail and the prediction of changes that can be expected on a timescale of a few hours"*[2]. Later, the World Meteorological Organization (WMO) defined *nowcasting* as forecasting with local detail, by any method, over a period from

the present to 6 hours ahead, including a detailed description of the current weather.

1.1.1 Radars

Nowcasting is highly dependent on observational data. While surface and upper-air observations are essential, only remote sensing systems can adequately provide high-resolution spatial coverage. A weather radar is the most crucial instrument for nowcasting, particularly for severe local storms associated with thunder, lightning, heavy rain, hail, strong winds, and sudden temperature changes.

Radar has an advantage over all other observing systems in weather forcasting because it directly observes precipitation particles in three dimensions over a large area with an update rate of a few minutes. At radar ranges of less than 60 kilometers, the resolution of the precipitation is better than 1 kilometer squared. Radar makes it possible to estimate rainfall rates and amounts, observe the 3D structure of a storm, which has proven useful in estimating storm severity, and obtain the movement of storms, which is central to nowcasting. With the addition of Doppler capability[1], it is possible to estimate the wind direction and speed. The further addition of dual-polarization[2] enables differentiation of the precipitation particle type, such as rain, snow, or hail, and to identify non-precipitation echoes, such as insects and ground clutter. Dual polarization is particularly useful for data quality control. [3]

The radar collects the measurements in reflectivity, a measure of the power returned to the radar, from atmospheric targets, such as raindrops, snowflakes, or hailstones. The radar reflectivity is measured in dBZ and is represented on a logarithmic scale in decibels (dB) to accommodate the observations in a wide range of signal strengths. The unit dBZ expresses radar reflectivity relative to a reference value (Z), which is the radar reflectivity of a 1 mm diameter droplet of water at a standard distance from the radar. The higher the radar reflectivity, the more intense the precipitation is likely to be.

Radar attributes, such as radar wavelength, Doppler capabilities, dual-polarization, sensitivity, and scanning capabilities, affect the radar's abilities. The primary one is the radar wavelength. There are three wavelengths: S-band (10 cm), C-band (5 cm), and X-band (3 cm). For example, if the primary use of the radar is for estimating heavy rainfall over large regions and to warn of high-wind events from thunderstorms, then an S-band, Doppler, dual-polarization radar is the most suitable. However, if the primary use of the radar is for nowcasting snow, then C-band or even X-band may be suitable. [3]

[1] Doppler shift of the radar waves allows the radar to determine the speed and direction (only toward or away from the radar) of precipitation particles.

[2] Transmitting and receiving two differently polarized waveforms.

1.2 Convolutional Neural Networks

NNs are computational models inspired by biological neural networks. Their goal is to minimize errors between outputs and given target values computed by loss function, e.g., MSE. The function is minimized by adjusting the weights inside the network based on these errors using error backpropagation and gradient descent algorithm[3]. After a sufficient number of successive adjustments, NN produces an output similar to the target output, and the training can be terminated based on specific criteria. NNs can theoretically learn to represent and model any given continuous function, no matter how complex, as long as the network has an appropriate architecture and the function is well-defined within a particular domain [4].

CNNs first appeared in the 90s, but their widespread use can be attributed to breakthroughs in the ImageNet image classification challenge [6]. CNNs have enormously impacted the world of image processing, making them a worthy candidate for predicting the weather phenomena captured in radar images. CNNs leverage ideas, such as sparse interactions, parameter sharing, and equivariant representations, to improve machine learning systems. To understand the benefits convolutional CNNs bring, I need to introduce a mathematical operation that makes CNN a CNN.

1.2.1 Convolution

Under the hood CNNs use, as the name implies, an operation called convolution. Bread and butter of NNs are affine transformations. Vector of inputs is received, then multiplied by some transformation matrix[5]. These operations can be applied to various input types, such as images or sounds. Every input can be represented as a multi-dimensional array, which can then be flattened to apply the transformation. There often are dimensions along which ordering of the data matters[6]. Similarly, the data often has something we can call "channel axis", which represents different views of the data [7]. Flattening the data does not consider these relationships, which may be important in solving tasks like computer vision and speech recognition. [7, pg. 6-8]

Discrete convolution is the answer to this concern. An excellent way to imagine the convolution operation is through a window called the kernel, sliding across the input. The input values are multiplied by the corresponding value in the overlapping window and summed up to obtain the convolution output in the window's current location. The window is then moved to the

[3] I will not go into the inner workings of this algorithm, as it is not the main focus of this thesis. I assume a basic understanding of forward and backward passes from the reader in feed-forward NNs. See paper [4] in which was the backpropagation first proposed.

[4] See paper [5]

[5] A bias vector is added before passing the result to non-linear function

[6] Width and height of images or time axis in sound

[7] Different color channels in images or audio channels in sound

next position. Output for all valid kernel positions[8] is called output feature map. This whole procedure can be repeated with many more kernels to form as many output feature maps as desired. In figure 1.1, we see an example of convolution with 4×4 input and 3×3 kernel. From now on, I will focus on 2D convolution, in which the input and the kernel are two-dimensional.

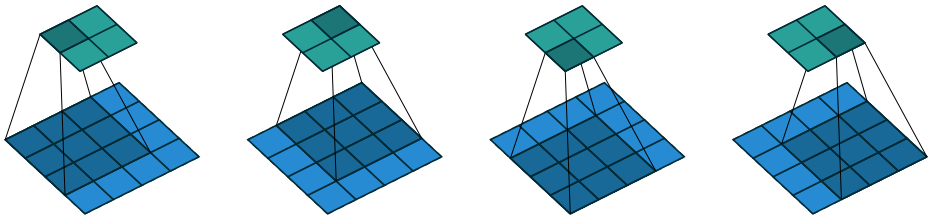


Figure 1.1: Convolving a 3×3 kernel over a 4×4 input. [7]

There can be several feature maps in the input, e.g., representing different image color channels. In that case, the kernel can have multiple dimensions. Using a multi-dimensional kernel is equivalent, to creating the output feature map, by using multiple kernels and summing the resulting unique feature maps element by element. The usage of a collection of kernels to both create multiple output feature maps and convolve over input with multiple channels is illustrated in figure 1.2.

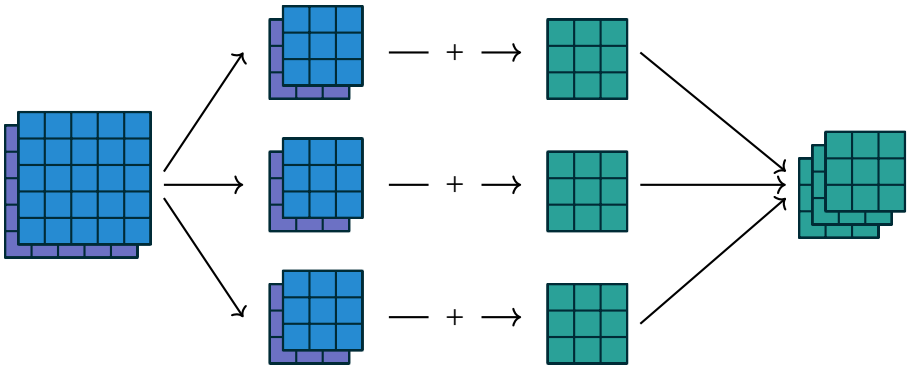


Figure 1.2: A convolution mapping from two input feature maps to three output feature maps using three 3×3 kernel pairs $\mathbf{w}$. In the top pathway, input feature map 1 is convolved with kernel $\mathbf{w}_{1,1}$ and input feature map 2 is convolved with kernel $\mathbf{w}_{1,2}$, and the results are summed together elementwise to form the first output feature map. The same is repeated for the middle and bottom pathways to form the second and third feature maps, and all three output feature maps are grouped to form the output.[7]

[8] Position of a kernel on top of input is valid when an input value exists for every kernel value.

Convolution has two parameters: stride and padding. They, among other things, change the size of the output. Strides represent the size of the step that the kernel window takes when moving to the next position[9]. Padding is the "thickness" of a border of zeros that are added around the input[10]. These parameters can be seen at work in figure 1.3.

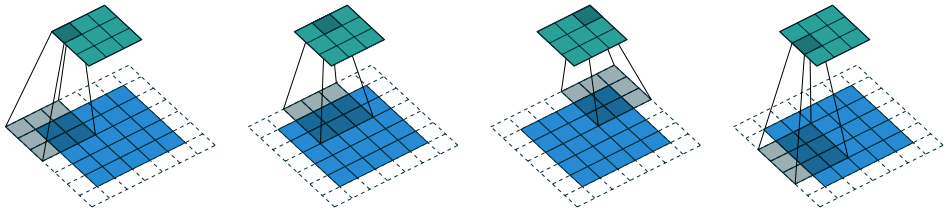


Figure 1.3: Convolving a 3×3 kernel over a 5×5 input padded with a 1×1 border of zeros using 2×2 strides. [7]

Convolution can also be represented as a multiplication of the input by matrix **C**. An example of such matrix for the same case as in figure 1.1 can be seen in figure 1.4.

$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

Figure 1.4: Convolution with 4×4 input and 3×3 kernel represented as matrix multiplication. Inputs and outputs are concatenated.

We can see that multiplying the input by this matrix gives us the same result as sliding the kernel window over the input. The backward pass is easily obtained from this representation by transposing **C**. The error is backpropagated by multiplying the loss with $\mathbf{C}^T$.

If we look at the matrix, we can see the properties of convolution mentioned above. This representation is closer to the actual implementation used in NNs, as it employs matrix multiplication [11]. The matrix is sparse. Therefore fewer parameters need to be stored, reducing memory requirements. It also means we can compute the convolution result faster than in the fully connected case. These efficiency improvements are pretty significant. Most algorithms used in practice run in $O(mn)$ time where m and n are input and output sizes, respectively. For kernel with k parameters, convolution can be run in $O(kn)$ time. Speed improvement is evident when we realize that k can often be orders of magnitude smaller than m while obtaining a good performance.

[9] Size of this step can be different in each dimension of the input.
[10] Padding can also be defined to be different in each dimension of the input.
[11] Software implementations will typically not perform the useless zero multiplications.

Parameter sharing can also be seen in the matrix. Kernel weights are reused multiple times, unlike in traditional NN, where each element of the weight matrix is used only once. Parameter sharing does not affect the forward propagation's runtime but further reduces the model's storage requirements.

Equvariance to translation is another property of CNNs. Equivarience of function $f(x)$ to function g means that $f(g(x)) = g(f(x))$. In other words, equivariance to translation means that if I shifted the input by some number of pixels, I would get the same output but also shifted by the same amount. It is perhaps harder to see from the examples, but it is nonetheless useful. [8, pg. 329-335]

CNN is defined as a NN that has at least one convolutional layer.

1.2.2 Transposed Convolution

Transposed convolutions are quite useful, especially in generative models. The need arises from the desire to use a transformation that goes in the opposite direction of normal convolution. It is easy to assume that transposed convolution is the "opposite" operation of a regular convolution without giving it much thought. Its purpose is to go from something with the shape of some convolution output to something with the shape of its input. This operation should be done while maintaining a consistent connectivity pattern with said convolution. It is important to remember that there does not exist a reverse operation for convolution[12]. [7, 9]

Every convolution boils down to an efficient implementation of a matrix operation. A transposed convolution works by swapping forward and backward passes of a convolution. One kernel defines matrix $\mathbf{C}$, an example of which can be seen in figure 1.4, and its transposition $\mathbf{C}^T$. The matrix used for a forward and backward pass determines the convolution type. In the transposed convolution, $\mathbf{C}^T$ is used for the forward pass instead of $\mathbf{C}$ in standard convolution.

It is always possible to emulate a transposed convolution with a direct convolution. The disadvantage is that it usually involves adding many columns and rows of zeros to the input, resulting in a much less efficient implementation. It is a more useful representation of transposed convolution. It is necessary to zero-pad the input if the same connectivity pattern, in the equivalent convolution to the transposed one, is to be maintained. It must be done so that the kernel's first (top-left) application only touches the top-left pixel.

Suppose we now imagine a convolution with non-unit strides and an "opposite" transposed convolution to it. In that case, the strides in the transposed convolution need to be fractional, as we are now increasing the output's

[12] Sometimes the term "deconvolution" is used instead of transposed convolution, which may be misleading.

size. Hence, we need to take steps smaller than one[13]. Fractional steps can be achieved in the representation of transposed convolution by the standard one by inserting zeros between input units. Inserting zeros between rows and columns of the input makes the kernel move around at a "slower pace" than with unit strides[14]. Figure 1.5 illustrates the fractional strides. Transposed convolution in figure 1.5 is "opposite" of the convolution in figure 1.3. [7] [15]

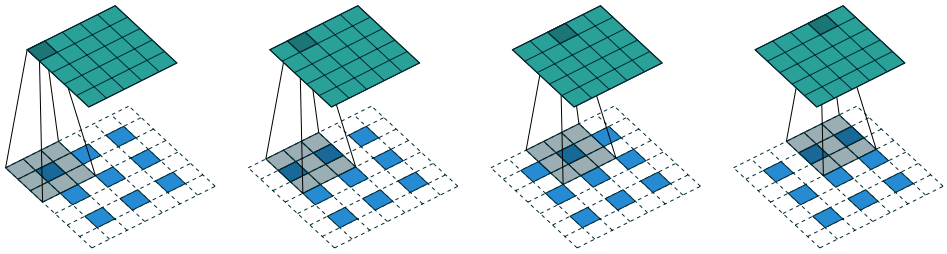


Figure 1.5: Transposed convolution to convolving a 3×3 kernel over a 5×5 input padded with a 1×1 border of zeros using 2×2 strides (figure 1.3). It is equivalent to convolving a 3×3 kernel over a 3×3 input (with one zero inserted between inputs) padded with a 1×1 border of zeros using unit strides. [7]

1.2.3 Pooling

Pooling operations reduce the size of the feature maps by using some function to summarize subregions of the input, such as taking the average or the maximum value. Pooling works like discrete convolution as it slides a window across the input, but it uses a different function instead of the linear combination used in convolution. In addition to discrete convolutions, the pooling operation makes up another vital building block in CNNs.

1.2.4 Validation Metrics

Different evaluation metrics are used for assessing the performance of CNNs, particularly in image processing and computer vision tasks. Here is a brief overview of metrics that I will use later in the thesis.

- SSIM is a perceptual metric that quantifies the similarity between two images. It considers the images' luminance, structure information, and contrast, which are more aligned with human visual perception. SSIM values range from -1 to 1, with 1 indicating a perfect match between the

[13] This is why transposed convolution is sometimes also called fractionally-strided convolution.

[14] The padding has to be equal to the size of the kernel minus one.

[15] For more visualizations and a deeper explanation of how different convolution parameters, such as padding and strides, influence the output size, you can refer to [7].

two images. In the context of NNs, SSIM can be used to evaluate the quality of generated or reconstructed images, such as in image denoising, super-resolution, or image-to-image translation tasks.

