On the usage of wavelet-based techniques for Synthetic Image Detection

by

Arian Joyandeh

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday August 25, 2023 at 13:00.

Student number:	4568745	
Project duration:	September 5, 2022 – August 25, 2023	
Thesis committee:	Prof. dr. ir. G. Jongbloed,	TU Delft
	Dr. H. N. Kekkonen,	TU Delft
	Prof. dr. M. Loog,	TU Delft

Abstract

With the rise of zero-shot synthetic image generation models, such as Stability.ai's Stable Diffusion [36], OpenAI's DALLE [34] or Google's Imagen [37], the need for powerful tools to detect synthetic generated images has never been higher. In this thesis we contribute to this goal by considering wavelet-based approaches for synthetic image detection.

We will introduce multi-level discrete wavelet transform, which to the best of our knowledge has never been considered for this goal prior to this work. A similar approach that has been considered for the goal of synthetic image detection, is the multi-level wavelet packet transform used by Wolter et al. [48]. We will show that not only is our proposed approach more efficient and easier interpretable, it also performs better in a number of experimental settings and therefore forms a suitable addition to the toolset for the detection of synthetic images.

Moreover, we will try and generalize performance of our used classifiers to out-of-dataset samples and see that our used classifier in general does not allow for such generalization. Finally, we will discuss the challenges of this work and offer interesting directions for further research.

Preface

This thesis is about the usage of wavelet-based approaches for synthetic image detection. I have written this thesis as part of my double master degree Applied Mathematics and Computer Science at the TU Delft.

Just before I started my thesis journey, Stability.ai released their open-source Diffusion Model, Stable Diffusion. Since the training of such models usually takes thousands of hours and a lot capital, the free availability of this model caught a lot of attention. All kinds of interesting tools were made by the developers that used this open-source model. However, similarly, the open-source nature of this model also provided an avenue for malicious actors to develop their own tools.

The importance for tools to detect synthetically generated images was clear to me. I was happy that I could spend time to understand how these models work. I have been intrigued by the methods researchers have tried to detect these type of images and what challenges they faced and that I have also faced similar challenges. It made me feel as a small part of the scientific community.

Writing this thesis for both Applied Mathematics and Computer Science has been a challenge: Finding a subject that is interesting from both perspectives, is not an easy feat. However, both of my thesis supervisors have supported me well in this process and for that I want to thank dr. Kekkonen and prof. dr. Loog.

I would like to thank the people close to me for their continued support during the process of writing my thesis. Finally, I would like to thank prof. dr. Jongbloed for taking seat in my thesis commitee.

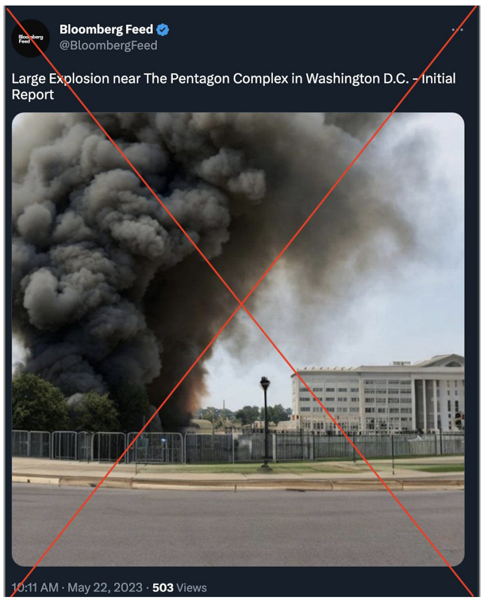
Arian Joyandeh
Delft, August 2023

Contents

1 Introduction

Recent advances in the field of Synthetic Image Generation have made it possible to generate natural seeming images with little effort. Users now merely need to describe the type of image they desire, and the generator can generate it accordingly. While the positive applications of these generators are evident in creative domains like art [40] and game design [31], these generators also come with various potential negative consequences.

One example of a negative consequence of accessible Synthetic Image Generation, were the events following the Tweet of May 22nd, 2023 shown in Figure 1.1. An account seemingly affiliated with Bloomberg shared the news story that a large explosion took place near the Pentagon Complex in Washington D.C. In reality, nothing happened and someone purporting to be Bloomberg shared an AI-generated image of smoke near the Pentagon. Nonetheless, the “news” quickly spread around and caused the S&P 500, one of the most commonly followed equity indices, to lose 500 billion dollars in market cap. The equity index eventually rebounced after the image was confirmed to be fake. Of course multiple facets played part in this event, but the role of the synthetically generated image, is not to be neglected.


Bloomberg Feed
@BloombergFeed

Large Explosion near The Pentagon Complex in Washington D.C. - Initial Report

10:11 AM · May 22, 2023 · 503 Views

Figure 1.1: Screenshot of a fake tweet by the @BloombergFeed Twitter account, reporting a bombing near the Pentagon Complex, accompanied by what seems to be a picture of the bombing. This Twitter account is not related to Bloomberg and the accompanying image is AI generated, the bombing did not occur.

The Al Jazeera Journalism Review eloquently articulated the gravity of these events: *“These events signal that journalism may be on the brink of a major crisis as AI tech can now produce complete* *'journalistic' content and create almost-real images. Media groups need to find practical mechanisms to monitor content generated partially or fully by AI. But how?"*

The need for practical mechanisms to monitor synthetic generated imagery, is exactly the topic of this thesis. There are numerous ways this type of synthetic generated imagery can be detected, but in this thesis we will focus on the usage of wavelets for this goal. The usage of wavelets to this end, is relatively unexplored: Wolter et al. [48] are the first to introduce the usage of wavelets in the setting of Synthetic Image Detection. More specifically, Wolter et al. use a wavelet-packet based approach to this end, without considering the possibility of using the more efficient and easier interpretable regular wavelet approach.

This leads us to the main question of this thesis: "**How can wavelet-based approaches be used towards the goal of synthetic image detection, more specifically using regular wavelets and wavelet packets?**" To gain better insights into this question, we will consider a number of different scenarios in which we will compare the two different approaches.

Firstly, it is important to note that the recent advances in the field of Synthetic Image Generation are partly due to the adoption of Diffusion Models (DMs) instead of Generative Adversarial Networks (GANs). The latter model used to be the state-of-the-art generators, until the seminal paper by Dhariwal and Nichol [8] showed that DMs can achieve superior sample quality over GANs. Thus, **it is of interest to consider how both wavelet-based approaches compare in terms of these different generating models**: Not only might this lead to a better understanding of the two wavelet approaches, but we might also gain a better understanding in the specific characteristics of the two different generators.

Secondly, since wavelets allow for both frequency and spatial analysis, it is to be expected that the two wavelet based approaches might have different results over different datasets. This is what brings us to our second subquestion: **How do regular wavelets compare to wavelet packets in synthetic image detection over different datasets?**

Another question of interest will be one with regard to the classifier used by Wolter et al. In their work, they show that a relative simple classifier can achieve near perfect accuracy. However, they quickly abandon this model for a more complex classifier. However, noticing that this simple classifier already achieved such high accuracy, **we ponder the question to what extent this simple classifier could be used for synthetic image detection**.

To answer these questions of interest, this thesis will start off with Chapter 2 on Synthetic Image Generators, where we will take a look at both GANs and DMs. The goal of these sections is not to give the reader an in-depth understanding of both approaches, but rather to allow for sufficient understanding such that theories and hypotheses based on the inner-workings of synthetic image generators can be formulated and explored.

Following this section, we will consider Chapter 3 on representations. We will start this section by analysing how images can be understood as discrete signals. Afterwards, we will leverage this knowledge of an image as a discrete signal to understand how the Discrete Fourier Transform (DFT) can be used to analyse frequencies apparent in the image. Afterwards, we will introduce wavelets, for which we will need to introduce a number of concepts such as filter and filter banks. The goal of this section is to give the reader an understanding of how images can have different representations that can all potentially be used in Synthetic Image Detection.

Afterwards, Chapter 4 on Synthetic Image Detection is introduced, where we will formalize our problem and take a deeper dive into relevant previous work in this field. Chapter 5 concerns itself with the experimental setup and datasets used in our experiments. Results of these experiments and interpretation can be found in Chapter 6. We will circle back to our original questions of interest in Chapter 7 and spend time on discussion and future recommendations in Chapter 8.

2 Synthetic Image Generators

Synthetic Image Generation is concerned with the question of generation. Given a set of objects e.g. images of alpacas or images of hand-written digits, is it possible to generate a new, similar object? This heavily depends of course on our specific notion of what it means for an object to be similar to a given set. Understanding how such processes work, are crucial for understanding how we can detect synthetically generated images.

Mathematically speaking, we can understand a grayscaled image as a two-dimensional signal, where each element of the signal has an integer value ranging from 0 to 255. These elements are often referred to as pixels. Elements that are assigned a value of 0, appear as black and elements that are assigned a value of 255 appear as white. Of course, most images on the internet are not grayscale, but rather have colours. To represent such a coloured image, every pixel itself has to be able to represent colours. This can be achieved in multiple ways, but for our purposes we will only consider the RGB-approach, where each pixel has a of red, green and blue pixel-value, each ranging from 0 to 255. The colour of the pixel is determined by the combination and intensity of these three primary colors.

First of, we will note that a grayscaled image of size n by n can be seen as an element of $\mathbb{R}^{nxn}$. Thus a set of objects X can be seen as a subset of $X \subset \mathbb{R}^{nxn}$. We can assume that this set is generated by some probability distribution μ, but then we are tacitly assuming that μ is absolutely continuous, which needs not to be the case. It is possible that even though data exists in a high-dimensional space, that the actual dimension space might lie on a manifold in a low-dimensional subspace. For now, we will continue without worrying about this assumption. If we were to have access to this μ, we could generate new objects by simply sampling from it.

However, in reality, having direct access to such a μ is not possible: We only have access to a set of samples from μ, X. We could try and approximate μ by finding some approximate probability distribution ν which is close to μ. If ν is sufficiently close to μ, we can sample from ν and find new objects that might be similar enough lookinh to X.

Synthetic Image Generators try to find efficient and good ways of finding such a probability distribution ν. More traditional parametric methods which rely on assuming ν comes from a certain family of probability distributions e.g. the Gaussian distribution with parameters $N(m, I)$ where we try to fit the parameters so that the likelihood of observing the data under the probability distribution is maximized, are less suited for estimation of ν, as we are very limited in the possible distributions we can choose. Additionally, due to the high-dimensionality of the state space, we would need a very large amount of data to get a good estimate of such distributions.

Let us assume γ is a probability distribution that is defined on $\mathbb{R}^d$, where d does not need to be the same as n^2 and such that we can easily sample from γ. E.g. γ can be distributed standard normally $N(0, I_d)$. Synthetic Image Generators aim to find a function $G : \mathbb{R}^d \to \mathbb{R}^{nxn}$ such that if a random variable $z \in \mathbb{R}^d$ has distribution γ, then $G(z)$ has distribution μ. If we could find such a distribution, our challenge of sampling from μ, which might be a highly complex distribution, reduces to sampling from γ, which in general is tractable to sample from, especially if γ is chosen standard normally.