- MAE is a simple and easy-to-interpret metric that measures the average absolute difference between the predicted and actual values. It is used for both regression and image-processing tasks. For image processing, MAE calculates the average absolute difference between the pixel values of two images. A lower MAE indicates better model performance, which signifies fewer predicted and actual value discrepancies.

- MSE is another metric used to measure the difference between predicted and actual values. It calculates the average squared difference between the two sets of values. In image processing, MSE measures the average squared difference between the pixel values of two images. Similar to MAE, a lower MSE value indicates better performance. MSE tends to penalize more significant errors more severely than MAE, as the squared term magnifies the differences.

Figure 1.6 captures differences between each metric. These metrics are used to evaluate and compare the performance of different neural network models or configurations in various tasks, such as image synthesis, denoising, segmentation, and more. Models can be fine-tuned to achieve better results by analyzing the performance using these metrics.

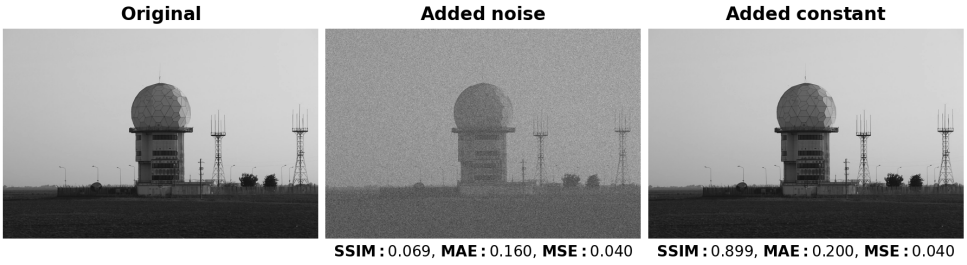
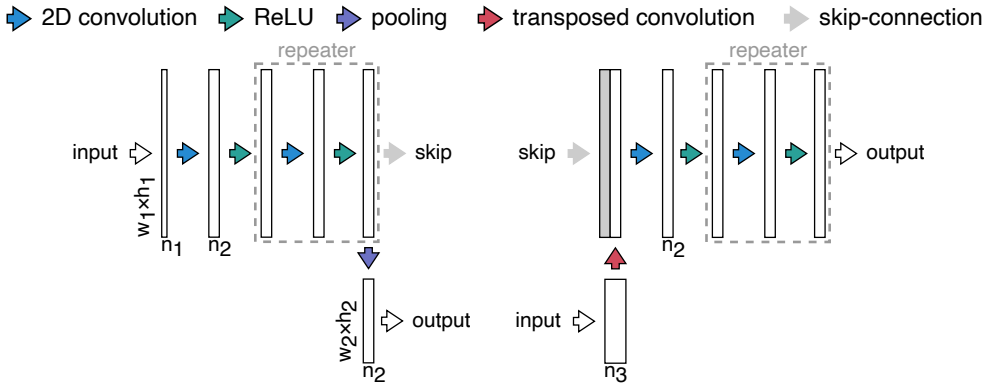


Figure 1.6: Differences between evaluation metrics. Modified images have the same MSE, but one was created from the original by adding a random noise and the other by adding a constant. Image used for the comparison: [10].

1.3 UNet

UNet is a CNN that was first proposed for biomedical image segmentation in the paper [11]. The architecture of this network consists of there main parts, encoder layers, bottleneck, and decoder layers. Encoders are typical convolu-

tion layers that consist of convolution stage, nonlinearity, and pooling stage[16]. The convolution and nonlinearity stages can be repeated multiple times in one layer. A diagram showing this layer can be seen in figure 1.7a. The pooling stage reduces the resolution of the images. The convolution inside the encoder is also used for increasing the number of features.



(a) Encoder (b) Decoder

Figure 1.7: Types of layers in the UNet architecture

Figure 1.7b illustrates that the decoder layer starts with upsampling the output of the previous decoder layer or bottleneck. Upsampling is done using a transposed convolution, which increases the resolution and decreases the number of features. The bottleneck is practically the same as the encoder layer but without the pooling stage at the end.

Each encoder layer in the architecture acts as a low-pass filter, suppressing details and higher spatial frequencies in the input. For the higher frequencies to be transmitted to the output, the encoder must acquire the ability to encode them in the intermediate features of the network. [1]

The original paper [11] introduced skip-connections in order to achieve this goal. These concatenate the encoder features before the pooling stage with the upsampled features in the decoder. Concatenation is done at each "level" of the network, effectively bypassing the network's lower levels. These "fast-forward" connections, as they are sometimes called, allow for the details from the input to be transferred directly to the output without passing them through a bottleneck. Removing the need to encode the higher spatial frequencies allows, the lower levels to better capture global features. Skip connections can be seen in figures 1.7 and 1.8.

Similarly to the encoder, convolution with activation stages can be repeated multiple times in the decoder.

[16] There are two conventions when it comes to the term "layer" in CNNs. In one, the convolution, nonlinearity, and pooling are all called layers, and in the other layer is a term for all of them together. I will use the latter one.

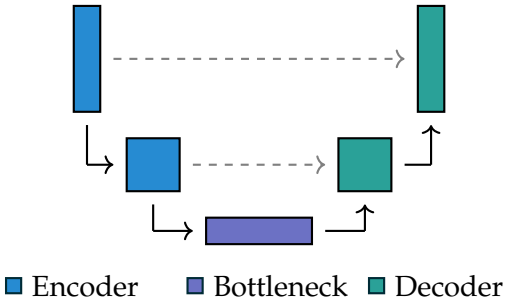


Figure 1.8: UNet architecture

The architecture can be designed for tiling to take advantage of the properties of NNs that I mentioned, particularly the equivariance to translation. Tiling means that the NN output can be extended without increasing the network size by running the net on different "tiles" of the image separately to produce output in a mosaic. For this to work near the edges of the tiles, the convolutional stages cannot have padding, which means the output of the NN will be smaller than the input. An example of this can be seen in paper [11, fig. 2].

1.4 Bias in Neural Networkts

Bias in machine learning, also sometimes called algorithm bias or AI bias, is a phenomenon that occurs when an algorithm produces systemically prejudiced results. It is separate from the bias parameter inside NNs.

1.4.1 Structural Bias

Many CNN architectures employed a similar architecture to the one described in the section 1.3 about UNet. When examining images created by these CNNs, one may observe a distinct checkerboard pattern of irregularities, as shown in figure 1.9. When generating images from NNs, the common practice is to start from lower-resolution images and gradually upscale them, which allows the NNs to describe the rough image and then fill in some details in multiple steps. In order to achieve this, a technique for converting low-resolution images into high-resolution images is required. Transposed convolution is a commonly used method for this task, as explained in section 1.2.2. Layers of transposed convolutions are often found in the decoder of AutoEncoders or the generator part of Generative Adversarial Networks (GANs). The UNet is based on the AutoEncoder architecture and therefore uses transposed convolution.

Unfortunately, transposed convolution can easily lead to artifacts due to uneven overlap. In particular, uneven overlap in transposed convolution hap-

Figure 1.9: These instances show generative models that exhibit checkerboard artifacts in their outputs. The image used is sourced from [9].

pens, when the kernel size is not divisible by the stride. In the section 1.2.2, I talked about fractional strides in transposed convolution and how they can be pictured as padding between input columns and rows. It is evident from the visualization in figure 1.10 that the kernel occasionally overlaps one, two, or four inputs. This behavior forms grid-like artifacts. CNNs generally have multiple layers with transposed convolution, so these artifacts compound on each other. [9]

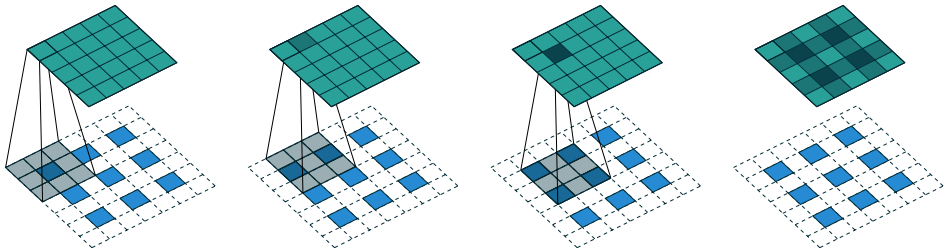


Figure 1.10: Demonstration of artifact formation on the same transposed convolution as in figure 1.5. [7]

The models are learning weights used in convolution and, in theory, could learn to mitigate the effects of uneven overlap. Learning to reduce these artifacts can be challenging, particularly when the convolution is done on multiple channels that impact each other. While possible, it significantly restricts the potential filters and therefore sacrifices the model's capacity. In practice, CNNs struggle with completely avoiding these artifacts. In fact, not only do NNs with uneven overlap not learn to avoid this, but models with even overlap often learn kernels that cause similar artifacts! While it is not their default behavior the way it is for uneven overlap, it is still very easy for transposed

convolution with even overlap to cause artefacts[17]. Completely avoiding artifacts in models with even overlap is still a significant restriction on filters, and in practice, the artifacts are still present in these models, although they seem milder.[9]

While many factors are at play here, the transposed convolution is a big part of the problem. It is fragile because it easily represents artifact-creating functions, even when the size is carefully chosen. At worst, creating artifacts is the default behavior of transposed convolution. In the upcoming section 1.5, I will explain a method to increase the resolution of an image less susceptible to these visual abnormalities.

1.4.2 Spectral Bias

Recent study [12] focused on deep ReLU networks through the lens of Fourier analysis found that while NNs can approximate arbitrary functions[18], they favor low-frequency ones. Therefore they exhibit a bias towards smooth functions. This phenomenon is called spectral bias. I will focus on two experiments in the study mentioned above.

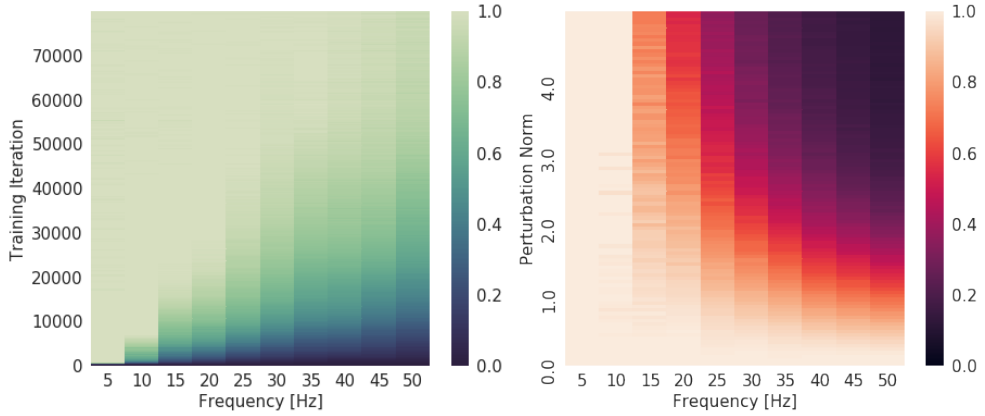
In the first experiment, the researchers trained six layers deep and 256 unit wide fully-connected ReLU network on an output defined by the mapping $\lambda : [0, 1] \to \mathbb{R}$ given by

$$\lambda(z) = \sum_i A_i \sin(2\pi k_i z + \varphi_i), \tag{1.1}$$

where $\kappa = (k_1, k_2, ...)$ are the frequencies with corresponding amplitudes $\alpha = (A_1, A_2, ...)$ and phases $\phi = (\varphi_1, \varphi_2, ...)$. Network was trained to regress λ with $\kappa = (5, 10, ..., 45, 50)$ and $N = 200$ input samples spaced equally over $[0, 1]$. The spectrum of the output was monitored as training progressed. In the first setting, equal amplitudes $A_i = 1$ were set for all frequencies, and in the second setting, the amplitude gradually increased from $A_1 = 0.1$ to $A_{10} = 1$. Figure 1.11a shows that lower frequencies are regressed first, regardless of their amplitudes. [12]

The same setup as the first experiment was used for the second experiment. Once the network had converged on certain parameters θ^*, the authors made random perturbations $\theta = \theta^* + \delta\hat{\theta}$ of a specific size δ, where $\hat{\theta}$ is a random unit vector in the parameter space. They evaluated the network function f_θ at the perturbed parameters and computed the magnitude of its discrete Fourier transform at frequencies k_i to get $|\tilde{f}_\theta(k_i)|$. To obtain $|\tilde{f}_{\mathbb{E}\theta}(k_i)|$, which was then normalized by $|\tilde{f}_{\theta*}(k_i)|$, they averaged the results over 100 samples of $\hat{\theta}$. $|\tilde{f}_{\mathbb{E}\theta}(k_i)|$ was then averaged over the phases ϕ from equation 1.1,. Figure 1.11b displays the experiment's outcome. [12]

[17] Even overlap happens when the kernel size is divisible by the stride.
[18] See paper [5]



(a) Evolution of the spectrum during training. The colors show the measured amplitude of the network spectrum at the corresponding frequency, normalized by the target amplitude at the same frequency. [12]

(b) Spectrum of the model with perturbed parameters as a function of parameter perturbation. Amplitudes are normalized by the amplitudes obtained from the network without the perturbed parameters. [12]

Figure 1.11: Results of experiments from the study [12] showcasing the spectral bias.

The second experiment discovered that higher frequencies are more susceptible to changes in parameters than lower frequencies. This result suggests that precise tuning of the parameters is necessary for higher frequencies to be expressed effectively. In other words, parameters that contribute towards expressing high-frequency components occupy a small volume in the parameter space. [12]

I hypothesize that the use of transposed convolution negatively impacts the learning of higher frequencies. It may not be possible to fine-tune the parameters required for capturing higher frequencies due to the limitations on filters in transposed convolution, as discussed in section 1.2.2. Increasing the likelihood of the model learning higher frequencies can be achieved by removing the structural biases that arise from upscaling using transposed convolution. In the following section 1.5, I will focus on one upsampling method that aims to accomplish this by replacing the transposed convolution with guided upsampling.

1.5 GUNet: Guided UNet

Transposed convolution in the decoder layers of the UNet architecture can cause checkerboard artifacts, as mentioned in section 1.4.1. Other non-learned upsampling methods can cause blurring since they are based on pre-defined interpolation.

Several methods that attempt to alleviate these upsampling artifacts have been proposed. A resize convolution technique is introduced in the research paper [9]. This method utilizes a technique that separates upsampling from convolution to ensure more precise feature computation. Before the convolution process, the image is resized using either nearest-neighbor or bilinear interpolation. Other methods try to reduce the checkerboard artifacts by correlating the kernel weights in transposed convolution. An example of such a method can be found in the paper [13], which proposes a specialized initialization scheme for transposed convolutional layers that correlates the kernel weights at initialization. These methods generally eliminate the artifacts, but they fail to use the high-frequency details from the encoder efficiently and are trying to extract the information in the decoder. Non-learned methods are also prone to blurring. Furthermore, it should be noted that utilizing learned upsampling through transposed convolutions, even when the kernel weights are correlated, does not assure the avoidance of minima that produce artifacts during optimization, as previously discussed in section 1.4.1. [1]

This section describes the GUNet architecture proposed in the paper [1].