The distribution of $G(z)$ is $\gamma \circ G^{-1}$, where G^{-1} sends subsets of $\mathbb{R}^{nxn}$ to subsets of $\mathbb{R}^d$. Ideally, we would have that $\gamma \circ G^{-1} = \mu$, but we could also try and find G such that $\gamma \circ G^{-1}$ is approximately equal to μ. However, the astute reader might have noticed that we do not know μ exactly, rather we have a set of samples distributed according to μ. Thus, given G, it is still not trivial how we could quantify how well this G performs in terms of creating similar objects.

The difference in Synthetic Image Generators comes exactly from how they go about answering this question: What does it mean for two objects to be similar? We will look into how the Generative Adversarial Network (GAN) and Diffusion Model (DM) try to quantify this notion of similarity and how these can be used to create models that can generate objects which are similar to our real objects of interest

2.1. Generative Adversarial Networks

GANs rely on the premise that competition causes improvement and try to answer the question of similarity by leveraging an adversarial game. This is done by introducing a discriminator, $D(x)$, which is a function from $\mathbb{R}^{nxn}$ to $[0,1]$, and represents the probability that the given image x, is synthetically generated by the generator, $G(z)$. The goal of the discriminator is to discriminate well between real objects, elements of X, and synthetically generated objects, elements of $G(z)$. Whilst the goal of the generator is to generate objects $G(z)$ such that the discriminator classifies them as real.

At the beginning, the generator is not performing very well and the discriminator will have an easy time classifying the objects correctly. The generator will learn from each mistake that it makes and adjust itself a little bit to perform better the next time it is in a similar scenario. However, if the generator performs too well, and the discriminator makes a mistake, then the discriminator will learn from that mistake and adjust itself a little bit.

This process of adaptively improving yourself as mistakes are made, continues until an equilibrium is reached. When the discriminator $D(x)$ does no better than random guessing, we expect that the generator is doing quite well in generating samples and thus we expect $\gamma \circ G^{-1} \approx \mu$.

The observant reader might have noticed that we explicitly need both $D(x)$ and $G(z)$ to be able to adjust given that a mistake has been made. Additionally, we would want that both the functions are able to express complex relations. Both of these reasons make the choice of a deep neural network (DNN) very appropriate, Not only is a DNN able to learn from its mistakes by means of applying gradient-based updates on the weights, but a DNN is also able to act as a universal function approximator due to the universal approximation theorem [20]. This means that a DNN, which needs to have atleast one hidden layer, can approximate any continuous function in theory, and therefore also our functions of interest, $D(x)$ and $G(z)$.

We can rewrite the discriminator and generator as $D_\phi(x)$ and $G_\theta(z)$ respectively. We do this as to explicitly state that the parameters of the discriminator DNN is the vector ϕ and similarly that those of the generator, is the vector θ. Our goal is to update θ in such a way that $\gamma \circ G_\theta^{-1}$ becomes very close to μ.

2.1.1. GAN Objective

It is not clear how we could achieve our goal of finding a generator G such that $\gamma \circ G_\theta^{-1}$ becomes very close to μ. We will firstly introduce some general theory that is not DNN specific, but we will then just use this in the DNN setting.

One approach to find a generator G which can generate realistic samples is by utilizing the following loss-function, introduced by Goodfellow et al. [15]:

$$V(D,G) := \mathbb{E}_{x\sim\mu}[\log D(x)] + \mathbb{E}_{z\sim\gamma}[\log(1-D(G(z)))], \tag{2.1}$$

for which the solution to the minimax problem, $\min_G \max_D V(D,G)$ is shown to be exactly what we are looking for, if we assume that the distributions have densities and that $d \geq n^2$ [1]. That is: Assuming that μ has density $p(x)$ and $\gamma \circ G^{-1}$ has density $q(x)$, then the minimax solution will be achieved only if $p(x) = q(x)$. Meaning that we have exactly what we wanted: After optimization, we will have a generator for which the distribution of generated objects is exactly equal to the distribution of our objects of interest. Goodfellow et al. [15] have not shown that these conclusions hold when we opt for using DNNs for both the classifier and the generator, but it is reasonable to expect good enough results.

[1]When $d < n^2$, there is a similar result which holds μ-almost surely.

2.1.2. Deconvolution and checkerboard-artifacts

In general, the dimensionality d of our easy to sample from distribution γ, is a lot smaller than the dimensionality of the objects we are generating. E.g. the images generated by the GAN in Figure 4.2 were created by a using $d = 100$. To transform such a sample to a final image of dimensionality 1024-by-1024, we will need to apply some operator that transforms our data sample from $\mathbb{R}^{\mathbb{d}}$ to $\mathbb{R}^{nxn}$. Such operators that transform an element from a space with less dimensionality to a space with higher dimensionality, are called upsampling operators.

One natural approach to do this, is by using deconvolutions. To explain what a deconvolution does, we will consider Figure 2.1, where a one dimensional signal deconvolution is shown, with a stride of two and a deconvolution filter of length three. A stride of two means that we only apply the deconvolution on every other element. Note that the value of every other element, shown as color intensity is the same, and thus we would expect a deconvolved signal to be constant in its value, as the assumed deconvolution filter here is uniform.

The deconvolution shown in Figure 2.1 uses every other element of the original, top signal, to paint three elements in the deconvolved signal. Since the stride is two and the filter length is three, this causes for overlap to happen and creates a checkerboard-esque pattern. For a two-dimensional signal and a two-dimensional deconvolution filter, similar behaviour can happen, as shown in Figure 2.2. Note that this behaviour is even more extreme, as there are some elements that have twice the size of the others. The consequence of such overlapping deconvolutions for images, can be seen in Figure 2.3, where a checkerboard pattern is clearly visible in the image.

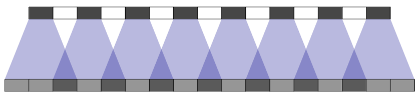


Figure 2.1: One dimensional deconvolution with stride two and filter-length 3 taken from [32].

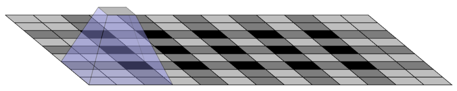


Figure 2.2: Two dimensional deconvolution with stride two and filter-length 3 taken from [32].

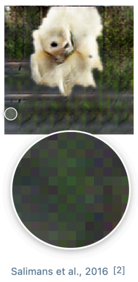


Figure 2.3: Checkerboard pattern occurring in GANs as consequence of deconvolutions taken from [32].

Of course, a neural network uses multiple layers of deconvolutions to create images from a lower level representation. One might think that these deconvolutions could cancel eachother out, but in practice these multiple layers often compound these artifacts [32].

This description of the deconvolution problem is a bit simplistic: In practice, neural networks learn the weights of these convolutions as part of their training. They might learn weights to evade such artifacts, but in practice they struggle to learn that avoid such artifacts [32].

One approach that has shown to be successful to have less artifacts, is to first apply a resizing operator on the original signal to make it larger. Such a resizing operator necessarily introduces pixels for which there is no known value. To fill in the blank spaces, one can use interpolation techniques such as nearest neighbour, bilinear interpolation or binomial interpolation. Afterwards a regular convolution operator can be applied. This approach causes the checkerboard-esque pattern that appears in Figure 2.3 to disappear. We will see in Chapter 4 that even though artifacts are less visible using this approach, they can still be found.

Note that this behaviour is not unique to GANs, rather this is an inherent property of the deconvolution operator. Odena et al. show this by considering images generated by a GAN before any prior training, with different upsampling operators used in the generator. This can be seen in figure Figure 2.4, where we see samples from generators that have not been trained. We would expect these samples to be some noise, but rather we see that there is a clear pattern visible if deconvolutions are used. Moreover, we see that these patterns are not visible if the resize-convolution approach is used.

Another approach that sounds reasonable, but would not work in practice, is to set $d = n^2$. By doing

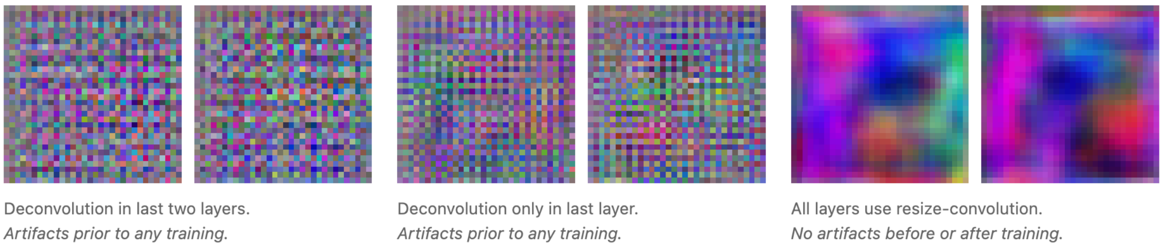

Deconvolution in last two layers.
Artifacts prior to any training.
Deconvolution only in last layer.
Artifacts prior to any training.
All layers use resize-convolution.
No artifacts before or after training.

Figure 2.4: Samples from generators using different upsampling operators without any training, taken from [32].

this, there is no need to upsample. However, this makes the problem intractable and therefore is not a good idea.

In conclusion, Generative Adversarial Networks are synthetic image generators that by design need to use upsampling operators which can introduce artefacts. These artefact of course are not visible in real images and thus we can be interesting for usage in Synthetic Image Detection.

2.2. Diffusion Models

Diffusion Models are another type of Synthetic Image Generators. As opposed to GANs, which were trying to implicitly learn a model by focusing on the sampling process rather than the data, DMs are closer to more regular likelihood-based models, which try to learn a model by maximizing the likelihood of the observations.

There is a natural trade-off between GANs and regular likelihood-based models. Likelihood-based models require the likelihood to be computed, which can be intractable in high-dimensional settings, because of the normalizing constant. One could try to find a likelihood-based model, but to make that tractable, we would have to choose very simple models, which might not be suited to capture the complexity of the true data generating distribution μ.

GANs, which are able to approximate way more complex data generating distribution, have the natural drawback that they can be very unstable. One example of this instability can be found in the occurrence of mode collapse, where all generated samples look nearly identical. This can happen since we train the generator to be able to fool the discriminator, but not necessarily to do this in a manner so that the samples it uses are as diverse as the real dataset.

Diffusion Models aim to combine the best of both worlds, by using a likelihood-based model, which does not need to explicitly access the normalizing constant to compute the likelihood. Thus it suffers less from the problems of implicit generative models, but it is tractable.

We will work towards understanding how Diffusion Models work. To this end, we will firstly consider the Evidence Lower Bound, then we will consider Variational Autoencoders, after which we will look at Markovian Hierarchical Variational Autoencoders and finally we will look at Variational Diffusion Models.

2.2.1. Latent Variable Model

We could try to simplify the problem of directly approximating the full data generating distribution μ by leveraging some kind of structure that might be apparent in the data. Let us say for example that our given set of objects X consists out of images of two types of alpacas: The first type is alpacas with white fur and the second type is alpacas with black fur. The images of the white alpacas will have, in general, higher pixel values than the images of black alpacas, since low pixel values in grayscale correspond to black pixels and high pixel values to white pixels.

We could thus simplify the problem of generating images of alpacas similar to the images in our dataset into the two problems of generating images of black alpacas and generating images of white alpacas. However, even if we have two perfect models for the latter two tasks, we would still need to have an estimate of what fraction of white and black alpacas there are in the data generating distribution μ.

Mathematically speaking, this model can be noted in the following form. Variable z denotes whether an alpaca is black or white in a binary fashion and variable x denotes the image taken of the alpaca. This relation is shown in Figure 2.5. Given the latent variable z, we can make more accurate estimations of what the conditional distribution $p(x \mid z)$ should look like. If additionally, we also have an estimate of the probability of the latent variable $p(z)$, then we can sample from $p(x)$ by sampling from $z_1 \sim p(z)$ and using that sample to sample $x_1 \sim p(x \mid z_1)$.

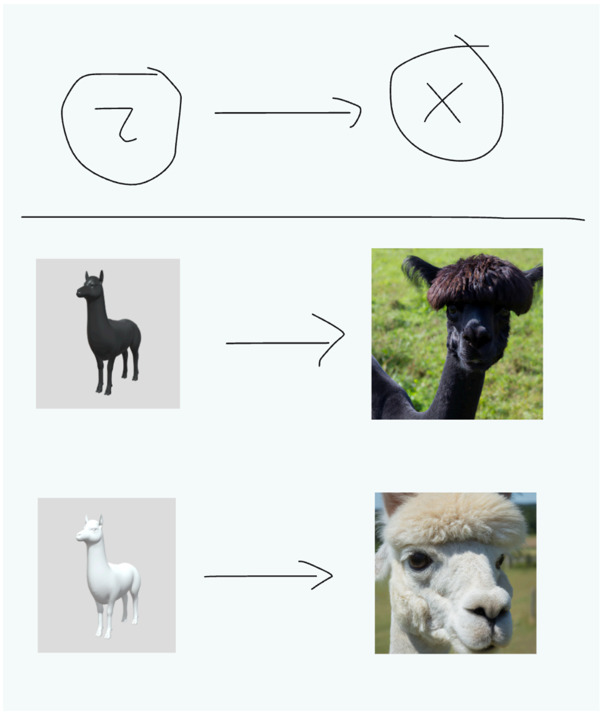


Figure 2.5: Top: Latent variable model, where z causes x. Below: An example of such a latent variable model, where the colour of the alpaca (black or white) causes the observation x to behave differently.

The full relation between z and x is given by the joint distribution $p(x, z)$, in our example this joint distribution would tell us how likely it it to see image x received by photographing alpaca with colour z. We can make use of the definition of conditional distributions to get the following identity for the joint distribution: $p(x, z) = p(x \mid z)p(z)$. This will proof to be useful later on.

It is worth considering that z can be of a different dimensionality than x. It can be useful for high-dimensional x to have a lower-dimensional z. This causes the model to learn a lower-dimensional representation of the data and might uncover semantically meaningful structures in the data.

2.2.2. Optimization of a Latent Variable Model

Suppose we had some other latent variable model for the alpaca images, e.g. at what time of the day the image was taken. How could we prefer the latent model that uses time of the day the image was taken over the colour-based model? One approach would be to find the model for which the likelihood of observing our alpacas image $X \in \mathbb{R}^{nxn}$ is the highest, since a priori we do not necessarily know whether observations other than ours are likely, so trying to find a model which can be described well by our observations seems like a good approach.

We would thus like to relate our model $p(x, z)$ to the likelihood $p(x)$ in some type of equation. To this end, we have two possibilities. Firstly, we can try to marginalize over z to get $p(x) = \int p(x, z)dz$. And secondly we can use the definition of conditional probabilities to find the following equation: $p(x) = \frac{p(x,z)}{p(z|x)}$.

The first approach is intractable if z is of a high dimensionality, as we would need to integrate over a high-dimensional space.[2] The second approach needs access to $p(z \mid x)$, which is in general not known. Thus it seems that our approaches to find the best latent variable model by simply optimizing the likelihood, are not sufficient.

Instead of finding a direct expression for the likelihood, we could also try to find some lower bound for the likelihood and compare the lower bounds for different latent variable models. Of course, if for

[2]For our alpaca example z is not of high-dimensionality, but if z were to be, let's say a full three-dimensional volume model of the alpaca, it would be.

two models we find that one of the two has a higher, thus better, lower bound for the likelihood, it does not necessarily mean that it also has a higher likelihood, since it's just a lower bound. However, we will see that using this weaker notion of one model being better than the other, will lead to tractable models which can be used in practice.

We will use the **Evidence Lower Bound** (ELBO) to this end. The ELBO is a lower bound for the log-likelihood, which does not depend on the, in general, unknown relation $p(z \mid x)$, but rather on a variational distribution $q_\theta(z \mid x)$, which tries to approximate $p(z \mid x)$ and is parametrized by parameters θ.

This notion of the variational distribution deserves a bit more attention. Since we do not have access to $p(z \mid x)$, we can try to approximate it by some function $q(z \mid x)$. Of course, we would like $q(z \mid x)$ to be close to $p(z \mid x)$ and for practical considerations, it would be useful if $q(z \mid x)$ was tractable. Thus we can choose q to be from a tractable family of distributions that allows for such efficient computations. Additionally, we can allow θ to be the parameters of the tractable family of distributions, such that we can learn from our data what is the best fitting distribution rather than guess ourselves. An example of such a family of variational distributions is e.g. a multivariate Gaussian distribution where we assume zero covariance between different dimensions i.e. a diagonal covariance matrix and allow for non-zero means, can be seen as such a family. The parameters here would be the means of every dimensions and the variance scales for every dimension. Note that there is this inherent trade-off between optimizing to use the best distribution and just choosing a distribution: The former is intractable but performs well, whereas the latter is tractable, but performs worse. Introduction of a variational distribution allows us to find a compromise between the two: We have some freedom in optimizing, whilst keeping things tractable.

Theorem 2.2.1. [Evidence Lower Bound (ELBO)] Given observations x generated by joint distribution $p(x, z)$ and variational distribution $q_\phi(z \mid x)$ which is parametrized by ϕ, we can lower bound the log-likelihood of x:

$$\log p(x) \geq \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z \mid x)}\right], \tag{2.2}$$

where the RHS of Equation 2.2 is called the evidence lower bound (ELBO).

Proof. First we will make use of the fact that $\int q_\phi(z \mid x) dz = 1$, since q_ϕ is a distribution.

$$\begin{aligned}
\log p(x) &= \log p(x) \int q_\phi(z \mid x) dz \\
&= \int q_\phi(z \mid x) \log p(x) dz \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log p(x)\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{p(z \mid x)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z \mid x)}\right] + \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{q_\phi(x,z)}{p(z \mid x)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z \mid x)}\right] + D_{KL}\left(q_\phi(z \mid x) \parallel p(z \mid x)\right).
\end{aligned}$$

Note that KL-divergence is non-negative and thus we can make the final needed step.

$$\log p(x) \geq \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z \mid x)}\right]$$

□

Theorem 2.2.1 shows us that a better ELBO can potentially lead to a higher log-likelihood. Thus if we are to choose between different models with two different values for the ELBO, we could prefer the one with the higher ELBO, assuming that this could mean that the real log-likelihood might also be higher. But note that does not necessarily need to be the case: The log-likelihood is exactly equal to the KL-divergence plus the ELBO term as can be see in Figure 2.6. If we are to compare the ELBO for two different variational distributions, q_ϕ and q_θ, it might very well be that the ELBO is higher for q_θ, but that the KL-divergence is sufficiently low for q_θ, that the sum of KL-divergence + ELBO for q_ϕ is still higher, thus yielding a higher log-likelihood. When using the ELBO as a proxy-objective, we accept the fact that it might not give us the best model, but in turn it does give us a tractable manner of optimizing the likelihood.

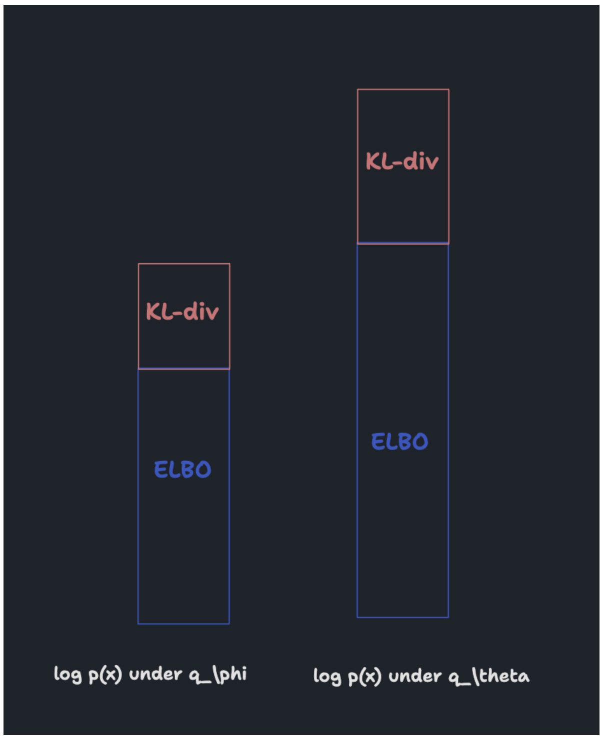


Figure 2.6: The ELBO is a lower bound on the log-likelihood of the data. The ELBO is in terms of a variational distribution $q_\phi(z \mid x)$, which is parametrized by ϕ. Choosing a variational distribution with larger ELBO can mean that the log-likelihood of the implicit model is higher.

2.3. Variational Autoencoder

An example of a generative model that uses the ELBO as a proxy-objective is the Variational Autoencoder (VAE). We can once again assume the setting of a latent variable model, where we have a latent variable z, which influences our observation x in some way. If we can simultaneously learn how an element from the object space is represented in the latent space and vice versa, then we can make use of that to generate objects.

In our previous example of the coloured alpacas, we have touched upon using a latent variable model for generating objects, but we needed to already assume some kind of model for that. Using the recipe as described by a VAE, removes the need for establishing such a model, rather it learns from the data what that latent variable structure might look like. This might thus help in cases where a high-dimensional latent variable structure is a good fit and cannot easily be designed by a human, but can by be learned by means of relying on the data.