1.5.1 Guided Image Filtering

Guided Image Filter (GIF) is an edge-preserving smoothing filter. Its main advantage over bilateral filters is the computational complexity. The output image of the GIF is consistent with the gradient direction of the guidance image, which prevents gradient reversal prominent in the bilateral filters.

GIF works by assuming that there is a local linear model between the guidance I and the filtering output q. The following definition of the filter is taken directly from the paper [14], in which the filter was proposed.

We assume that q is a linear transform of I in a window ω_k centered at the pixel k:

$$q_i = a_k I_i + b_k, \forall i \in \omega_k, \tag{1.2}$$

where a_k and b_k are some linear coefficients assumed to be constant in ω_k. We use a square window of a radius r.
To determine the linear coefficients a_k and b_k, we need constraints from the filtering input p. We model the output q as the input p subtracting some unwanted components n like noise/textures:

$$q_i = p_i - n_i. \tag{1.3}$$

We seek a solution that minimizes the difference between q and p while maintaining the linear model in equation 1.2. Specifically, we minimize the following cost function in the window ω_k:

$$E(a_k, b_k) = \sum_{i \in \omega_k} \left(\left(a_k I_i + b_k - p_i \right)^2 + \epsilon a_k^2 \right). \tag{1.4}$$

Here, ϵ is a regularization parameter penalizing large a_k. Equation 1.4 is the linear ridge regression model, and its solution is given by

$$a_k = \frac{\frac{1}{|\omega|}\sum_{i\in\omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon}, \qquad b_k = \bar{p}_k - a_k \mu_k. \tag{1.5}$$

Here, μ_k and σ_k^2 are the mean and variance of I in ω_k, $|\omega|$ is the number of pixels in ω_k, and $\bar{p}_k = \frac{1}{|\omega|}\sum_{i\in\omega_k} p_i$ is the mean of p in ω_k. Having obtained the linear coefficients a_k and b_k, we can compute the filtering output q_i by equation 1.2. However, a pixel i is involved in all the overlapping windows ω_k that covers i, so the value of q_i in equation 1.2 is not identical when it is computed in different windows. A simple strategy is to average all the possible values of q_i. So after computing a_k and b_k for all windows ω_k in the image, we compute the filtering output by

$$q_i = \frac{1}{|\omega|}\sum_{k|i\in\omega_k} (a_k I_i + b_k). \tag{1.6}$$

Noticing that $\sum_{k|i\in\omega_k} a_k = \sum_{k\in\omega_i} a_k$ due to the symmetry of the box window, we rewrite equation 1.6 by

$$q_i = \bar{a}_i I_i + \bar{b}_i, \tag{1.7}$$

where $\bar{a}_i = \frac{1}{|\omega|}\sum_{k\in\omega_i} a_k$ and $\bar{b}_i = \frac{1}{|\omega|}\sum_{k\in\omega_i} b_k$ are the average coefficients of all windows overlapping i.

The definition of the guided filter is provided by equations 1.5 and 1.7. Local linear model in equation 1.2 ensures that q has an edge only if I has an edge because the gradient of q is just scaled gradient of I ($\nabla q = a\nabla I$). After the modifications in equation 1.7, this no longer holds true because the linear coefficients $\bar{a}_i$ and $\bar{b}_i$ vary spatially. Nevertheless, as $\bar{a}_i$ and $\bar{b}_i$ are the output of a mean filter, their gradients can be expected to be much smaller than that of I near strong edges. In this situation, we can still have $\nabla q \approx \bar{a}\nabla I$, meaning that abrupt intensity changes in I can be mostly preserved in q. [14]

The algorithm constructed from this definition runs in $O(N)$ and can be improved to $O(N/s^2)$, where s is a subsampling ratio. Improvement is achieved using a Fast Guided Filter (FGF) described in paper [15]. In various applications, this leads to a speedup of more than ten times with almost no visible degradation. A pseudocode to both algorithms can be found in the appendix A.

1.5.2 Guided Upsampling

Due to computational and memory costs, it is beneficial to downsample large images when performing image analysis and enhancement tasks such as tone

mapping, colorization, stereo depth rendition, and photomontage. The result is then obtained by upsampling the solution computed on the smaller images. This upsampling does not consider the additional information in the original high-resolution image. Images upsampled by convolving the low-resolution image with an interpolation kernel typically also suffer from a blurring of sharp edges because of the smoothness inherent in the linear interpolation filters. A Joint Bilateral Upsampling (JBU) operation was proposed in the paper [16], which leverages this information and produces outstanding full-resolution results from stereo depth rendition, image colorization, adaptive tone mapping, and graph-cut based image composition computed at very low resolutions.

Just like JBU, GIF can be utilized for upsampling by using an initial image as a basis for the operation. The algorithm differs from the one in section 1.5.1 because we now have a guidance image at a different scale than the filtering input. In this case, we compute the linear coefficient a and b in using the equation 1.5 at the coarse scale, bilinearly upsample them to the fine-scale (replacing the mean filter on a and b) and compute the output by $q = aI + b$ at this scale. The result of GU is visually comparable to the JBU. [14]

GIF provides a better performance than the bilateral filter used in JBU, as I previously mentioned. The upsampling adjustments can also be made on FGF for an even faster algorithm.

1.5.3 Guided UNet Architecture

GIF can be incomported in the UNet architecture discussed in section 1.3. Using GIF, we can benefit from the filter's edge-preserving nature and use the higher-level features from the encoder as guidance in the upsampling. UNet takes advantage of skip connections and combines them with features that pass through the bottleneck by concatenating them and combining them with convolution. GUNet architecture proposed in paper [1] replaces this with GU.

In contrast to the UNet method, two skip connections are used in the upsampling process, which involves two levels of resolution: high and low. We can upscale the low-resolution feature by utilizing information from features in the skip connections of encoders from both levels.

Both of these features are used as guides in the filter. Let's call them $I^{\mathtt{hr}}$ and $I^{\mathtt{lr}}$. Input to the filter is an output of the previous decoder or bottleneck layer[19]. The illustrations of how this works can be found in figures 1.12 and 1.13.

The operations of the GU are modified to accommodate two guides. According to the paper [1], lower-resolution guide $I^{\mathtt{lr}}$ and input p are used to

[19] When upsampling the bottleneck features in the first decoder, there is no skip-connection of lower-resolution we can use. One effective way to address this issue is to utilize the bottleneck as a lower resolution guide *and* as an input for the filter.

compute $\bar{a}_k^{\mathtt{lr}}$ and $\bar{b}_k^{\mathtt{lr}}$ on the lower resolution. The coefficients are then upsampled back to the higher resolution to form $\bar{a}_k^{\mathtt{hr}}$ and $\bar{b}_k^{\mathtt{hr}}$, using bilinear upsampling. The coefficients are then applied on the higher resolution guidance feature $I^{\mathtt{hr}}$ to compute the final filtered decoder feature:

$$q_i = \bar{a}_i^{\mathtt{hr}} I^{\mathtt{hr}} + \bar{b}_i^{\mathtt{hr}}. \tag{1.8}$$

The dotted arrows in figure 1.12 represent the abovementioned operations. Some readers might have noticed the discrepancy in the channel counts of lower coefficients ($\bar{a}_k^{\mathtt{lr}}$, $\bar{b}_k^{\mathtt{lr}}$) and higher coefficients ($\bar{a}_k^{\mathtt{hr}}$, $\bar{b}_k^{\mathtt{hr}}$). The authors of the paper state that the upsampling is done on each channel separately, but this poses a challenge, which I will discuss in the section 2.3.1. One solution might be to use convolution to decrease channel counts of low-resolution coefficients.

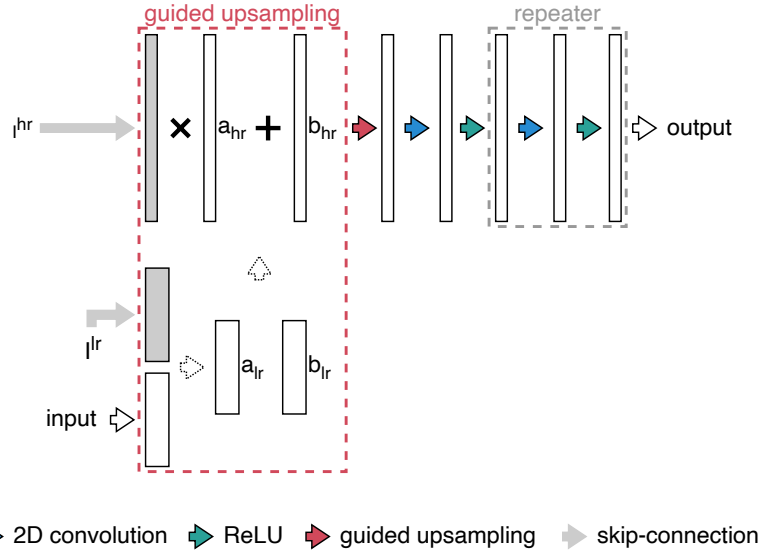


Figure 1.12: Decoder layer in the GUNet architecture

Guided feature upsampling combines the encoder and decoder features of the architecture and aims to guide its features at each upsampling stage in the output to be structurally similar to the corresponding feature set of the input features in the encoder. It is worth pointing out that GUNet results in fewer parameters than UNets with transposed convolutions since the upsampling is parameter-free[20] and the concatenation layer is avoided. The paper [1] later investigates the effects of the structural biases of CNNs on network outputs. The improvement attained by GUNet was evident in the Fourier domain. The effectiveness of this approach was demonstrated in the inverse tone

[20] Even if we use convolution to decrease the channel counts of the low-resolution coefficients $\bar{a}_k^{\mathtt{lr}}$ and $\bar{b}_k^{\mathtt{lr}}$

mapping and colorization of grayscale images. State-of-the-art performance was achieved in the inverse tone mapping application. In colorization, GUNet exhibited benefits compared to alternative UNet architectures. [1]

My goal is to apply this architecture to weather nowcasting and compare it with the traditional UNet both in the Fourier domain and using traditional metrics such as SSIM, MAE, and MSE.

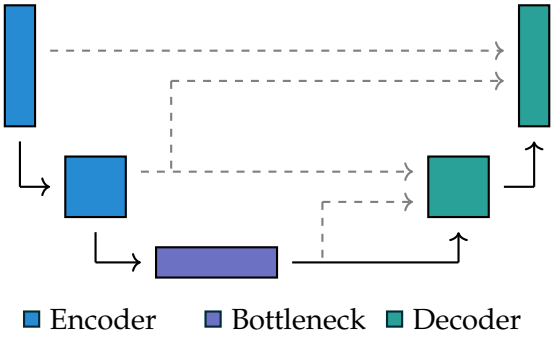


Figure 1.13: GUNet architecture

Chapter 2

Methodology

2.1 Dataset

I used a collection of radar images to train and evaluate the models. These radar images were composites from a heterogeneous network of operational weather radars created by OPERA [17]. In October 2013, the network consisted of 202 operational radars, of which 184 had Doppler capability and 48 were dual-polarization radars. Most radars operate at the C band (168 sites). However, several S-band radars are installed in the south of Europe (33 sites)[21].

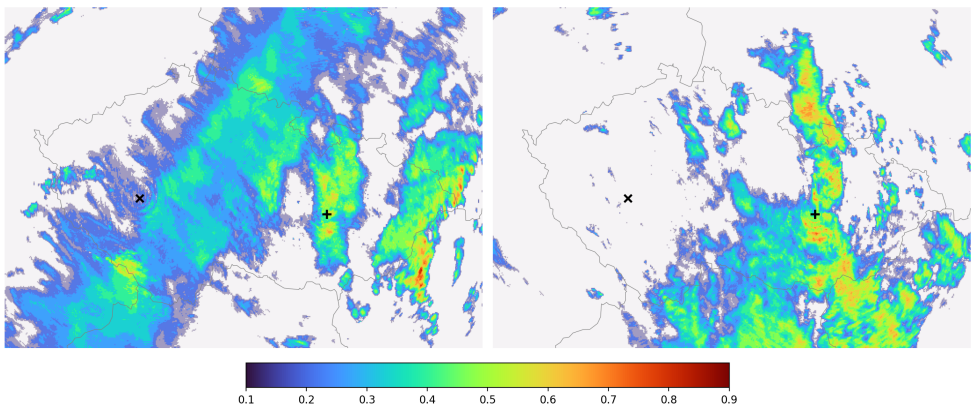


Figure 2.1: Example of images from the dataset after preprocessing. Values smaller than the lower bound of the color bar are transparent, and those higher than the upper bound are displayed as if they were equal to it. Positions of Czech radars are marked with symbols × (Brdy-Praha) and + (Skalky).

The OPERA Radar Data Center (Odyssey) has been operational since 2011.

[21] For more information about different radars used in weather nowcasting, refer to section 1.1.1

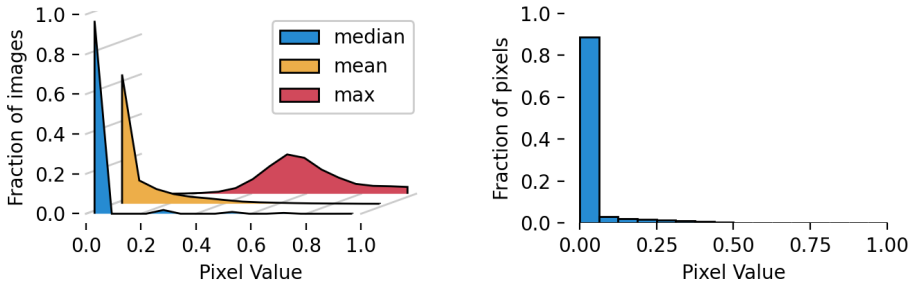
Polar volume data are collected from 134 radar sites in 21 countries, and continental scale mosaic products (surface rain rate, rainfall accumulation, and maximum reflectivity) are generated in real-time. These mosaic products are generated on a Cartesian grid covering Europe (3800×4400 km^2). [17]

The images I used cover the area above the Czech Republic, as illustrated in figure 2.1. The resolution of images is 544×352 pixels, where one pixel represents an area of 1 km^2. Images are generated every 10 minutes, but some may be missing. Absent images were taken into account when creating network inputs and targets. The dataset covers the period from October 23, 2015, to July 21, 2020.

2.1.1 Data Preprocessing and Augmentation

The radar data from OPERA was already cleaned using a combination of anomaly removal and hit-accumulation clutter filtering, which was done by the anomaly-removal module[22].