Figure 2.7 shows a high-level interpretation of a VAE. We try to learn the relation between latent variable z and x in both directions, and parameterize both of the models. The distribution $q_\phi(z \mid x)$ models how the observed variable x influences the latent variable z, and $p_\theta(x \mid z)$ models how the latent variable influences the observable variable.

By learning $q_\phi(z \mid x)$, we can understand what our observed object x could look like in the latent space. Moreover, we can then ”check” how well our model performs by using $p_\theta(x \mid z)$ on this latent representation and look at how close it is to the original object.

In VAEs, it is often assumed that the dimension of the latent space is less than the dimension of the object space. For this reason $q_\phi(z \mid x)$ can be referred to as the encoder, as it tries to encode the

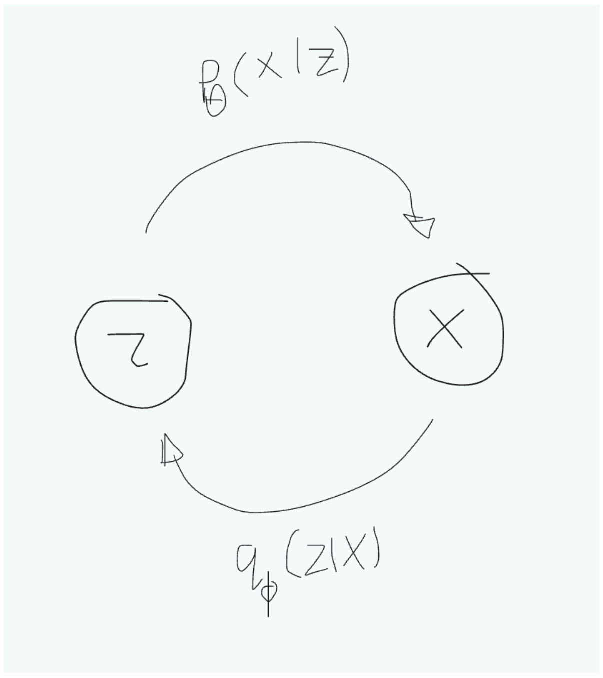


Figure 2.7: High-level structure of how a VAE works. There is a latent variable z, and the observed variable x. We wish to learn $q_\phi(z \mid x)$ a model for how the observed variable x influences the latent variable z, and we wish to learn $p_\theta(x \mid z)$, a model for how the latent variable influences the observable variable.

input x into a latent representation z, which is of lower dimensionality. Similarly, $p_\theta(x \mid z)$ is referred to as the decoder tries to decode the latent representation z into an output x. This is why the variational autoencoder is also called an autoencoder.

To better understand what the VAE does, we can further analyse the ELBO, over which the VAE optimizies. The ELBO can be broken down into two terms, a reconstruction term and a prior matching term.

Lemma 2.3.1. Given a latent variable model $z \to x$, a variational distribution $q_\phi(z \mid x)$, and a generative model $p_\theta(x \mid z)$, the ELBO can be written as:

$$\mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z \mid x)}\right] = \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x \mid z)\right] - D_{KL}(q_\phi(z \mid x) \parallel p(z)), \tag{2.3}$$

where $p(z)$ is the prior distribution over the latent space.

Proof. We can make use of the fact that multiplying by $\frac{p(z)}{p(z)}$ can always be done, note that this does mean that $p(z)$ should not be zero, but this assumption is almost always met.

$$\begin{aligned}
\mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z \mid x)}\right] &= \mathbb{E}_{q_\phi(z|x)}\left[\log(\frac{p(x,z)}{q_\phi(z \mid x)} \cdot \frac{p(z)}{p(z)})\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{p(z)} - \log \frac{q_\phi(z \mid x)}{p(z)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{p(z)}\right] - \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{q_\phi(z \mid x)}{p(z)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log p(x \mid z)\right] - D_{KL}(q_\phi(z \mid x) \parallel p(z)).
\end{aligned}$$

In practice, we will use $p_\theta(x \mid z)$ rather than the true encoder $p(x \mid z)$, thus this becomes an approximation.

$$\mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(x,z)}{q_\phi(z \mid x)}\right] \approxeq \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x \mid z)\right] - D_{KL}(q_\phi(z \mid x) \parallel p(z)).$$

□

Equation 2.3 is composed of two terms: the reconstruction term and the prior matching term respectively. The reconstruction term measures the quality of the encoder and decoder networks on the input data. It takes the expectation over the encoder distribution given the input data, which represents the possible latent representations that could be generated for the input. The reconstruction term is then a weighted average of the log-likelihood of the decoder given these latent representations. This measure of the quality of the encoder and decoder networks encourages the model to produce outputs that are similar to the input data. The reconstruction term formalizes the earlier discussed notion of ”checking” how well our model performs by encoding and decoding our objects of interest and looking at how close the encoded-decoded representation of the objects are to the original objects they came from.

The prior-matching term can be understood as a penalty for using a $q_\theta(z \mid x)$ which is too different from $p(z)$, the prior distribution of the latent space. Because of this term, during optimization we are explicitly introducing a trade-off between having a $q_\theta(z \mid x)$ which might yield us a better reconstruction term, but that is very different from the prior and having a worse reconstruction, but one that is more similar to the prior in terms of KL-divergence. This can be seen as some kind of **regularization**, where we can introduce prior knowledge by choosing $p(z)$ in a certain way.

One defining characteristic for the variational autoencoder, is that assumptions are made on both the prior distribution $p(z)$ and the variational distribution $q_\phi(z \mid x)$. The prior distribution $p(z)$ is assumed to be a standard normal distribution, and the variational distribution $q_\phi(z \mid x)$ is assumed to be a diagonal Gaussian distribution. Let's assume the latent space has dimension m, then:

$$p(z) = \mathcal{N}(z \mid 0, I_m), \quad q_\phi(z \mid x) = \mathcal{N}(z \mid \mu_\phi(x), \sigma_\phi(x)^2 I_m).$$

Note that the prior-matching is, by assumption of these distributions for both $p(z)$ and $q_\phi(z \mid x)$, the KL-divergence between two multivariate Gaussians with scaled identity variance, for which we know there is a closed-form solution by [24].

Lemma 2.3.2. Assuming $p(z) = \mathcal{N}(z \mid 0, I_m)$ and $q_\phi(z \mid x) = \mathcal{N}(z \mid \mu_\phi(x), \sigma_\phi(x)^2 I_m)$, then:

$$-D_{KL}(q_\phi(z \mid x) \parallel p(z)) = \frac{1}{2}\sum_{j=1}^{m}\left(1 + \log(\left[\sigma_\phi(x)\right]_j^2) - \left[\mu_\phi(x)\right]_j^2 - \left[\sigma_\phi(x)\right]_j^2\right).$$

Proof. The proof can be found in [24]. □

2.3.1. VAE in practice

It is not entirely clear how we could go about training a VAE. In this section we will spend some time to explain how it does that. We also place special attention on understanding what exactly our network looks like, what our loss function is and how the chosen network relates to the loss.

We learn the parameters of the networks, θ and ϕ, by jointly optimizing the parameters of the network. We will leverage the fact that we have real objects of which we have a very good idea of what we should find after having encoded an object and afterwards decoding that encoding-representation, namely, the object we had at the very start.

To have a better of grasp what this would mean, we have included Figure 2.8. It can be seen that the encoder-network transforms an input image into a mean $\mu_\phi(x)$ and $\ln \sigma_\phi(x)$, where the logarithm is used for the variance because this removes the need for a non-negative restriction in our networks: Rather, we can just take the exponent of $\ln \sigma_\phi(x)$, and we will automatically have a non-negative value.

We should place special attention to the fact that the encoder $q_\phi(z \mid x)$ and the encoder network are not the same. Rather, the encoder network learns the parameters which characterize the encoder. That

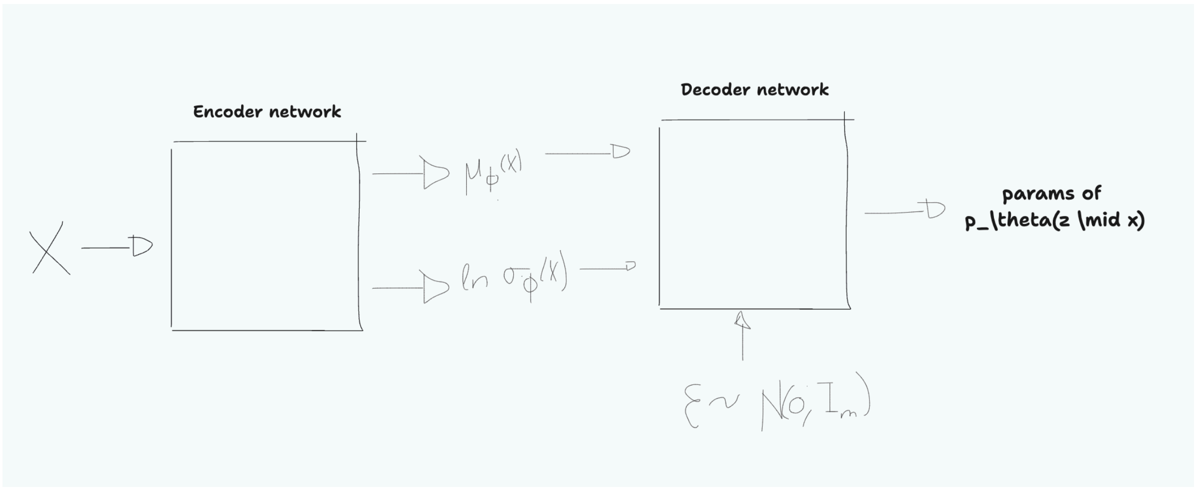


Figure 2.8: Network structure of a general VAE, where we assume the encoder to be diagonal Gaussian. Additionally, we make use of the reparametrization trick to allow for differentiable gradients in the network. Lastly, the decoder networks output have not been note

is also the reason why in the network structure in Figure 2.8 we have an additional normally distributed $\epsilon \sim N(0, I_m)$ as input for the decoder network. Since $\mu_\phi(x) + \sigma_\phi(x)\epsilon$ is distributed exactly as $q_\phi(z \mid x)$, we can use this as an input to our decoder network.

However, notice that the decoder network is also not the same as the decoder, $p_\theta(x \mid z)$, rather it should put out parameters of the distribution $p_\theta(x \mid z)$, such that the reconstruction term of the ELBO can be calculated. In practice, often the distribution of $p_\theta(x \mid z)$ is assumed to be Gaussian, such that the $\log p_\theta(x \mid z)$ simplifies to $||x - \hat{x}_\theta||^2$ disregarding constants, where $\hat{x}_\theta$ is the output of the decoder network.

If we follow the convention of assuming the distribution to be Gaussian such that the log-term simplifies, our ELBO becomes the following:

$$\begin{aligned} \text{ELBO} &= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x \mid z)\right] - D_{KL}(q_\phi(z \mid x) \parallel p(z)) \\ &\approx \sum_{k=1}^{n} ||x^{(i)} - \hat{x}_\theta^{(i)}||^2 + \frac{1}{2}\sum_{j=1}^{m}\left(1 + \log([\sigma_\phi(x)]_j^2) - [\mu_\phi(x)]_j^2 - [\sigma_\phi(x)]_j^2\right), \end{aligned}$$

where we use Monte Carlo samples to remove the dependence on $\mathbb{E}_{q_\phi(z|x)}$ and Lemma 2.3.2 to find a close-form representation of the KL-divergence. Note that this expression allows for direct parameter updates of both the decoder and encoder network, as all terms are differentiable.

2.4. Markovian Hierarchical Variational Autoencoders

A Hierarchical Variational Autoencoder (HVAE) is similar to a VAE, but it has the additional property of having multiple latent states, which exist in a hierarchical structure. For illustrative purposes, we will give an example of a HVAE. Let's say that we are once again the in the setting where our objects of interest are images of alpacas. There might be multiple latent variables that influence our images: e.g. the colour of the alpaca, the type of camera that was used or what kind of plants are visible. Note that what kind of plants are visible, itself depends on other latent variables: In which country the image was taken, at what time of year the image was taken, since in different countries and different seasons, different plants are visible. Note that this latent variable model has different levels, which influence the object of interest via a variable on a lower level.

We will consider the case of Markovian Hierarchical Variational Autoencoders (MHVAEs), where we additionally set the latent variables to be Markovian. That is, assume there are T levels[3] of latent

[3]We assume that there is a single latent variable per level, but this is an accurate assumption, because any level with more than one latent variable per stage can be represented with a single latent variable.

variables, then the graphical representation looks as following: $z_T \to z_{T-1} \to \cdots \to z_1 \to x$. That is, given z_l for $l = j, \ldots, T$, the probability of observing z_{j-1}, the states of the latent variables higher in the hierarchy than z_j does not give us any new information. i.e. $p(z_{j-1}|z_T, z_{T-1}, \ldots, z_j) = p(z_{j-1}|z_j)$.

An example of a MHVAE could be where we once again assume x to be an image of an alpaca and z_t to be the colour of the fur of the alpaca at t time before the taking of the picture.[4] There are multiple levels, which are Markovian: Given the colour of an alpaca at times $T, T-1, \ldots, 1$ before taking the image, their fur at time of taking the image is only a function of the fur colour at time 1 before taking the image.

This Markovian property can be used to rewrite the joint distribution and the encoder in the following manner:

$$
\begin{aligned}
p(x, z_{1:T}) &= p(z_T)p_\theta(x|z_1)\prod_{t=2}^{T} p_\theta(z_{t-1} \mid z_t) \\
q_\phi(z_{1:T} \mid x) &= q_\phi(z_1 \mid x)\prod_{t=2}^{T} q_\phi(z_t \mid z_{t-1})
\end{aligned}
\tag{2.4}
$$

We can extend the ELBO for MHVAEs in the following manner

Lemma 2.4.1 (ELBO for MHVAE)**.** The ELBO for MHVAE is:

$$
\mathbb{E}_{q_\phi(z_{1:T}|x)}\left[\log \frac{p\left(x, z_{1:T}\right)}{q_\phi\left(z_{1:T} \mid x\right)}\right],
\tag{2.5}
$$

or equivalently:

$$
\mathbb{E}_{q_\phi(z_{1:T}|x)}\left[\log \frac{p\left(z_T\right) p_\theta\left(x \mid z_1\right) \prod_{t=2}^{T} p_\theta\left(z_{t-1} \mid z_t\right)}{q_\phi\left(z_1 \mid x\right) \prod_{t=2}^{T} q_\phi\left(z_t \mid z_{t-1}\right)}\right].
\tag{2.6}
$$

Proof. We will first start off by showing that the ELBO is equal to Equation (2.5).

$$
\begin{aligned}
\log p(x) &= \log p(x) \int q_\phi(z_{1:T} \mid x) dz_{1:T} \\
&= \int \log p(x) q_\phi(z_{1:T} \mid x) dz_{1:T} \\
&= \mathbb{E}_{q_\phi(z_{1:T}|x)}\left[\log p(x)\right] \\
&= \mathbb{E}_{q_\phi(z_{1:T}|x)}\left[\log \frac{p(x, z_{1:T})}{p(z_{1:T} \mid x)}\right] \\
&= \mathbb{E}_{q_\phi(z_{1:T}|x)}\left[\log \frac{p(x, z_{1:T}) q_\phi(z_{1:T} \mid x)}{p(z_{1:T} \mid x) q_\phi(z_{1:T} \mid x)}\right] \\
&= \mathbb{E}_{q_\phi(z_{1:T}|x)}\left[\log \frac{p(x, z_{1:T})}{q_\phi(z_{1:T} \mid x)}\right] + \mathbb{E}_{q_\phi(z_{1:T}|x)}\left[\log \frac{q_\phi(z_{1:T} \mid x)}{p(z_{1:T} \mid x)}\right] \\
&= \mathbb{E}_{q_\phi(z_{1:T}|x)}\left[\log \frac{p(x, z_{1:T})}{q_\phi(z_{1:T} \mid x)}\right] + D_{KL}(q_\phi(z_{1:T} \mid x)||p(z_{1:T} \mid x)) \\
&\geq \mathbb{E}_{q_\phi(z_{1:T}|x)}\left[\log \frac{p(x, z_{1:T})}{q_\phi(z_{1:T} \mid x)}\right].
\end{aligned}
$$

Now we can show Equation (2.6) by direct application of Equation (2.4). □

2.5. (Variatonal) Diffusion Models

(Variational) Diffusion Models ,(V)DMs, are Markovian Hierarchical Variational Autoencoders, with a number of additional constraints, which we will discuss in this section. Additionally, we will use x_t instead of z_t, this can be done because the latent dimension is the same as the data dimension.

[4]Alpacas can have different color fur over the duration of the course of their life.

The generic pipeline of a DM exists out of three parts: the forward process, the reverse process and some sampling procedure. Both the forward process and the reverse process can be understood as Markov chains over multiple timesteps. Both processes are implemented in parallel, where the forward process concerns itself with steadily adding noise, whereas the reverse process concerns itself with steadily removing noise.

2.5.1. Forward Process

To better explain what the exact goal of the forward process is, we will consider the situation where our samples of interest $x_1, ..., x_n$ are one-dimensional i.e. $\in \mathbb{R}$, so that we can visualize the different distributions over the different timesteps. Since we assume that our latent dimensions is the same over all timesteps, we know that the distribution at any time step is one-dimensional.

The forward-process of one-dimensional samples has been visualized in Figure 2.9. The leftmost circle corresponds to the unknown data distribution, that is the distribution that has given us $x_1, ..., x_n$. A single forward transition is described by $q(x_t \mid x_{t-1})$, which we assume to be a Gaussian distribution centered around[5] the output of the previous timestep with some noise ϵ_t.

We see that between two sequential timesteps in the chain, the pertubation is quite small, but over time these small pertubations add up and can cause the final distribution to be quite different from the original distribution. Moreover, in the forward process of a VDM, we wish that the rightmost distribution is standard Gaussian. We will see that this is for sampling purposes.

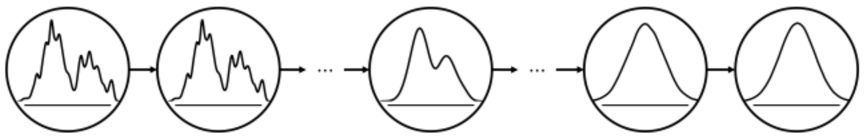


Figure 2.9: One-dimensional example of what a forward process could look like. The leftmost circle represents the initial data distribution, $p(x_0)$. At every timestep in the Markov Chain, the original distribution is somewhat perturbed, until it becomes completely Gaussian noise at the final rightmost timesteps. Every circle represents the distribution of the perturbed distribution at a higher timestep. This figure comes from [4].

Additionally, note that the transitions between timesteps in Figure 2.9 are not parameterized by any parameter. This is because these transitions are not learned, rather in VDMs, as mentioned, assume that every step of the forward process is a linear Gaussian model, based on the last step.

One might wonder why we bother using so many time steps: Couldn't we just use one transition from the data distribution to a normal distribution, similar to a VAE? We will see that the transitions in the forward process are chosen in a very specific manner, such that we can learn a reverse process which allows us to effectively model the original data distribution.

For scenarios where the objects of interest have higher dimensionality than one, e.g. images, the forward process generalizes quite easily. The transitions are linear Gaussian models, but in addition also multivariate and the final distribution of x_T is still Gaussian, but also multivariate. Additionally, the pixels are assumed to be uncorrelated.

2.5.2. Reverse Process

To better understand the reverse process, we will also consider the case where the objects of interest are one-dimensional. Since the time-reversal of a Markov Chain is also a Markov Chain [4], the reversal process is also a Markov Chain.

The reverse process has been visualized in Figure 2.10. where we see that we are starting off with some Gaussian distribution and the transitions $p_\theta(x_{t-1} \mid x_t)$ with the hope of finding parameters θ, such that in the end our estimated initial distribution is similar to the real initial distribution $p_\theta(x_0) \sim p(x_0)$ [39].

The observant reader might notice that we have still to assume anything about the specific structure of the reverse transitions. If were to have large enough T and we were to use sufficiently small pertubations at each timestep, then the reverse process does not only exist, but it also has the same functional

[5]It is actually centered around $\sqrt{\alpha_t}x_{t-1}$, but α_t will be close to 1.

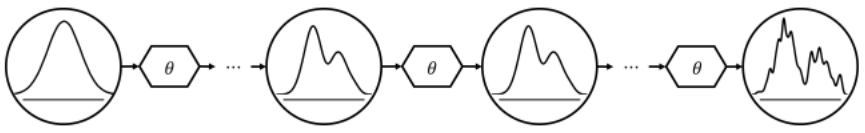


Figure 2.10: One-dimensional example of what a reverse process could look like. The leftmost circle represents a Gaussian distribution at time T and the rightmost distribution represents an estimate of what the initial distribution could look like. Every transition step is parameterized by θ, which tries to gradually denoise the sample from the previous timestep, so that the rightmost circle, at time 0, follows a distribution that is similar to the initial data distribution. This figure comes from [4].

form as the forward process [39]. In this case, this means that the reverse transitions $p(x_{t-1} \mid x_t)$ are Gaussian. Thus, we can reasonably assume that $p_\theta(x_{t-1} \mid x_t)$ is Gaussian for any t, with some mean $\mu_\theta(x_t, t)$ and variance $\Sigma_\theta(x_t, t)$.

Lastly, If we have optimized our model for the transitions $p_\theta(x_{t-1} \mid x_t)$, sampling a new object is as simple as sampling from $p(x_T)$ and recursively applying $p_\theta(x_{t-1} \mid x_t)$ until we have x_0, which will be our newly generated sample.