In the mosaic, each pixel comprises of data from different multiple radars and elevations. The highest value from all the available heights and radars is chosen for each pixel. The dataset's images have a maximum reflectivity captured in 16 values ranging from 0 to 60 dBZ. These values are then mapped to a range of 0 to 255. I utilized min-max normalization to ensure all values ranged from 0 to 1. I then cropped a rectangle of 512×256 pixels from the images, preventing pooling problems[23].



(a) Distribution of maximum, median, and mean values for each image in the dataset.

(b) Distribution of individual pixel values in the dataset.

Figure 2.2: Distribution of radar echo intensities in the dataset.

[22] Some anomalies are still visible in the figure 2.1 in the form of circles around the radars.

[23] Pooling is used in both models with 2×2 stride and 2×2 kernel, which halves the feature resolution in both axes. This pooling can be performed without padding or dropping values when the images have sides equal to powers of two. See section 1.2.1 about convolution and section 1.2.3 about pooling.

I did not exclude any images from the data. In hindsight, it may have been a mistake not to do this because the dataset includes numerous almost empty images, as illustrated in figure 2.2a.

2.1.2 Creating Data Points

Both models were trained on a sequence of consecutive images spaced by parameter `stride`[24]. The dataset was split into chunks, each containing `chunk_size` of images from the sequence. Data points were generated by setting an `input_length`, and `target_length`.

A sliding window of length `input_length+target_length` is then moved along the images in one chunk. Each position of the sliding window represents one data point for the models. The corresponding data point will not be created if the dataset has a missing image in data points inputs or outputs.

The dataset is split by randomly assigning generated chunks to some of the datasets. Two consecutive chunks have no overlap, which is essential for creating training, validation, and testing datasets. The probability of assigning a chunk to one of the datasets is given by parameters `test_frac` and `val_frac` representing fractions of the dataset which will be used for testing and validation respectively[25]. Data points in assigning chunks are then concatenated to create the datasets. The random assigning can be seeded by setting the `seed` parameter. This way, the same datasets are generated each time.

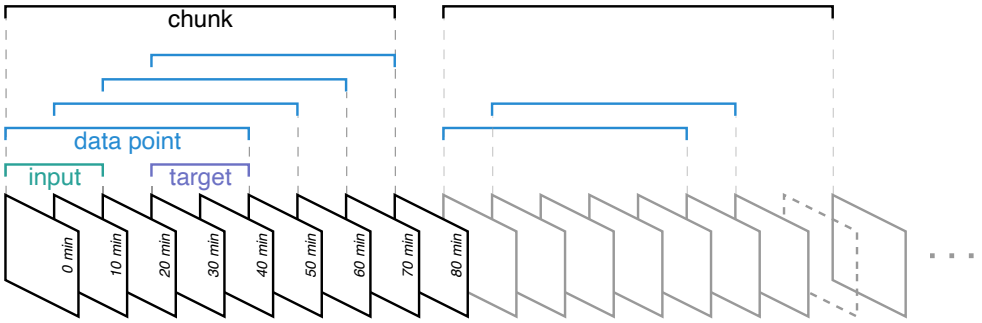


Figure 2.3: Generating data points from the dataset in chunks. Datapoint is added to the chunk only if it has no missing images in the input or target images. There is no overlap of data points between different chunks.

Adjusting the `chunk_size`[26] changes the number of data points overall as smaller chunks skip more images to maintain no overlap between the chunks. I chose these values for the parameters:

[24] Parameter `stride` is in minutes.

[25] The probability that these datasets will contain the same number of images is small, because, aside from the randomness, the chunks do not contain the same number of data points due to the missing images.

[26] Parameter `chunk_size` needs to be ≥ than `input_length` + `target_length`.

`stride` = 10, `test_frac` = 0.1, `val_frac` = 0.1, `seed` = 42,
`chunk_size` = 100, `input_length` = 8, `target_length` = 8.

This resulted in training, validation, and testing datasets with 168110, 21108, and 20957 data points, respectively. The whole process is depicted in figure 2.3.

2.2 UNet

I chose similar architecture as in the original paper [11]. The network has five "levels", meaning five pooling and transposed convolution layers exist. The encoder layer consists of a convolution block and max pooling. As mentioned earlier, max pooling is done with 2×2 strides and 2×2 kernel.

Convolution block is a sequence of 2D convolution followed by Batch normalization, ReLU, and Dropout. The convolution block can be repeated multiple times. I chose to do three repetitions. All 2D convolutions in the block are executed with 1×1 stride and 3×3 kernel and 1×1 padding[27].

Batch normalization is a technique used in deep learning to improve the training process of artificial neural networks[28]. I used it as it has been shown to improve training speed, enable the use of higher learning rates, and often lead to better generalization performance. *Dropout* is a regularization technique used in neural networks to prevent overfitting and improve generalization[29]. During the training process, Dropout randomly "drops out"[30] a proportion of the neurons in a given layer at each training step. The dropped-out neurons do not contribute to that training iteration's forward or backward pass. A hyperparameter called the `dropout_rate` can set the probability of a neuron being dropped.

The number of output channels from the convolution block can be adjusted by setting different values of parameters `in_channels` and `out_channels`. The first convolution in the block changes the number of feature maps as described in the section 1.2.1 and figure 1.2. In the encoder, the number of features is doubled each time.

The decoder layer begins with a transposed convolution. The output of this operation is then combined with a feature from the skip connection. Convolution block follows, but this time the number of features is halved. The transposed convolution also halves the number of features. It has a 2×2 stride

[27] In practice, it would be better to use no padding and therefore have smaller output than input. Doing it this way guarantees the same network output near edges when tiling larger surfaces. See [11, fig. 2]
[28] See paper [18].
[29] See paper [19]
[30] I. e. temporarily removes or sets to 0.

and variable kernel size, which is given by parameter `kernel_size` when the network is initialized. I will examine how this parameter's best value was found later in the section 2.4.1. Padding in transposed convolution must be adjusted based on the kernel size to $\lfloor (k-2)/2 \rfloor$.

Finally, preventing some artifacts which may appear from the transposed convolution is achieved by 2D convolution with 1×1 stride and 1×1 kernel. Moreover, I added a sigmoid activation as the network's last layer to make the network outputs in the correct range. Figure 2.4 shows a diagram of the implementation. The network used for eventual training had 11 713 080 parameters, all trainable. The size of the weights is 44.68 MB.

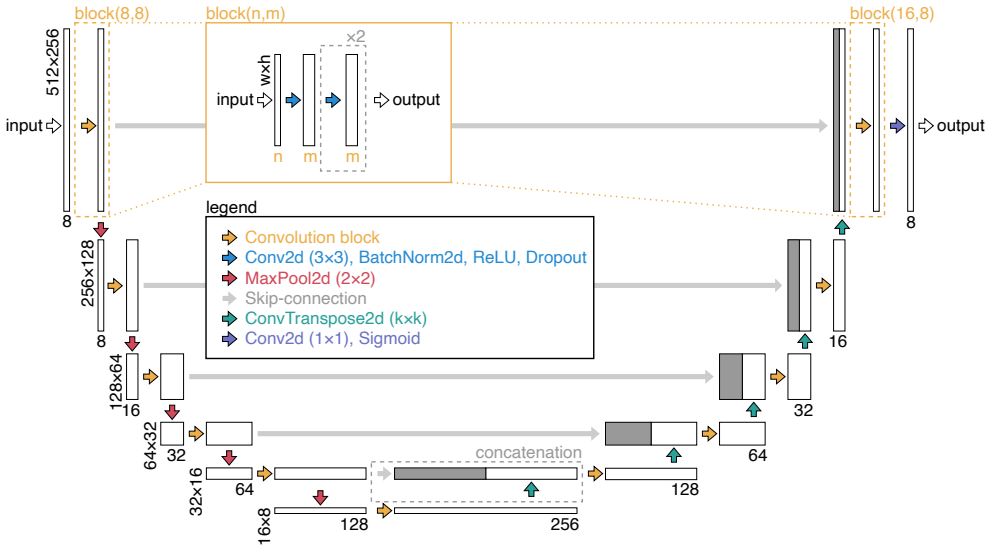


Figure 2.4: UNet implementation diagram.

2.3 GUNet

My implementation of the GUNet architecture is inspired by the one in the paper [1]. It is similar to the UNet, as portrayed in figure 2.5. The convolution block, which follows the same structure as in the UNet architecture, uses 1×1 stride, 3×3 kernel, and 1×1 padding for all 2D convolutions. The network's final layers utilize 2D convolution with a 1×1 stride and 1×1 kernel, along with sigmoid activation.

Upsampling modules inside the decoder are replaced by GU[31]. The GIF in GU has two parameters[32]: ϵ, a regularization parameter penalizing large a_k and r which is a radius defining ω_k. The value of these parameters was found

[31] Workings of GU are described in section 1.5.2
[32] See section 1.5.1

by hyperparameter search, which I explore in the section 2.4.1. Using GIF instead of transposed convolution lowered the number of trainable parameters to 10 578 728, which take up 40.35 MB.

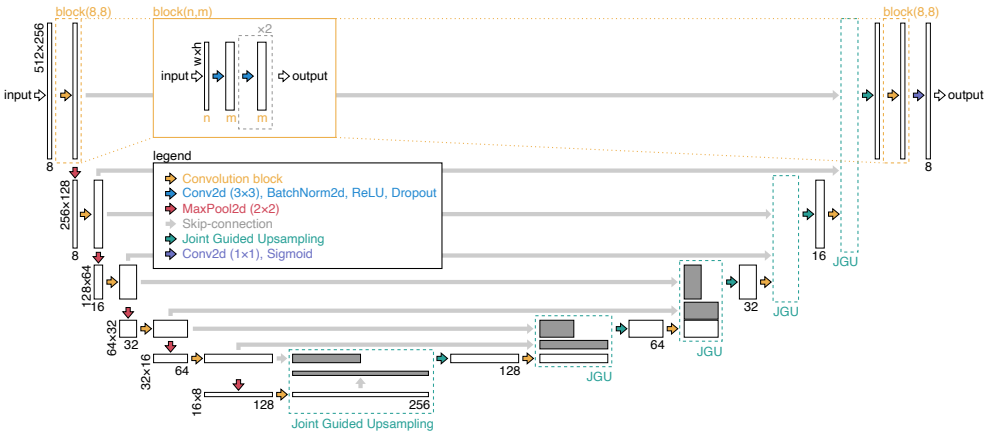


Figure 2.5: GUNet implementation diagram.

2.3.1 Feature Channel Count Discrepancy

The GU was implemented by slightly adjusting the GIF implementation from the paper [20]. In the paper [1], the authors mention that the GIF should be applied on each channel separately. Here I ran into an issue because, as shown in figure 2.5 and discussed previously in the section 1.5.3, the GUNet architecture's lower layers have more channels than the higher ones.

The issue with the feature channels in the "higher" and "lower" skip-connections is that the number of channels in the feature from the "higher" skip-connection used as the higher resolution guide, I^{hr}, is not the same as the number of channels in the feature from the "lower" skip-connection used as the lower resolution guide, I^{lr}. The filter, therefore, cannot be applied on each channel separately, as implied in the paper [1].

I contacted the authors of this paper about this issue and learned that they used an identical channel count on each "level" of their networks, which seemed odd because the diagram in [1, fig. 2] displayed higher channel counts in the "lower" layers of the UNet. After further consultation with my advisor, I decided to reduce the number of channels in the coefficients $\bar{a}_k^{\mathrm{hr}}$ and $\bar{b}_k^{\mathrm{hr}}$, using 2D convolution with 1×1 stride and 1×1 kernel. The coefficients then can be utilized with the higher resolution guidance feature I^{hr} to compute the final filtered decoder feature with equation 1.8.

2.4 Model Optimization

Both networks were implemented in PyTorch and trained on the NVIDIA A100-SXM4-40GB graphics card.

2.4.1 Hyperparameter Tuning

For finding values of tunable hyperparameters of both networks, I used a hyperparameter optimization framework called Optuna, from paper [21]. I used the default Tree-structured Parzen Estimator (TPE) algorithm[33]. I created a study of 50 trials for each model. Each trial could have run for at most five epochs, where one epoch passed over all data points in the training dataset. Batch sizes for both models were set to 70. I used the Adam optimizer [23]. Trials were compared based on average SSIM[34] value on the whole validation dataset. Based on this validation metric, trials could be pruned sooner than after five epochs. The parameters from the trial with the highest SSIM score were selected after conducting 50 trials. These parameters were then used for training the networks. The search space for the UNet consisted of parameters:

$$\texttt{learning\_rate} \in [10^{-5}, 10^{-1}], \qquad \texttt{dropout\_rate} \in [10^{-5}, 0.5],$$
$$\texttt{loss} \in \{\texttt{mse}, \texttt{mae}\}, \qquad \texttt{kernel\_size} \in \{2, 4, 6, 8\}.$$

Odd sizes of kernels were skipped because of artifacts mentioned in the section 1.4.1, created when the kernel size is not divisible by the stride, which is 2×2 in the case of transposed convolution in the UNet. The best values from the search were:

$$\texttt{learning\_rate} \approx 0.0100, \quad \texttt{dropout\_rate} \approx 0.0160, \quad \texttt{loss} = \texttt{mse},$$
$$\texttt{kernel\_size} = 2.$$

The search space for the GUNet consisted of parameters:

$$\texttt{learning\_rate} \in [10^{-5}, 10^{-1}], \qquad \texttt{dropout\_rate} \in [10^{-5}, 0.5],$$
$$\texttt{loss} \in \{\texttt{mse}, \texttt{mae}\}, \qquad \texttt{epsilon} \in [10^{-5}, 1],$$
$$\texttt{radius} \in \{1, 2, 3\}.$$

The best values from the search were:

$$\texttt{learning\_rate} \approx 0.0031, \quad \texttt{dropout\_rate} \approx 0.0024, \quad \texttt{loss} = \texttt{mse},$$
$$\texttt{epsilon} \approx 0.1483, \quad \texttt{radius} = 2.$$

[33] To learn more about the TPE algorithm, please refer to the paper cited as [22, pg. 4].
[34] See the section 1.2.4.

2.4.2 Training

Both models were trained with the parameters found in the search for 100 epochs, again with the Adam optimizer. During training, I measured the progress with SSIM, MAE, and MSE. When computing these metrics, I have set the models to evaluation mode. When the models are in this mode, particular layers like Dropout and BatchNorm will behave differently to ensure the network produces deterministic outputs. In this mode, dropout layers will not drop any activations, and batch normalization layers will use the running mean and variance calculated during training rather than the statistics of the current batch.

The best values in these metrics measured on the validation dataset for the UNet were:

$$\texttt{SSIM} : 0.8713, \qquad \texttt{MAE} : 0.01269, \qquad \texttt{MSE} : 0.0019390$$

and the best values for the GUNet were:

$$\texttt{SSIM} : 0.8693, \qquad \texttt{MAE} : 0.01290, \qquad \texttt{MSE} : 0.0019398.$$

Progress of the training is captured in figure 2.6. Both networks' capabilities are comparable, although UNet performed slightly better in all metrics- The UNet was trained in a shorter amount of time, as shown in figure 2.6b[35].

[35] The model architecture may not have affected this as the training was not the only process running on the GPUs

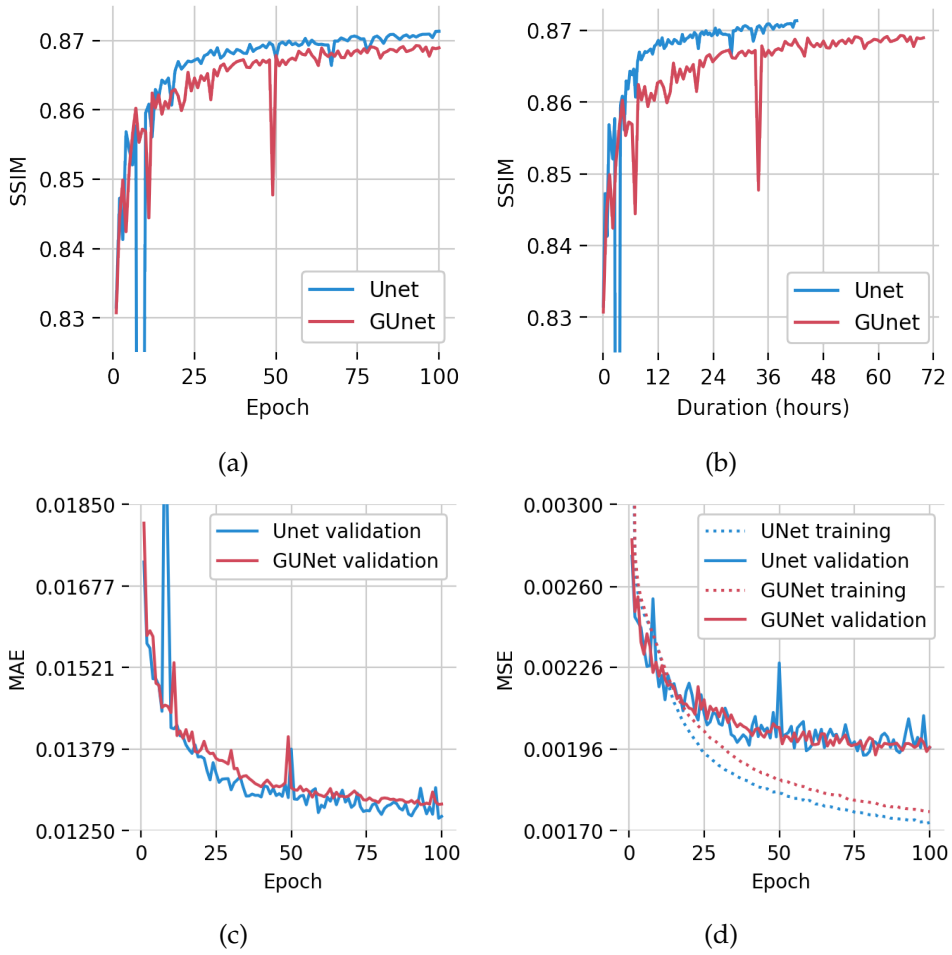


Figure 2.6: Training progress quantified by SSIM, MAE and MSE. Relationship between SSIM and the number of epochs can be seen in (a), while (b) shows SSIM as a function of training time. MAE on the validation dataset is in (c). MSE was measured on the training data, too, as it was used as a loss function for training. Comparison between training and validation loss can be seen in (d).

CHAPTER 3

Evaluation and Results

Models that achieved the best validation SSIM values during the training were analyzed. For the investigation, I used the third distinct dataset. I used full resolution 544×352 images, without cropping, as the factors of the image sides are $544 = 2^5 \times 17^1$ and $352 = 2^5 \times 11^1$. Therefore the five max-pooling layers with 2×2 kernel and 2×2 will behave the same as in training. I tested the models in evaluation mode and on the same metrics as in the section 2.4.2. While the models in evaluation mode are not inherently stochastic, they can introduce some level of randomness if there are any numerical instabilities during the computation. I ran ten tests on the dataset to obtain accurate results and calculated the average. Average values over the runs for the UNet were:

`SSIM` : 0.8784421, `MAE` : 0.0126786, `MSE` : 0.0021389

and for the GUNet were:

`SSIM` : 0.8782260, `MAE` : 0.0125654, `MSE` : 0.0020741.

3.1 Comparison of Forecasts

Both networks generally fail to predict the location of the most intense storms and tend to spread the higher-intensity clusters more and more as the predictions advance. To present the differences in the weather forecasts produced by the models, I created a visualization of the outputs side by side with the observed weather phenomena. I decided to crop all the ground truth observations and network outputs to images with resolution 256×256, as portrayed in figure 3.1, to better visualize the cloud structure.

For further improvements in the visual comparison, I restricted the color bar in the visualization to only show values between lower bound l and upper bound h. I adjusted the alpha value of each pixel x in each image to have the value:

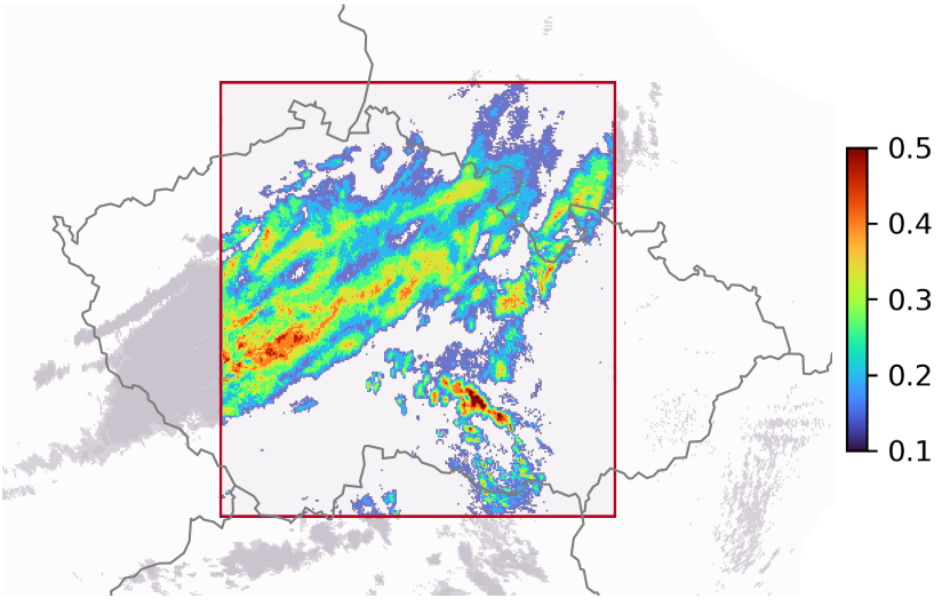


Figure 3.1: Cutout which was used for images in figure 3.2.

$$f(x) = \max\left(0, \tanh\left(4\pi\frac{(x-l)}{h-l}\right)\right). \tag{3.1}$$

The alpha values guarantee a smooth change to transparency when the pixel's value approaches l. If a pixel's value is greater than h, it is adjusted to match the value of h.

Figure 3.2 illustrates the predictions made by GUNet and UNet for a particular set of inputs and targets. Only four out of the eight targets and outputs are displayed, while the inputs are not shown. The predictions are spaced 20 minutes apart, enabling more evident observation of cloud movement and future prediction changes. The SSIM and MAE computed from the cutouts are provided below each prediction.

To see additional examples, please refer to figures B.3, B.4, B.5, and B.6 in appendix B. In figure B.3, you will find a situation with higher-intensity radar echoes, while figure B.4 displays more complex cloud formations on radar images. The other images were chosen from a randomized batch of data points from the test dataset to ensure an unbiased selection. To guarantee that the images included radar echoes, To sort the data points, I utilized the mean value of the pixels in the inputs and then chose the images that had the highest mean values.

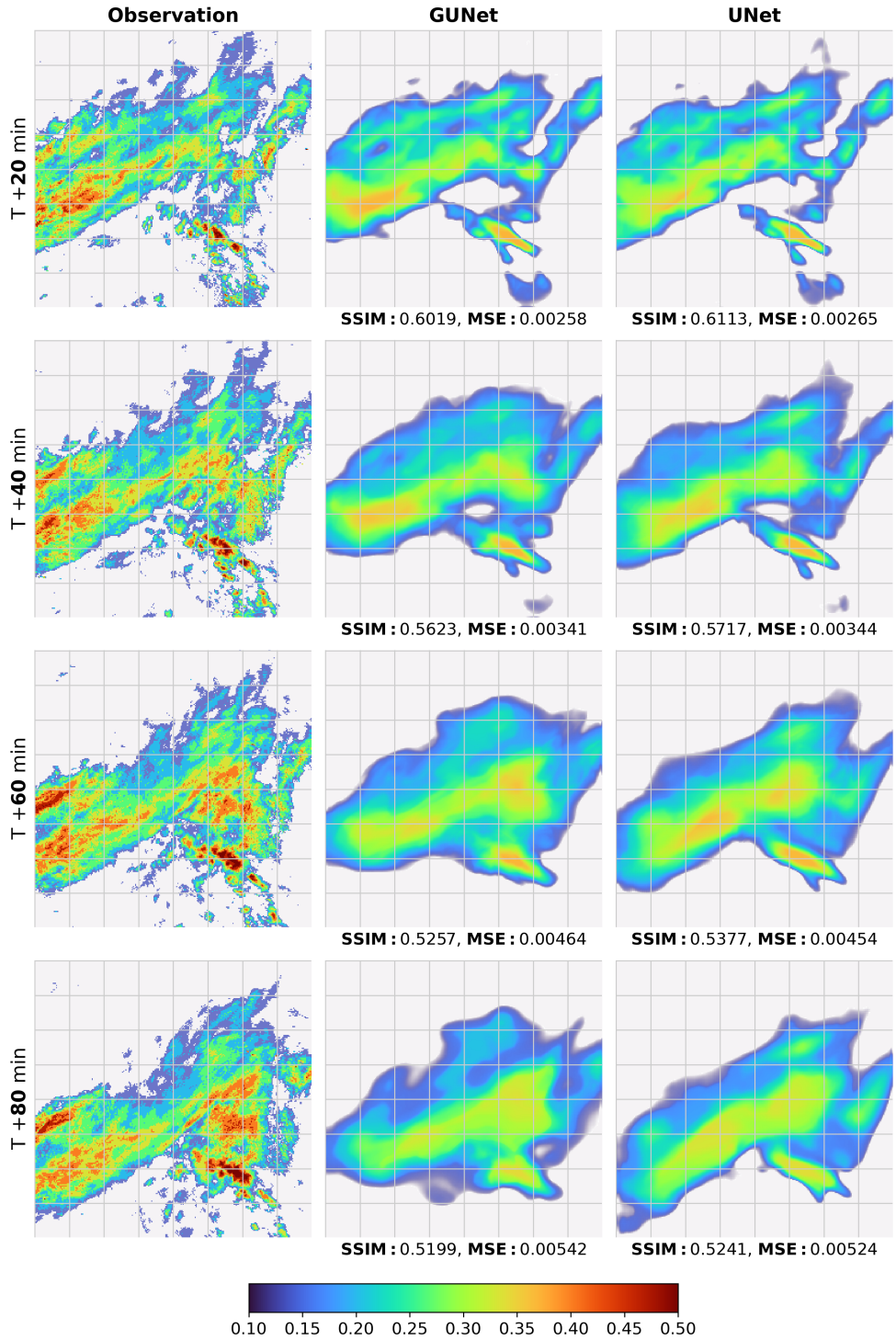


Figure 3.2: Comparison of UNet and GUnet outputs with ground truth observations.

3.1.1 Forecast of Higher Intensity Storms

To evaluate the performance of the models on higher-intensity storms, I set the values in the output and target tensors to 0 if they were below some threshold. I measured the metrics SSIM, MAE, and MSE for different threshold values on the test dataset, the same way as I mentioned at the beginning of the chapter. Figure 3.3 shows that GUNet achieved better performance than the UNet when $\texttt{threshold} \geq 0.2$.

As the threshold increases, fewer pixels are available for model comparison, leading to a decrease in performance differences between the models. At a $\texttt{threshold} \approx 0.35$, both models achieved the same level of performance in all metrics. Therefore, the discrepancy observed at $\texttt{threshold} = 0.2$ is unrelated to the natural decrease of the differences. If the threshold is high enough, both models can achieve an MSE and MAE of 0 and an SSIM of 1. The appendix includes figures B.1a and B.1b, demonstrating the relationship between threshold and MAE and MSE.

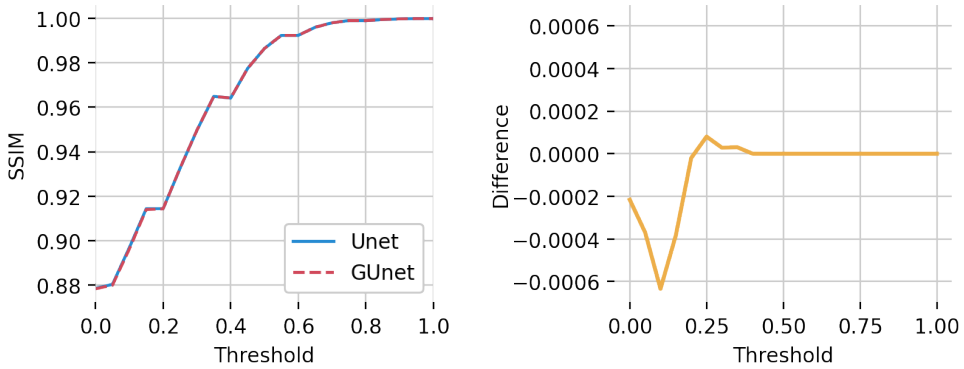


Figure 3.3: The test dataset was used to calculate the average SSIM, comparing outputs and targets. Any values below a specified threshold were considered to be 0. The chart on the right displays the difference between $\texttt{ssim\_gunet} - \texttt{ssim\_unet}$ based on the threshold value. Negative values indicate that UNet performed better in SSIM, and positive values indicate that GUNet had better SSIM.

3.2 Impact of Guided Upsampling on Spectral Bias

In order to determine whether the GU had any influence on the spectral bias by eliminating the structural bias[36], I followed a similar method to the one in the paper [1]. I compared the average Fourier spectra of the targets and both models. I did this by averaging all outputs over the whole test dataset and

[36] Reasoning behind this hypothesis was explained in section 1.4

computing a spectrum of the average using a 2D Fourier transform, equivalent to computing the Fourier spectra for each image and averaging that[37]. Let $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$ be the set of data points, where each $d_i = \{I_{i1}, I_{i2}, \ldots, I_{i8}\}$ represents a set of 8 images. I defined the function `AvgMagSpectrum` to compute the magnitude spectrum of the average images across all data points and within each data point:

$$\texttt{AvgMagSpectrum}(\mathcal{D}) = \left| \mathcal{F}\left(\frac{1}{8N} \sum_{i=1}^{N} \sum_{j=1}^{8} I_{ij} \right) \right|, \tag{3.2}$$

where $\mathcal{F}\{\cdot\}$ denotes the Fourier Transform, and $|\cdot|$ represents the magnitude of the complex Fourier coefficients.