2.5.3. Optimizing the parameters for the reverse process

Now that we have an idea of what we are looking for, that is the parameters $\mu_\theta(x_t, t)$ and variance $\Sigma_\theta(x_t, t)$ that characterize the reverse process, we can consider how we can go about optimizing these parameters. One approach would be to optimize the likelihood, $p_\theta(X)$, where X contains all of our data. However, as mentioned earlier, we cannot tractably maximize this likelihood. What we can do, however, is interpret the VDM as a special type of MHVAE, so that we can use the ELBO as a proxy-objective. This allows us to benefit of the aforementioned characteristics of the ELBO: Tractable proxy-optimization of the likelihood.

In the upcoming section we will dive into how this can actually be done: How can the ELBO be used specifically for a VDM. To this end, we will need to properly define what a VDM is and we will consider some technical specifications.

Definition 2.5.1 (Variational Diffusion Model)**.** Given noise schedule α_t, a Variational Diffusion Model is a MHVAE with the following three additional constraints:

1. The latent dimension is the same as the data dimension.
2. The latent encoder is assumed to be a linear Gaussian model. It is a Gaussian distribution centered around the output of the previous timestep, more specificly: $q(x_t|x_{t-1}) = N(x_t; \sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)I)$
3. The distribution of the latent variable at timestep T, x_T is standard Gaussian.

Note how the Gaussian encoder mentioned in item 2 is parametrized by mean $\mu_t(x_t) = \sqrt{\alpha_t}x_{t-1}$ and variance $\Sigma_t(x_t) = (1-\alpha_t)I$. Note that the Gaussian encoder is dependent on the noise schedule α_t. We can choose to set α_t beforehand, as a set of hyperparameters [19], or we can choose to learn them from the data [22]. For simplicity, let us assume that we set this beforehand. Due to item 3 of Definition 2.5.1 α_t needs to be chosen in such a way that $p(x_T)$ is standard Gaussian. As shown in the work by Scotta and Messina, [39], if α_t is chosen in such a way, then the forward process admits a reverse process that has the same functional form as the forward process. For reference: Ho et al. [19] opt to let α_t linearly decrease from $\alpha_1 = 1 - 10^{-4}$ to $\alpha_T = 0.98$ with $T = 1000$. Note that thus the encoder is not dependent on any learnable parameters and we do not need to learn anything for it.

We will derive the ELBO for VDMs and show that this exists out of a number of components which have interesting interpretations. To this end, we will use the fact that a VDM is a MHVAE and make use of the identity from Lemma 2.4.1. We need to make some small adjustments: Instead of $z_1, \ldots, z_T$, we will use $x_1, \ldots, x_T$ and x as x_0, because of the fact that the dimensionality of the latent variables is the same as the dimensionality of the our objects of interest.

The the encoder, q, also will not be parameterized by ϕ, i.e. q will be used instead of q_ϕ. This is because of the fact that there is nothing to learn: q is known fully in advance.

Theorem 2.5.1 (ELBO for VDM). The ELBO for a VDM is equal to:

$$\mathbb{E}_{q(x_1|x_0)}\left[\log p_\theta(x_0 \mid x_1)\right] - D_{KL}(q(x_T|x_0) \parallel p(x_T)) - \sum_{t=2}^{T} \mathbb{E}_{q(x_t|x_0)}\left[D_{KL}(q(x_{t-1} \mid x_t, x_0) \parallel p_\theta(x_{t-1} \mid x_t)\right],$$

where we refer to the first term as the reconstruction term, the second term as the prior-matching term and all the terms inside the final sum as the denoising matching terms.

Proof. Note that a VDM is a special case of a MHVAE, so we can use the identity from lemma 2.4.1.

$$\begin{aligned}
\log p(x) &\geq \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t)}{\prod_{t=1}^{T} q(x_t \mid x_{t-1})}\right] = \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p(x_T)p_\theta(x_0 \mid x_1)\prod_{t=2}^{T} p_\theta(x_{t-1} \mid x_t)}{q(x_1 \mid x_0)\prod_{t=2}^{T} q(x_t \mid x_{t-1})}\right] \\
&= \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p(x_T)p_\theta(x_0 \mid x_1)}{q(x_1 \mid x_0)} + \log \prod_{t=2}^{T} \frac{p_\theta(x_{t-1} \mid x_t)}{q(x_t \mid x_{t-1})}\right] \\
&= \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p(x_T)p_\theta(x_0 \mid x_1)}{q(x_1 \mid x_0)} + \log \prod_{t=2}^{T} \frac{p_\theta(x_{t-1} \mid x_t)}{\frac{q(x_{t-1}|x_t, x_0)q(x_t|x_0)}{q(x_{t-1}|x_0)}}\right] \\
&= \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p(x_T)p_\theta(x_0 \mid x_1)}{q(x_1 \mid x_0)} + \sum_{t=2}^{T} \log \frac{p_\theta(x_{t-1} \mid x_t)}{q(x_{t-1} \mid x_t, x_0)} + \sum_{t=2}^{T} \log \frac{q(x_{t-1} \mid x_0)}{q(x_t \mid x_0)}\right].
\end{aligned}$$

Note that the rightmost term telescopes, since $\log \frac{a}{b} = \log a - \log b$ and we can write out.

$$\log p(x) \geq \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p(x_T)p_\theta(x_0 \mid x_1)}{q(x_1 \mid x_0)} + \sum_{t=2}^{T} \log \frac{p_\theta(x_{t-1} \mid x_t)}{q(x_{t-1} \mid x_t, x_0)} + \log \frac{q(x_1 \mid x_0)}{q(x_T \mid x_0)}\right].$$

We combine the first and final term, noting that in both the denominator and numerator have $q(x_T \mid x_0)$, thus those cancel out.

$$\begin{aligned}
\log p(x) &\geq \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p(x_T)p_\theta(x_0 \mid x_1)}{q(x_T \mid x_0)} + \sum_{t=2}^{T} \log \frac{p_\theta(x_{t-1} \mid x_t)}{q(x_{t-1} \mid x_t, x_0)}\right] \\
&= \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log p_\theta(x_0 \mid x_1)\right] + \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p(x_T)}{q(x_T \mid x_0)}\right] + \sum_{t=2}^{T} \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p_\theta(x_{t-1} \mid x_t)}{q(x_{t-1} \mid x_t, x_0)}\right] \\
&= \mathbb{E}_{q(x_1|x_0)}\left[\log p_\theta(x_0 \mid x_1)\right] - \mathbb{E}_{q(x_T|x_0)}\left[\log \frac{q(x_T \mid x_0)}{p(x_T)}\right] + \sum_{t=2}^{T} \mathbb{E}_{q(x_t, x_{t-1}|x_0)}\left[\log \frac{p_\theta(x_{t-1} \mid x_t)}{q(x_{t-1} \mid x_t, x_0)}\right] \\
&= \mathbb{E}_{q(x_1|x_0)}\left[\log p_\theta(x_0 \mid x_1)\right] - D_{KL}(q(x_T|x_0) \parallel p(x_T)) + \sum_{t=2}^{T} \int \frac{p_\theta(x_{t-1} \mid x_t)}{q(x_{t-1} \mid x_t, x_0)} q(x_t, x_{t-1}|x_0)dt \\
&= \mathbb{E}_{q(x_1|x_0)}\left[\log p_\theta(x_0 \mid x_1)\right] - D_{KL}(q(x_T|x_0) \parallel p(x_T)) + \sum_{t=2}^{T} \int \log \frac{p_\theta(x_{t-1} \mid x_t)}{q(x_{t-1} \mid x_t, x_0)} q(x_t, x_{t-1}|x_0)dx_t, x_{t-1}.
\end{aligned}$$

We use the fact that $q(x_t, x_{t-1} \mid x_0) = q(x_{t-1} \mid x_t, x_0)q(x_t \mid x_0)$ and that we can split the double integral over x_t, x_{t-1} into two integrals: one over x_t and one over x_{t-1}.

$$
\begin{aligned}
\log p(x) \geq\ & \mathbb{E}_{q(x_1|x_0)}[\log p_\theta(x_0 \mid x_1)] - D_{KL}(q(x_T|x_0) \parallel p(x_T)) \\
& - \sum_{t=2}^{T} \int\int \log \frac{q(x_{t-1} \mid x_t, x_0)}{p_\theta(x_{t-1} \mid x_t)} q(x_{t-1}|x_t, x_0) q(x_t|x_0) dx_{t-1} dx_t \\
=\ & \mathbb{E}_{q(x_1|x_0)}[\log p_\theta(x_0 \mid x_1)] - D_{KL}(q(x_T|x_0) \parallel p(x_T)) - \sum_{t=2}^{T} \mathbb{E}_{q(x_t|x_0)}[D_{KL}(q(x_{t-1} \mid x_t, x_0) \parallel p_\theta(x_{t-1} \mid x_t)].
\end{aligned}
$$

□

Theorem 2.5.1 shows that the ELBO for a VDM can be divided into three terms. The first term is the reconstruction term, similar to the reconstruction term we saw in VAEs.

The second term is the prior matching term. By our assumption on the prior in combination with item 3 of definition 2.5.1, this term is expected to be zero. Moreover, it has no trainable parameters and thus won't be of interest in any optimization.

The final set of terms, are called the denoising matching terms. These terms can be interpreted as penalties for when the difference between our learned decoder $p_\theta(x_{t-1} \mid x_t)$ and our ground-truth denoiser, $q(x_{t-1} \mid x_t, x_0)$, should be small. The ground-truth denoiser knows what the final image, x_0, looks like, so given x_t, it should be able to accurately find a distribution for x_{t-1}.

We will work towards showing how Theorem 2.5.1 can be used to train a model for $p_\theta(x_{t-1} \mid x_t)$. To this end, we will introduce a number of useful lemmas:

Lemma 2.5.1. $q(x_t \mid x_0)$ is distributed according to $N(\sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$, where $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$.

Proof. A proof can be found in [47]. □

Lemma 2.5.2. $q(x_{t-1} \mid x_t, x_0)$ is distributed according to $N(\mu_q(x_t, x_0), \sigma_q(t)I)$, where we use the shorthand notations:

$$\mu_q(x_t, x_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)x_0}{1-\bar{\alpha}_t} \tag{2.7}$$

$$\sigma_q(t) = \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} \tag{2.8}$$

Proof. A proof can be found in [29]. □

Lemma 2.5.3 (KL-divergence between two multivariate Gaussians with the scaled identity variance)**.** Assume $\Sigma_1 = c \cdot I$, where c is some scalar, then the KL-divergence of two Gaussian, $N_1 \sim N(\mu_1, \Sigma_1)$ and $N_2 \sim N(\mu_2, \Sigma_1)$ is:

$$D_{KL}(N_1 \parallel N_2) = \frac{1}{2c}\|\mu_2 - \mu_1\|_2^2$$

Proof. A proof can be found in [10]. □

Our goal is to maximize the log-likelihood, for which we use maximizing the ELBO as a proxy-objective. To maximize the ELBO, we will need to somehow deal with the KL divergence term. Note that we assume p_θ to be Gaussian, as the true reverse process will also be Gaussian. Since $q(x_{t-1} \mid x_t, x_0)$ is also Gaussian, and has set variance, which only depends on the α_t coefficients, we can further assume that p_θ has this exact variance. Therefore, the only thing we need to learn of p_θ, is the mean $\mu_\theta(x_t, t)$[6].