The function can be applied to the sets $\mathcal{D}_{\texttt{GUNet}}$, $\mathcal{D}_{\texttt{UNet}}$, and $\mathcal{D}_{\texttt{tgt}}$ to compute the magnitude spectra $\bar{M}_{\texttt{GUNet}}$, $\bar{M}_{\texttt{UNet}}$, and $\bar{M}_{\texttt{tgt}}$ for the respective GUNet outputs, UNet outputs, and targets. After computing the magnitude spectra, the zero-frequency component was shifted to the center of the spectrum for better visualization and analysis. $\bar{M}_{\texttt{GUNet}}$, $\bar{M}_{\texttt{UNet}}$, and $\bar{M}_{\texttt{tgt}}$ are displayed in figure 3.4.

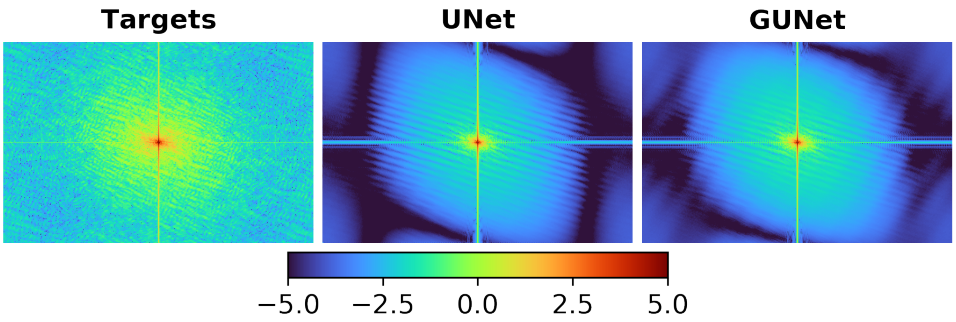


Figure 3.4: Average magnitude spectrum of UNet outputs $\bar{M}_{\texttt{UNet}}$, GUnet outputs $\bar{M}_{\texttt{GUNet}}$ and ground truth $\bar{M}_{\texttt{tgt}}$ on a logarithmic scale.

At first glance, the differences may not be immediately apparent. GUNet seems to capture more of the higher frequencies[38]. To investigate this more thoroughly, I calculated the absolute value of the difference between the average spectrum of the targets and the average spectrum of the network outputs by computing:

$$\texttt{diff}\left(\bar{M}_m\right) = \left|\bar{M}_{\texttt{tgt}} - \bar{M}_m\right|, \tag{3.3}$$

where $\bar{M}_m$ is $\bar{M}_{\texttt{UNet}}$ or $\bar{M}_{\texttt{GUNet}}$. The result of this can be seen in figure 3.5. In the appendix B, I included figure B.2a, which shows a different plot of the same

[37] The reason behind it is the linearity of the Fourier transform
[38] Higher frequencies are the ones further away from the center.

data, and figure B.2b that shows the difference between the output spectra of both models. Values were passed through moving average with a window size of 32×32, so the differences between the two spectra can be seen more clearly.

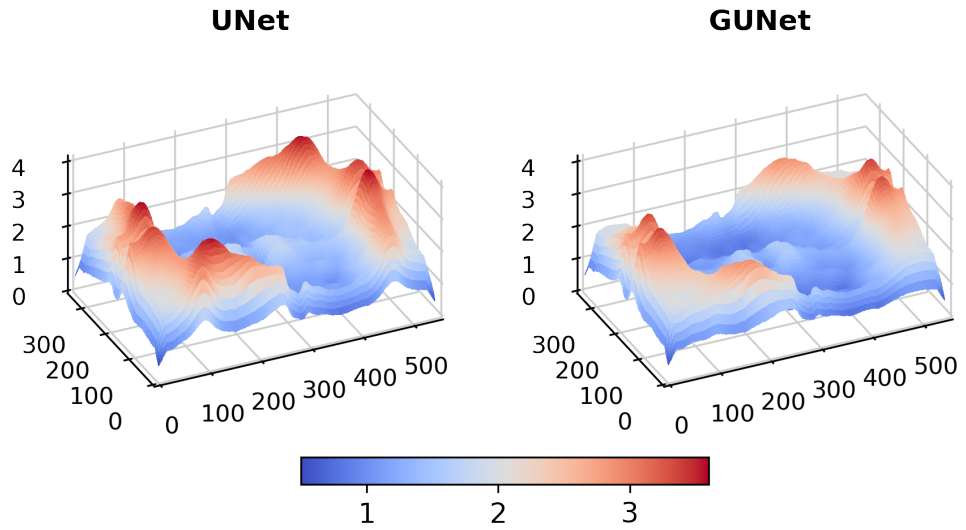


Figure 3.5: Absolute difference between the amplitudes of the average target spectrum and the average output spectra of models on a logarithmic scale. Values were smoothed by moving average with kernel size 32×32. An unsmoothed 2D chart is in figure B.2a. A higher pixel value means that the amplitude, at the frequency corresponding to that pixel, differed more from the desired amplitude of the average target spectrum.

Chapter 4

Discussion

4.1 Interpretation of Results

From the average Fourier spectrum of the UNet $\bar{M}_{\texttt{UNet}}$ in figure 3.4 is clear that the network did not produce the structural artifacts[39] because it produced $\bar{M}_{\texttt{UNet}}$ without the checkerboard-like structure which was present in results of a similar experiment in the paper [1, fig. 3]. The UNet in the paper[40] utilized a 4x4 kernel in their transposed convolution, while my UNet used a 2x2 kernel[41].

The smaller kernel may have helped to mitigate the artifacts since with 2×2 kernel with a 2×2 stride means that each pixel in the output of the transposed convolution is a function of just one input pixel. Whereas with a 4×4 kernel and the same stride, each output pixel is affected by four input pixels. In contrast, the GIF in GUNet was used with `radius` = 2, which means that output pixels are assumed to be a linear transformation of the input image with coefficients computed on a window ω_k, so in the case of `radius` = 2, the size of this window is $|\omega_k| = (2r + 1)^2 = 25$.

UNet may have learned to mitigate these artifacts, or they were not created due to the tiny kernel. Either way, it may have been at the expense of worse performance than the GUNet when I compare the average MAE and MSE on the test dataset. While the UNet was better in both metrics on the validation dataset, on the test dataset, GUNet outperformed the UNet. The SSIM is also interesting to compare. UNet scored better, but the margin has shrunk ten times from 0.002 to 0.0002. Intriguingly, the SSIM values improved on the testing dataset. The reason behind it could be the randomness in creating the datasets. Both networks were tested and trained on the same datasets so they could be compared on them. The SSIM on the validation dataset has practically only risen during training, so it is unlikely that overfitting of the UNet

[39] See the section 1.4.1.

[40] The paper [1] presents various UNet architectures. Specifically, I am talking about the one called Transposed Convolution UNet or"TC-UNet," as named in the paper.

[41] Size of the kernel was found in hyperparameter search. See section 2.4.1.

is the reason for its declined performance. However, it has more parameters than the GUNet, which would make it more likely to do so.

Another critical difference between the models, which can be seen in the results, is the ability of the GUNet to capture higher frequencies than the UNet. By itself, capturing higher frequencies does not imply better weather predictions, but the improvement is significant because of the spectral bias present in NNs[42] .

Weather nowcasting aims to anticipate severe weather conditions and safeguard us from dangers caused by rapidly changing weather patterns. In this regard, predicting high-intensity levels in radar images is more advantageous. Figures 3.3, B.1a, and B.1b indicate that the performance of the GUNet was superior to that of the UNet in images where lower intensities were removed. Although the improvement was relatively small, as shown in the figures.

Granting all these differences, when the weather predictions of both networks are compared to the target predictions, they look nearly identical.

4.2 Limitations and Potential Improvements

My approach has some limitations, some of which I already mentioned earlier. Firstly, I did not remove nearly empty images from the dataset, which may have impacted the training time and the performance of the models.

The second limitation I already mentioned is padding in the convolutions. If I instead allowed the convolution outputs to decrease each time, the network's final output would be much smaller than the input, but it would lead to better performance around the edges of the image.

Next, I would improve the speed of the GUNet. In the implementation of GUNet, I used the standard GIF algorithm A, but there is a faster algorithm A.2, which may improve the long training time observed in the figure 2.6b.

Finally, while the differences in the performance of the models are there, it is paramount to mention that they were observed on just one pair of networks and trained only once, which means that the results are not statistically significant and to obtain more definite outcomes, additional, comprehensive research is necessary. This research could involve exploring additional hyperparameters, like varying kernel sizes in each layer and parameterizing the model depth. The search for hyperparameters could also be longer, letting the networks run for more epochs. Final training could be done for more epochs and multiple times. Different, more complex loss functions could also be tried.

[42] I. e. tendency of neural networks to learn lower frequencies more easily. See the section 1.4.2.

Conclusion and Future Work

This thesis has sought to enhance the accuracy and reliability of short-term weather predictions, notably storm structure predictions from radar images, by addressing the structural biases present in deep learning models like the UNet architecture. By adapting and applying the Guided Upsampling technique from the Guided UNet (GUNet) architecture to weather nowcasting, I could effectively mitigate the adverse effects of transposed convolution on weather predictions. This approach led to the development of a more spectrally consistent model.

Through a comprehensive literature review, the reader was introduced to the concepts of weather nowcasting, CNNs, the UNet architecture, GIF, and spectral bias. A dataset of radar composites from a network of weather radars created by OPERA was utilized for training and evaluating the GUNet and UNet models. The impact of GU was analyzed on the GUNet and UNet models using metrics such as MSE, MAE, and SSIM.

The results demonstrated that, while the UNet successfully mitigated the structural artifacts, the GUNet model outperformed it in terms of average MAE and MSE on the test dataset. Additionally, the GUNet demonstrated an improved ability to capture higher frequencies compared to the UNet, which is a significant finding considering the spectral bias issue in neural networks.

Despite the improvements observed in the GUNet model, some limitations and potential improvements were identified, including the removal of nearly empty images from the dataset, addressing padding in convolutions, improving the speed of the GUNet using a faster GIF algorithm, and conducting a more exhaustive investigation of the models by multiple training runs with more hyperparameters.

The implications for weather nowcasting suggest that the GUNet model's improved performance on images with higher radar echo intensities may contribute to better predictions of severe weather events, which are of utmost importance for public safety and infrastructure protection. Although the performance improvement is relatively small, it still signifies progress in the ongo-

ing efforts to create more precise and dependable weather forecasts.

In conclusion, this thesis has successfully, even if marginally, advanced the field of weather nowcasting by addressing structural biases in convolutional neural networks used in this field.

Summary of Findings

The key findings of this thesis are as follows:

- The GUNet model, which incorporates GU, demonstrated improved performance compared to the UNet model in terms of average MAE and MSE on the test dataset while achieving nearly identical average SSIM, which suggests that the GUNet model provides a marginal improvement in accurate and reliable short-term weather predictions from radar images.
- The GUNet model more successfully captured higher spatial frequencies than the UNet model.
- The GUNet model performed better than the UNet model on images with higher radar echo intensities, which is particularly important for predicting severe weather events in the context of weather nowcasting.

Future Research Directions

Based on the findings and limitations of this thesis, the following future research directions are proposed:

- More extensive hyperparameter searches, including parameters such as model depth and different kernel sizes in each layer, could lead to a better comparison of both models and improvements in performance.
- Utilizing a faster GIF algorithm in the GUNet model could reduce training time, allowing for more extensive experimentation and potentially better model performance in the same training time. Removing nearly empty images from the dataset could also lead to similar improvements.
- Testing different, more complex loss functions may improve the model's ability to predict storm structures and contribute to more accurate and reliable weather forecasts.
- Investigating if using GIF in other deep learning models and techniques, such as recurrent GANs or stable diffusion, could improve their performance.

Bibliography

[1] Marnerides, D.; Bashford-Rogers, T.; et al. Spectrally Consistent UNet for High Fidelity Image Transformations. 2020, 2004.10696.

[2] Browning, K. A. Conceptual Models of Precipitation Systems. *Weather and Forecasting*, volume 1, no. 1, 1986: pp. 23 – 41, doi:https://doi.org/10.1175/1520-0434(1986)001<0023:CMOPS>2.0.CO;2. Available from: https://journals.ametsoc.org/view/journals/wefo/1/1/1520-0434_1986_001_0023_cmops_2_0_co_2.xml

[3] Wang, Y.; Coning, E.; et al. *Guidelines for Nowcasting Techniques*. 11 2017, ISBN 978-92-63-11198-2.

[4] Rumelhart, D. E.; Hinton, G. E.; et al. Learning representations by back-propagating errors. *Nature*, volume 323, no. 6088, 1986: pp. 533–536, doi:10.1038/323533a0. Available from: https://doi.org/10.1038/323533a0

[5] Hornik, K.; Stinchcombe, M.; et al. Multilayer feedforward networks are universal approximators. *Neural Networks*, volume 2, no. 5, 1989: pp. 359–366, ISSN 0893-6080, doi:https://doi.org/10.1016/0893-6080(89)90020-8. Available from: https://www.sciencedirect.com/science/article/pii/0893608089900208

[6] Krizhevsky, A.; Sutskever, I.; et al. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems*, volume 25, 01 2012, doi:10.1145/3065386.

[7] Dumoulin, V.; Visin, F. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, mar 2016, 1603.07285.

[8] Goodfellow, I.; Bengio, Y.; et al. *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[9] Odena, A.; Dumoulin, V.; et al. Deconvolution and Checkerboard Artifacts. *Distill*, 2016, doi:10.23915/distill.00003. Available from: `http://distill.pub/2016/deconv-checkerboard`

[10] Careddu, G. Italiano: Ravenna, radar dell'aeroporto. 2018, [Online; accessed 2023-05-08]. Available from: `https://commons.wikimedia.org/wiki/File:Ravenna,_radar_dell%27aeroporto_(04).jpg`

[11] Ronneberger, O.; Fischer, P.; et al. U-Net: Convolutional Networks for Biomedical Image Segmentation. 2015, 1505.04597.

[12] Rahaman, N.; Baratin, A.; et al. On the Spectral Bias of Neural Networks. 2019, 1806.08734.

[13] Aitken, A.; Ledig, C.; et al. Checkerboard artifact free sub-pixel convolution: A note on sub-pixel convolution, resize convolution and convolution resize. 2017, 1707.02937.

[14] He, K.; Sun, J.; et al. Guided Image Filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 35, no. 6, 2013: pp. 1397–1409, doi:10.1109/TPAMI.2012.213.

[15] He, K.; Sun, J. Fast Guided Filter. 2015, 1505.00996.

[16] Kopf, J.; Cohen, M. F.; et al. Joint Bilateral Upsampling. *ACM Trans. Graph.*, volume 26, no. 3, jul 2007: p. 96–es, ISSN 0730-0301, doi:10.1145/1276377.1276497. Available from: `https://doi.org/10.1145/1276377.1276497`

[17] Huuskonen, A.; Saltikoff, E.; et al. The Operational Weather Radar Network in Europe. *Bulletin of the American Meteorological Society*, volume 95, no. 6, 2014: pp. 897 – 907, doi:https://doi.org/10.1175/BAMS-D-12-00216.1. Available from: `https://journals.ametsoc.org/view/journals/bams/95/6/bams-d-12-00216.1.xml`

[18] Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015, 1502.03167.

[19] Hinton, G. E.; Srivastava, N.; et al. Improving neural networks by preventing co-adaptation of feature detectors. 2012, 1207.0580.

[20] Wu, H.; Zheng, S.; et al. Fast End-to-End Trainable Guided Filter. In *CVPR*, 2018.

[21] Akiba, T.; Sano, S.; et al. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[22] Bergstra, J.; Bardenet, R.; et al. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, volume 24, edited by J. Shawe-Taylor; R. Zemel; P. Bartlett; F. Pereira; K. Weinberger, Curran Associates, Inc., 2011. Available from: https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf

[23] Kingma, D. P.; Ba, J. Adam: A Method for Stochastic Optimization. 2017, 1412.6980.

Appendix A

Guided Image Filtering algorithms

A.1 Guided Filter

Input: filtering input image p, guidance image I, radius r, regularization ϵ

Output: filtering output q

1: $\texttt{mean}_I = f_{\texttt{mean}}(I, r)$
$\texttt{mean}_p = f_{\texttt{mean}}(p, r)$
$\texttt{corr}_I = f_{\texttt{mean}}(I .* I, r)$
$\texttt{corr}_{Ip} = f_{\texttt{mean}}(I .* p, r)$

2: $\texttt{var}_I = \texttt{corr}_I - \texttt{mean}_I .* \texttt{mean}_I$
$\texttt{cov}_{Ip} = \texttt{corr}_I p - \texttt{mean}_I .* \texttt{mean}_p$

3: $a = \texttt{cov}_{Ip} \,./\, (\texttt{var}_I + \epsilon)$
$b = \texttt{mean}_p - a .* \texttt{mean}_I$

4: $\texttt{mean}_a = f_{\texttt{mean}}(a, r)$
$\texttt{mean}_b = f_{\texttt{mean}}(b, r)$

5: $q = \texttt{mean}_a .* I + \texttt{mean}_b$

The dot before the operation denotes that it is performed elementwise. $f_{\texttt{mean}}(\cdot, r)$ denotes a mean filter with a radius r. Pseudocode was taken from paper [14].

A.2 Fast Guided Filter

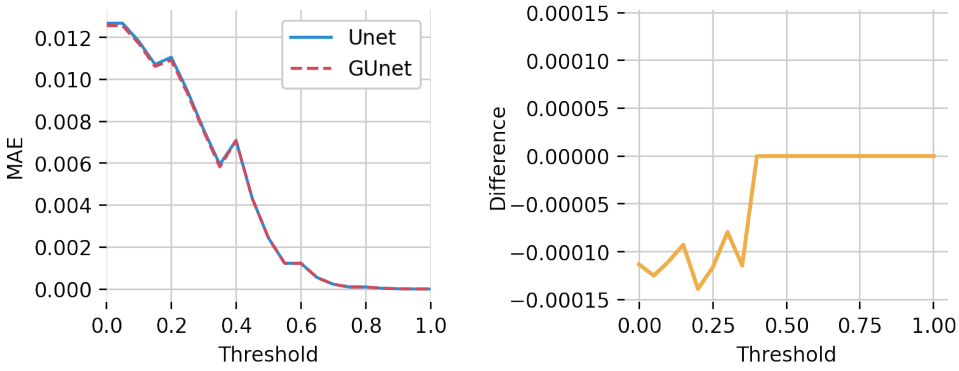
Input: filtering input image p, guidance image I, radius r, regularization ϵ, subsampling scale s
Output: filtering output q

1: $I' = f_{\texttt{subsample}}(I, s)$
$p' = f_{\texttt{subsample}}(p, s)$
$r' = r/s$
2: $\texttt{mean}_I = f_{\texttt{mean}}(I', r')$
$\texttt{mean}_p = f_{\texttt{mean}}(p', r')$
$\texttt{corr}_I = f_{\texttt{mean}}(I'.*I', r')$
$\texttt{corr}_{Ip} = f_{\texttt{mean}}(I'.*p', r')$
3: $\texttt{var}_I = \texttt{corr}_I - \texttt{mean}_I.*\texttt{mean}_I$
$\texttt{cov}_{Ip} = \texttt{corr}_{Ip} - \texttt{mean}_I.*\texttt{mean}_p$
4: $a = \texttt{cov}_{Ip} ./ (\texttt{var}_I + \epsilon)$
$b = \texttt{mean}_p - a.*\texttt{mean}_I$
5: $\texttt{mean}_a = f_{\texttt{mean}}(a, r')$
$\texttt{mean}_b = f_{\texttt{mean}}(b, r')$
6: $\texttt{mean}_a = f_{\texttt{upsample}}(\texttt{mean}_a, s)$
$\texttt{mean}_b = f_{\texttt{upsample}}(\texttt{mean}_b, s)$
7: $q = \texttt{mean}_a.*I + \texttt{mean}_b$

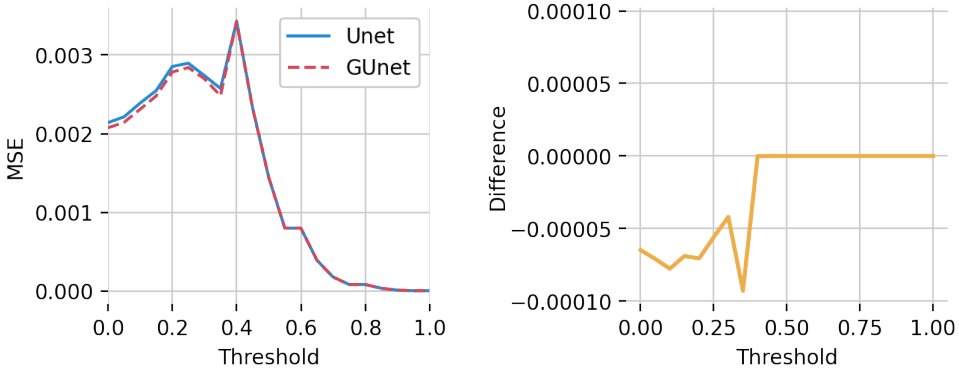
The dot before the operation denotes that it is performed elementwise. $f_{\texttt{mean}}(\cdot, r)$ denotes a mean filter with a radius r. Pseudocode was taken from paper [15]. Subsampling (nearest-neighbor or bilinear) of the input p and the guidance I by a ratio s is preformed to speed up the guided filter. All the box filters are performed on the low-resolution maps, which are the major computation of the guided filter. The two coefficient maps $\bar{a}$ and $\bar{b}$ are bilinearly upsampled to the original size. Finally, the output q is still computed by $q = \bar{a}I + \bar{b}$. In this last step, the image I is the full-resolution guidance that is not downsampled, and it will still faithfully guide the output. [15]

Appendix B

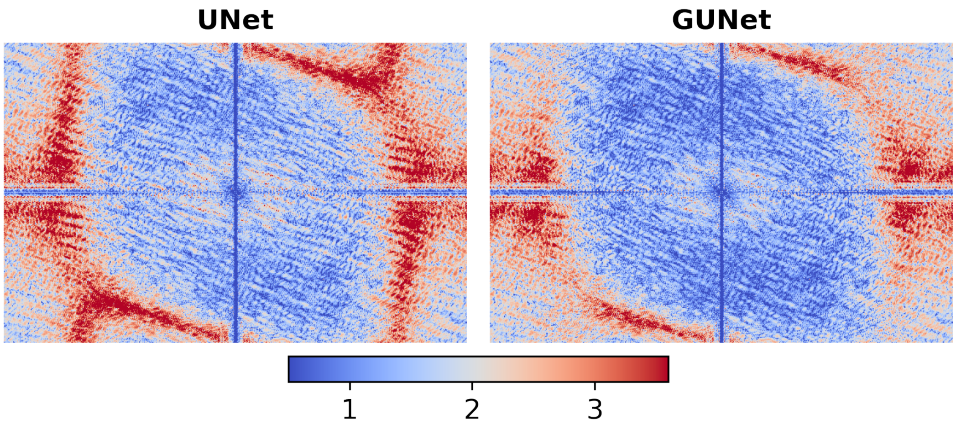
Model comparisons



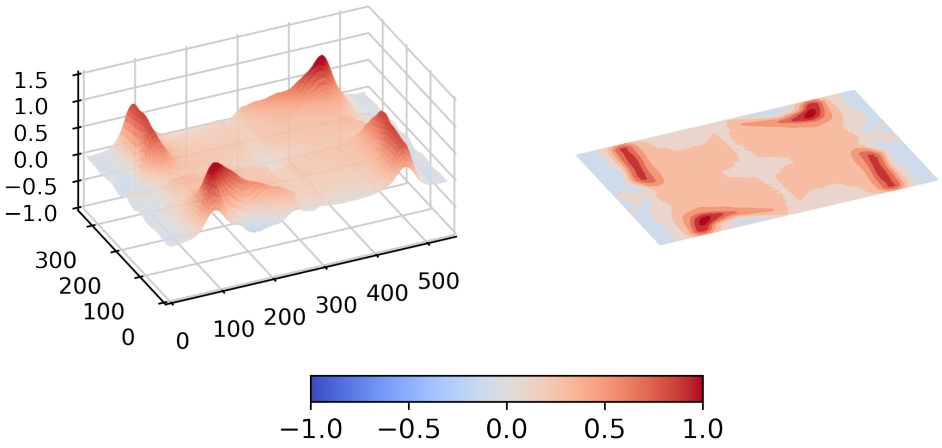
(a) Average MAE on the test dataset, computed from outputs and targets, which had some values, that were below some threshold, set to 0. Chart on the right shows the difference `mae_gunet` − `mae_unet`. Negative values mean that GUNet had better MAE, and positive values that UNet had better MAE.



(b) Average MSE on the test dataset, computed from outputs and targets, which had some values, that were below some threshold, set to 0. Chart on the right shows the difference `mse_gunet` − `mse_unet`. Negative values signify better GUNet MSE.



(a) Absolute difference between the amplitudes of the average target spectrum $\bar{M}_{\texttt{tgt}}$ and the average output spectra of the models $\bar{M}_{\texttt{UNet}}$ and $\bar{M}_{\texttt{GUNet}}$, on a logarithmic scale. Higher value of a pixel means, that the amplitude at the frequency corresponging to that pixel, differed more from the desired amplitude of average target spectra $\bar{M}_{\texttt{tgt}}$.



(b) Amplitudes of the average UNet output spectrum $\bar{M}_{\texttt{UNet}}$ subtracted from the amplitudes of the average GUNet outputs $\bar{M}_{\texttt{GUNet}}$ on a logarithmic scale. Values were passed through moving average with kernel size 32×32. Higher value of a pixel means, that GUNet outputs had higher average amplitude than the UNet outputs, at the frequency corresponging to the given pixel.

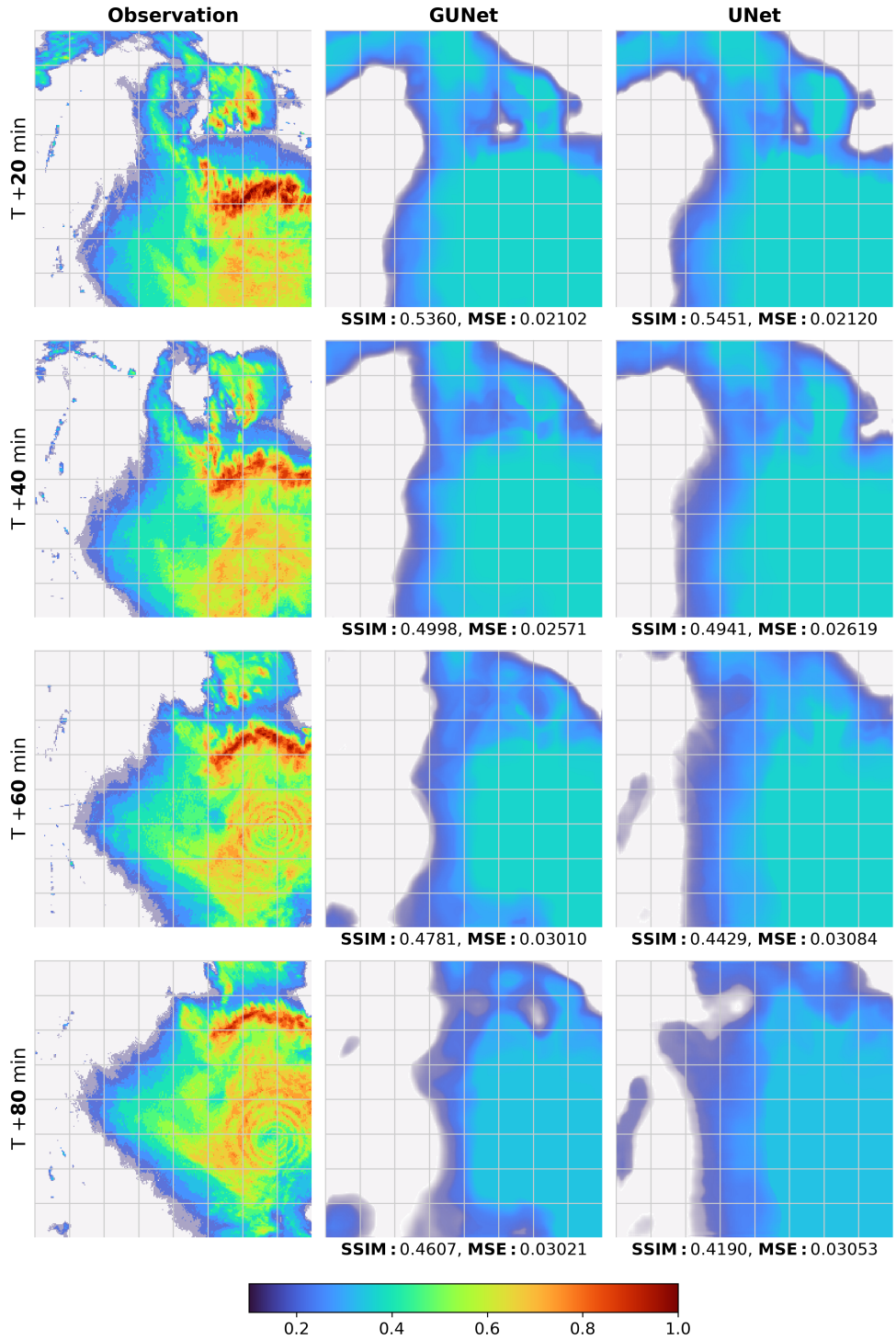


Figure B.3: Model comparison on radar images with very high echo intensities, which are not captured by either one of the models.