Using lemma 2.5.3, with for any t, $c = \sigma_q(t)$, $\mu_1 = \mu_q(x_t, x_0)$ and $\mu_2 = \mu_\theta(x_t, t)$, we see that the optimization comes down to choosing μ_θ to be close to μ_q with the difficulty laying in the fact that μ_q

[6]Note how we cannot condition on x_0, as $p_\theta(x_{t-1} \mid x_t)$ does not condition on x_0

does not have access to x_0. Noticing the exact form of $\mu_q(x_t, x_0)$ in eq. (2.7), we can choose $\mu_\theta(x_t, t)$ to be as similar as possible to $\mu_q(x_t, x_0)$:

$$\mu_\theta(x_t, t) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\hat{x}_\theta(x_t, t)}{1-\bar{\alpha}_t}, \tag{2.9}$$

Note that $p_\theta(x_{t-1} \mid x_t)$ conditions on x_t, so we can use it in the RHS of eq. (2.9), additionally, as the hyperparameters α_t are known, so we can also choose this as we wish. Note that with this choice of μ_θ, the KL-divergence simplifies to

$$D_{KL}\left[q(x_{t-1} \mid x_t, x_0 \parallel p_\theta(x_{t-1} \mid x_t)\right] = \frac{1}{2\sigma_q(t)}\frac{\bar{\alpha}_{t-1}(1-\alpha_t)^2}{(1-\bar{\alpha}_t)^2}\|x_0 - \hat{x}_\theta(x_t, t)\|_2^2. \tag{2.10}$$

Thus, optimizing the matching denoising terms comes down to estimating the original image x_0 given a noisy version of that image at time t. If we compare the LHS of eq. (2.10) to the RHS, we can see that the RHS is relatively straight forward to minimize for different values of θ.

Theorem 2.5.2. The denoising matching terms can be represented in three equivalent, ways. Firstly it can be represented in terms of the original image and an estimate of the original image given the noisified image at time t:

$$D_{KL}\left[q(x_{t-1} \mid x_t, x_0 \parallel p_\theta(x_{t-1} \mid x_t)\right] = \frac{1}{2\sigma_q(t)}\frac{\bar{\alpha}_{t-1}(1-\alpha_t)^2}{(1-\bar{\alpha}_t)^2}\|x_0 - \hat{x}_\theta(x_t, t)\|_2^2. \tag{2.11}$$

Secondly, the denoising matching terms can be represented in terms of trying to predict the source noise ϵ_0, which is the difference between x_t and x_0:

$$D_{KL}\left[q(x_{t-1} \mid x_t, x_0 \parallel p_\theta(x_{t-1} \mid x_t)\right] = \frac{1}{2\sigma_q(t)}\frac{(1-\alpha_t)^2}{(1-\bar{\alpha}_t)\alpha_t}\|\epsilon_0 - \hat{\epsilon}_\theta(x_t, t)\|_2^2. \tag{2.12}$$

Lastly, the denoising matching terms can be represented in terms of trying to learn the real score function $\nabla_{x_t} \log p(x_t)$, which tells us in what direction to move starting in x_t to get a higher log-likelihood:

$$D_{KL}\left[q(x_{t-1} \mid x_t, x_0 \parallel p_\theta(x_{t-1} \mid x_t)\right] = \frac{1}{2\sigma_q(t)}\frac{(1-\alpha_t)^2}{\alpha_t}\|\nabla_{x_t} \log p(x_t) - \hat{s}_\theta(x_t, t)\|_2^2. \tag{2.13}$$

Proof. For a proof of this theorem we refer to the work by Luo [29]. □

In Theorem 2.5.2 we can see that there are three different, but equivalent representations to optimize the denoising matching terms. Each of these representations have their own advantages, which the interested reader can find more on in [4]. For our purposes, it is especially interesting that the image-based approach, Equation (2.11), seems to be worse in reconstructing finer details [1].

Variational Diffusion Models often opt to optimize over the score-based representation shown in Equation (2.13). This choice allows reveals that there is a intrinsic connection between VDMs and another class of synthetic image generators, namely Score-based Generative Models [41]. This connection allows for a number of interesting properties. One of them being that they allow for understanding generalizations of VDMs with infinite number of timesteps, namely that this process becomes a stochastic differential equation.

3 Representations

Synthetic Image Generators are usually trained in such a way that the raw-pixel values produced correspond somehow to the training set we are using. This is most-straightforward for a GAN, where the loss of the generator is exactly decided by how well the discriminator performs on the generated image. This means that more abstract structures which are not easily apparent in the raw-pixel values, might show up in a different representation of the image.

Ideally, we would like to find a systematic difference between real and synthetic images in some domain. Since the images have been trained on the pixel-domain, it is more likely that we can find such a systematic mismatch when applying a transformation such as Fourier, wavelet or even wavelet-packet transform than when applying no such transformation.

To this end, we will start off by showing how images can be considered as discrete signals. Afterwards we will the Discrete Fourier Transform, which yields a frequency-based representation of an image. Considering localization challenges of the DFT, we will show that a transformation that takes both frequency and space information into account, such as wavelet-based approaches can be useful. Afterwards, we will work towards the one-dimensional Discrete Wavelet Transform (DWT), for which we will look at filters, filter banks and the continuous wavelet transform. Afterwards, we will consider how the one-dimensional DWT can be extended to two-dimensional signals. Finally, we will look at how images can be represented using wavelet-packet approaches.

3.1. Images as signals

To understand why it might be useful to represent an image under another representation, it is instructive to understand how an image can be interpreted as a function. A grayscale image can be seen as a two-dimensional function: It's a function that assigns a certain value to every pixel pair inside it's domain. Thus we could just plot an image as a three-dimensional surface, where the x and y axes correspond to the pixel location and where we use the z-axis to assign the corresponding grayscale pixel value, which ranges from 0 to 255[1]. For illustrative purposes, we have plotted such a representation in Figure 3.1.

Notice how in the middle plot of Figure 3.1, the surface seems to exists out of several different types of sections. Firstly, there is this large section which is all close to 1 and where there is not a lot of variation, corresponding to the sky in the original image. Secondly, the black sections, corresponding to the background have slightly more variation then the background, but this variation is quite gradual, the surface looks like a hilly landscape in these areas. Lastly, there are the areas that correspond with the fur of the alpaca, these areas have a large amount of variation in them as can be seen by the jittery surface area.

If we were to try and decompose this image with sine and cosine functions, then we would notice that we would need sine and cosine with certain frequencies such that these functions correspond to the previously mentioned three areas: For representing the sky, we would find that a function with a low frequency is very well suited to be used, whereas for the fur we would need a function with a rather high frequency to represent the jitteriness.

[1]We can normalize this to the range $0 - 1$ by simply dividing by 255

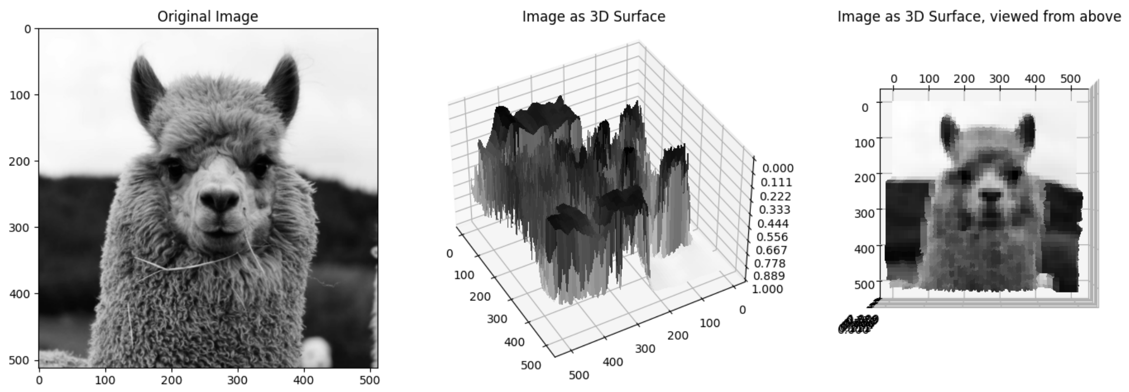


Figure 3.1: Grayscaled image of an alpaca and two different views of the image represented as a three-dimensional surface. Notice how the 3d-surface from above just looks like the image.

There are several reasons why such a decomposition might be useful. As you might have noticed, our alpaca image has a large area where the pixels are all 1, that is the sky. Thus if we were to save this image and we would need to save every single pixel value for that area, we would need quite some space, whilst there is not that much interesting going on. Another approach might help us in this regard.

A more interesting reason for this thesis, is the fact that such representations can possibly capture different levels of detail, which might not easily be noticeable when just looking at the raw pixel representation of the image.

3.2. Discrete Fourier Transform (DFT) representation

The DFT transforms an image from the spatial domain to the frequency domain. The frequency domain is called that because every pixel value corresponds to how much a certain frequency is present in the spatial domain image. The frequency domain is also referred to as the Fourier domain.

The difference with the regular Fourier Transform is that the DFT can be applied to a discrete input, such as an image. The output of the DFT will have the same dimensionality as the input, i.e. for an image of $256x256$, the DFT gives a new representation that is also $256x256$.

Definition 3.2.1 (Two-dimensional DFT)**.** Given a two-dimensional signal $f(x, y)$ of size $N \times N$, the two-dimensional DFT transforms it into a frequency-domain representation $F(a, b)$ for $0 \leq a, b \leq N - 1$, as follows:

$$F(a,b) = \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} f(k,l) e^{-i2\pi(\frac{ak}{N} + \frac{bl}{N})},$$

Definition 3.2.2 (Two-dimensional Inverse DFT)**.** The two-dimensional DFT-II reconstructs an image $f(x, y)$ from its frequency-domain representation $F(a, b)$ using the formula:

$$f(x,y) = \frac{1}{N^2}{}^2 \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} F(k,l) e^{i2\pi(\frac{xk}{N} + \frac{yl}{N})},$$

Note that for any pixel, the frequency representation is a complex number. Therefore, there are multiple ways of representing this frequency domain image. We could split every pixel into its real F_R and imaginary counterpart F_i, in the following manner:

$$F(a,b) = F_R(a,b) + iF_I(a,b).$$

It is also possible to split every image into the magnitude and phase by using the fact that $|F(a, b)|$ is the magnitude, and $\arctan(F_I(a, b)/F_R(a, b))$ is the phase. It is most common to visualize the frequency domain by using the magnitude representation. If we would want to invert the frequency domain image,

[2]Normalization, can also possibly be applied to the forward transform, but then it should not be here.

we also need the phase in addition to the magnitude. The set of all magnitude coefficients $|F(a, b)|$ of an image, is also referred to as the amplitude spectrum[3], this shows us exactly what frequencies contribute more to the magnitude of an original image.