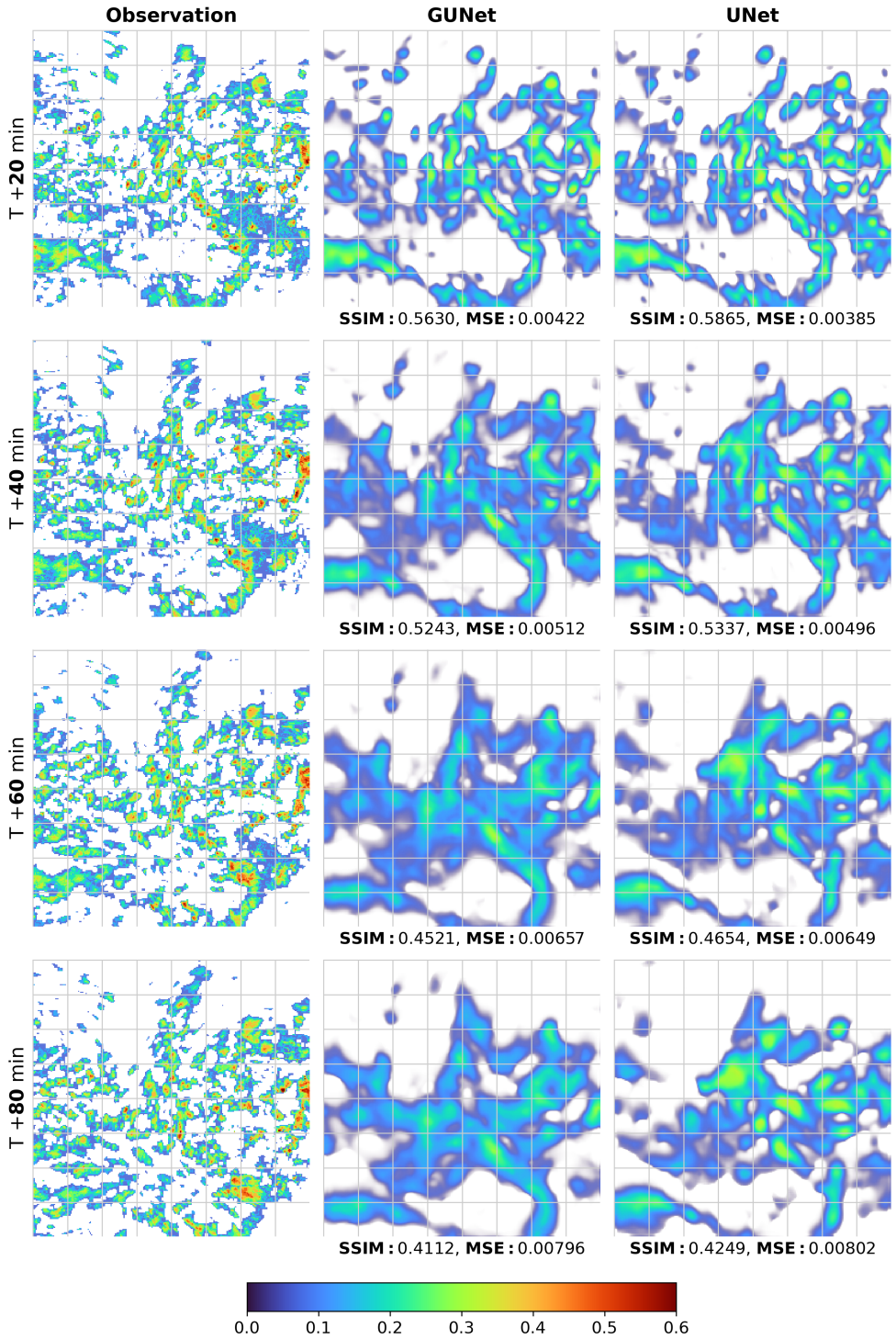


Figure B.4: Model comparison on radar images with complex cloud formations.

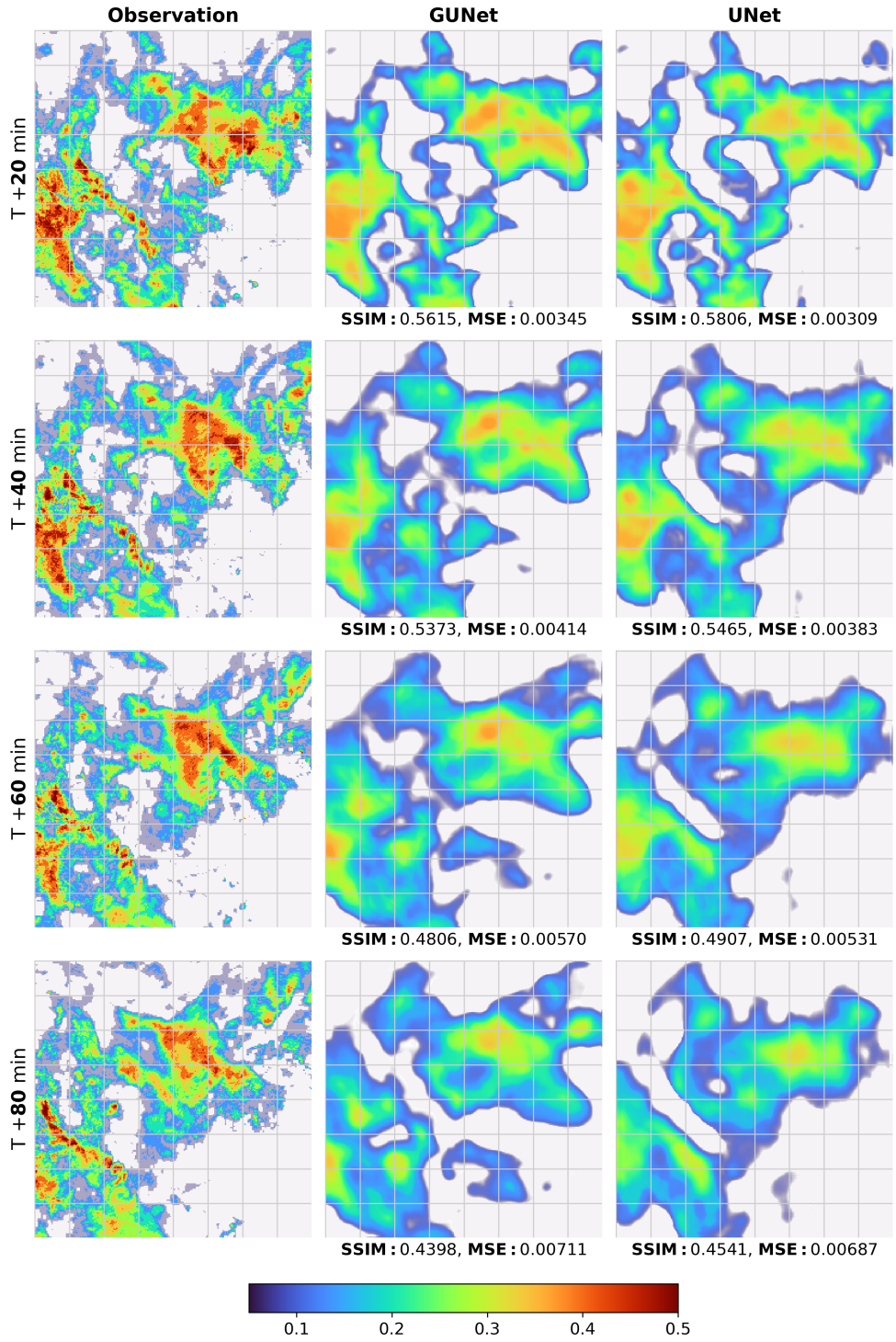

Observation
GUNet
UNet
T +20 min
SSIM : 0.5615, MSE : 0.00345
SSIM : 0.5806, MSE : 0.00309
T +40 min
SSIM : 0.5373, MSE : 0.00414
SSIM : 0.5465, MSE : 0.00383
T +60 min
SSIM : 0.4806, MSE : 0.00570
SSIM : 0.4907, MSE : 0.00531
T +80 min
SSIM : 0.4398, MSE : 0.00711
SSIM : 0.4541, MSE : 0.00687
0.1
0.2
0.3
0.4
0.5

Figure B.5

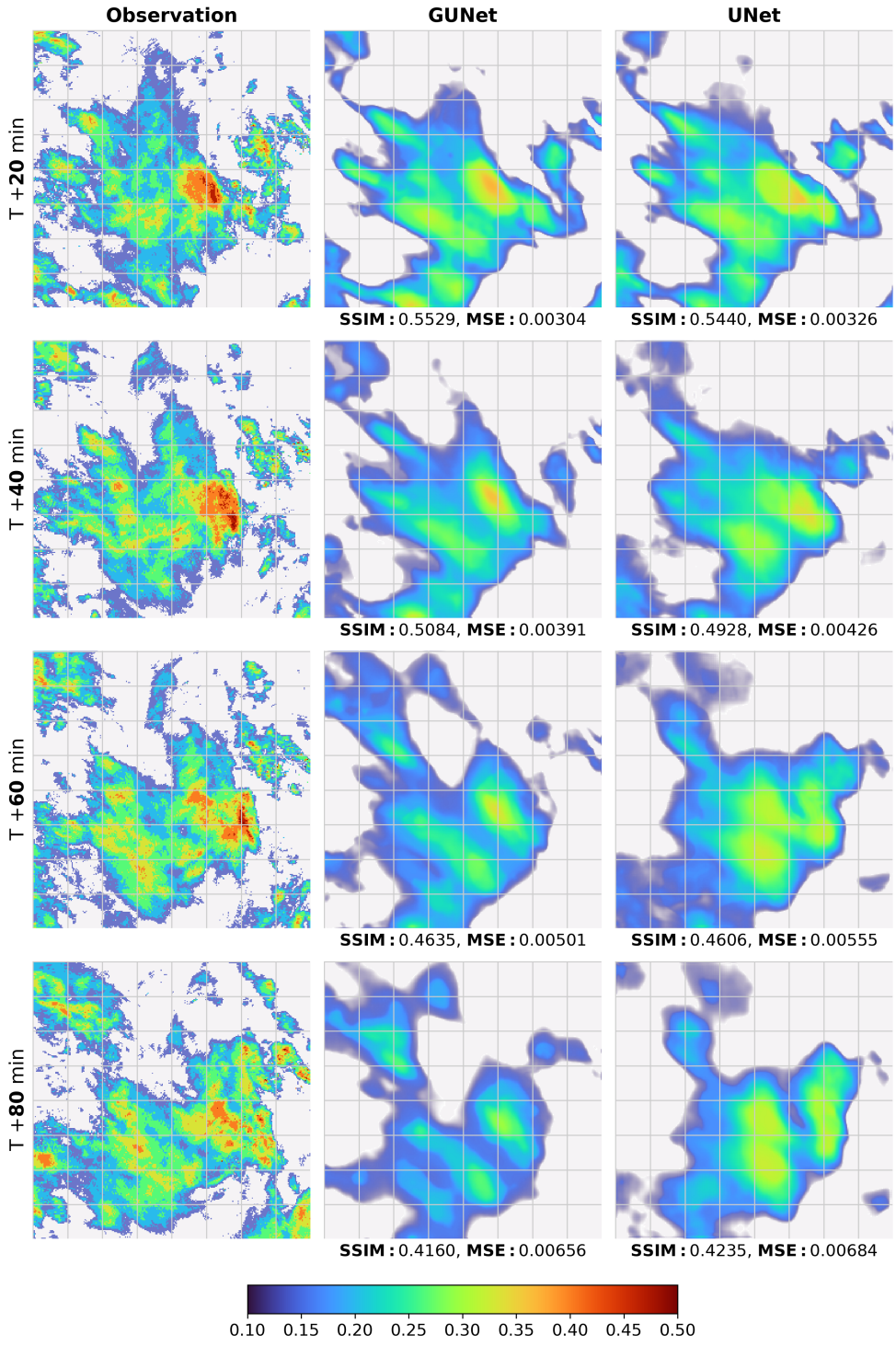

Observation
GUNet
UNet
T +20 min
SSIM : 0.5529, MSE : 0.00304
SSIM : 0.5440, MSE : 0.00326
T +40 min
SSIM : 0.5084, MSE : 0.00391
SSIM : 0.4928, MSE : 0.00426
T +60 min
SSIM : 0.4635, MSE : 0.00501
SSIM : 0.4606, MSE : 0.00555
T +80 min
SSIM : 0.4160, MSE : 0.00656
SSIM : 0.4235, MSE : 0.00684
0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50

Figure B.6

Appendix C

Acronyms

OPERA The European Operational Program for Exchange of Weather Radar Information. 2, 23, 24, 41

CNN Convolutional Neural Network. 1, 3, 7, 10–15, 21, 41

FGF Fast Guided Filter. 19, 20

GAN Generative Adversarial Network. 14, 42

GIF Guided Image Filter. 18, 20, 27, 28, 39–42

GU Guided Upsampling. 2, 3, 20, 27, 28, 36, 41, 42

GUNet Guided UNet. 1–3, 18, 20–22, 27–29, 33, 34, 36, 37, 39–42

JBU Joint Bilateral Upsampling. 20

MAE Mean Absolute Error. 2, 3, 12, 22, 30, 31, 33, 34, 36, 39, 41, 42, 49

MSE Mean Squared Error. 2, 3, 7, 12, 22, 30, 31, 33, 36, 39, 41, 42, 49

NN Neural Network. 3, 7, 9, 10, 12, 14–16, 40

SSIM Structural Similarity Index Measure. 2, 3, 11, 12, 22, 29–31, 33, 34, 36, 39, 41, 42

TPE Tree-structured Parzen Estimator. 29

WMO World Meteorological Organization. 5

Appendix D

Contents of the Archive

- `README.md` ..... information about the contents of the archive
- `src` ..... directory containing the implementation
  - `dataset.py` ..... script for generating the datasets
  - `tuning.py` ..... script for hyperparameter tuning
  - `train.py` ..... model trainining script
  - `gunet.py` ..... implementation of GUNet in PyTorch
  - `unet.py` ..... implementation of UNet in PyTorch
  - `gunet_weights_best.pt` ..... weights of the best GUNet
  - `unet_weights_best.pt` ..... weights of the best UNet
  - `visualization.ipynb` .. notebook used for creating the visualizations
- `thesis` ..... directory containing the text
  - `chapters` ..... directory containing text of the chapters
  - `images` ..... directory containing images used in the thesis
  - `thesis.tex` ..... main LaTeX source file
  - `bibliography.bib` ..... bibliography
  - `thesis.pdf` ..... this pdf