Using the raw values for visualization purposes of the magnitude, does not give useful results, as the range of coefficients can be so large that differences cannot be shown properly. That is why in visualizations of the Fourier domain, it is usual to take the logarithm of the magnitude.

Another practice that is quite custom in such images, is to switch around the first and fourth quadrant and the second and third quadrant. This is done such that the $F(0, 0)$, the average, is in the middle of the image.

An example of how DFT representations of images are commonly used, that is, where we use the magnitude, log-scale it and switch around the quadrants so that $F(0, 0)$ is in the middle, is shown in Figure 3.2. Note that even though the normal DFT forms an orthonormal basis and thus should one can retrieve the original image based on its DFT representation, if we are to only consider the Magnitude of the DFT for every coefficients, we cannot retrieve the original image, as we would need access to the phases to truly be able to reconstruct the original image.

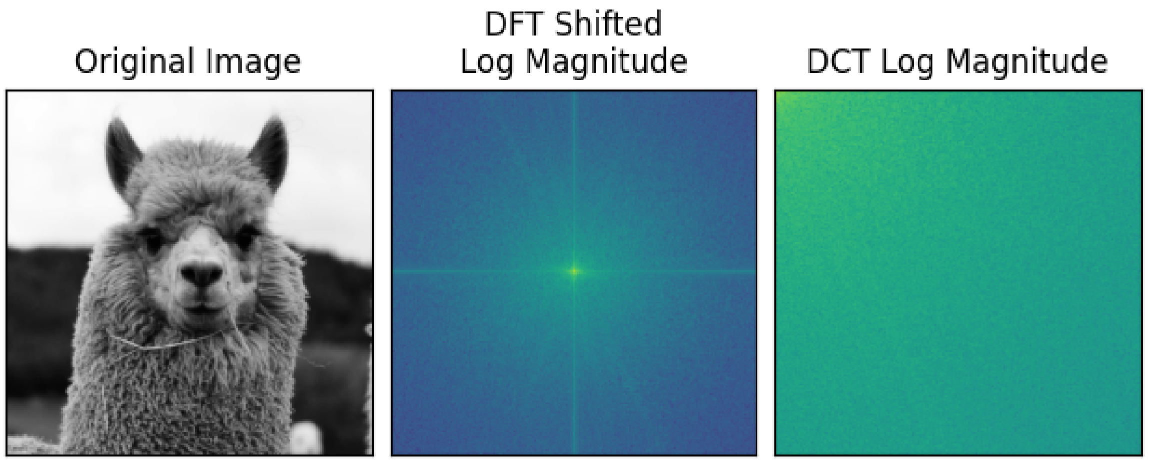


Figure 3.2: Image of an alpaca and two representations of the image in terms of frequency-based representations.

This specific choice of using only the magnitudes of DFT gives us some interesting properties. One of these properties is that the representation does not change under translations in the original image. That is, if we would change the original image by just translating a block of pixel values, then the DFT shifted log magnitude spectrum would still be the same.

3.2.1. Discrete Cosine Transform (DCT) representation

The Discrete Cosine Transform (DCT) is another type of representation which uses sine functions to represent an image. Thus, similar magnitude spectrum analyses can be used as in the case of DFT, by considering what kind of frequencies are needed to represent the original signal. There are a number of benefits the DCT has over the DFT, such as the fact that DCT only has real coefficients and that it has nice energy compaction properties, making it useful for compression.

There are multiple versions of DCT representations, of which the most commonly used form is the DCT-II algorithm. We will consider a paper that used this representation exactly for synthetic image detection in Chapter 4.

Definition 3.2.3 (Two-Dimensional DCT-II). Given a two-dimensional signal $f(x, y)$ of size $N \times N$, the two-dimensional DCT-II transforms it into a frequency-domain representation $F(u, v)$ for $0 \leq u, v \leq$

[3]Sometimes a power spectrum is mentioned, this is simply the set of all coefficients $|F(a, b)|^2$ of an image.

$N-1$, as follows:

$$F(a,b) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\left[\frac{\pi}{N}(x+\frac{1}{2})a\right]\cos\left[\frac{\pi}{N}(y+\frac{1}{2})b\right].$$

Definition 3.2.4 (Two-Dimensional Inverse DCT-II (IDCT-II))**.** The two-dimensional IDCT-II reconstructs an image $f(x,y)$ from its frequency-domain representation $F(u,v)$ using the formula:

$$f(x,y) = \sum_{a=0}^{N-1}\sum_{b=0}^{N-1} F(a,b)\cos\left[\frac{\pi}{N}(x+\frac{1}{2})a\right]\cos\left[\frac{\pi}{N}(y+\frac{1}{2})b\right].$$

In Figure 3.2 we have visualized such a DCT-based transform. Once again, we need to take the logarithm otherwise the coefficients are to spread out and such visualizations are not informative. Commonly, these DCT-based transform are not shifted, rather the $F(0,0)$, the average component is found in the upper left corner.

3.2.2. Localization Challenges for frequency-based representations

One of the major challenges for frequency-based representations such as DCT or DFT is that they do not have the capability to connect the observed frequencies to their location in the pixel domain. To illustrate this, let us consider a simplified Where's Waldo problem: We have an image of a number of objects and we want to find the object which is filled with white-red striped filling. This has been visualized in Figure 3.3. The striped filling can be understood as a higher frequency, and is visible throughout the DFT domain as the horizontal stripes. Note that the DFT domain does not give us any information about where our striped figure might be, rather only that there is some striped figure somewhere in the image.

This setting is exactly where wavelet-based techniques can be very useful: It can act as a bridge between the spatial and the frequency domain. As can be seen in the rightmost image of Figure 3.3, we can easily see where we can find the striped figure in the original image.

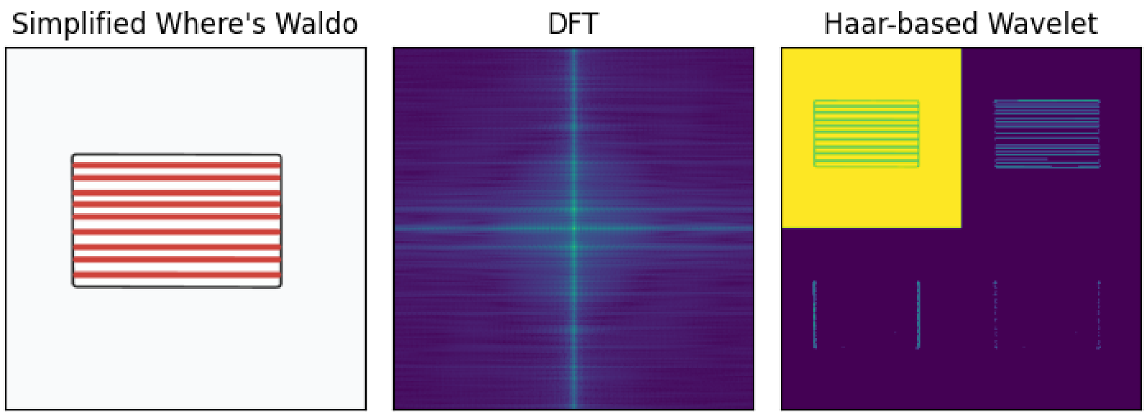


Figure 3.3: Left: Simplified version of Where's Waldo, our goal is to find the white-red striped filled object. Middle: Discrete Fourier Transform of the left image. Right: Haar-based wavelet representation of the left image. Notice how the rightmost picture tells us informs us exactly where the our high-frequency content is.

In the upcoming section we will dive deeper into how wavelets can allow for this type of localization of frequency information, what other advantages they offer over DFT-based approaches and what a wavelet is mathematically speaking.

3.3. One-dimensional wavelets

In this section, we will introduce wavelets. More specifically, we will work towards introduction of the one-dimensional Discrete Wavelet Transform (DWT), which we will be the representation we that our research is based upon.

Before introducing the Discrete Wavelet Transform, we will start off by describing easier concepts in the form of filters and filter banks. We will see later that the Discrete Wavelet Transform can be implemented as a filter bank. After concluding our introduction of filters and filter banks, we will consider a generalization of the Discrete Wavelet Transform in the form of the Continuous Wavelet Transform (CWT). Afterwards we will see how the CWT is a redundant transformation and a DWT can allow for a more succinct representation of a signal. Afterwards we will discuss specifics a number of specific details of the DWT and finally we will conclude this section by introducing wavelet packet transforms.

3.3.1. Filters

A filter is a linear operator that is time invariant. Given an input vector x, a filter returns an output vector y, which is convolved with a fixed vector h. This fixed vector h differs per filter and is the defining characteristic of a filter, which we will see later on.

Let us assume that our input vector, x is an equidistant discretization over time of some one-dimensional signal. That means that we can interpret the value of $x(n)$ and $y(n)$ for some $n = 0, \pm 1, \pm 2, \ldots$ as the value of the input signal at time n and the value of the filtered input signal at time n. Given this interpretation, we can define a filter mathematically, which we will do in Definition 3.3.1. We will only focus on discrete filters in this thesis and thus disregard continuous filters.

Definition 3.3.1 (Filter)**.** A filter is an operator that convolves an input signal x with a fixed vector h to produce output vector y. That is, for all $n \in \mathbb{Z}$:

$$y(n) = \sum_{k \in \mathbb{Z}} h(k)x(n-k). \tag{3.1}$$

We note that the fixed vector h in a filter also needs to carry information about the relationship between its elements and the corresponding positions in time: Suppose for example that we have h of length two, then the behaviour of the filter is different if we set $h(0)$ and $h(1)$ non-zero, or if we set $h(-1), h(0)$ as non-zero. For calculation of output value $y(n)$, the former would look at the current input element $x(n)$ and the last input element $x(n-1)$, whereas the latter would consider the current $x(n)$ and the next input element $x(n+1)$.

For any filter, a special input is the unit impulse, where $x(n) = \mathbb{1}_{n=0}$. This input is special, because it causes the convolution on the right-hand side of eq. (3.1) to simplify to $h(n)$, as $x(n-k)$ is zero if k is not equal to n. The resulting output vector of applying a filter operation to the unit impulse, is called the impulse response. In this thesis, we will only consider filters for which the impulse response has a finite number of non-zero components.

We will consider two types of filters: Low-pass filters and high-pass filters. To this end, it will be useful to introduce the Convolution theorem, which tells us that convolution by h in the time domain becomes multiplication by H in the frequency domain, where H is the one-dimensional DFT of $h(n)$. We will show why this property holds in the proof of Theorem 3.3.1.

Theorem 3.3.1 (Convolution Theorem)**.** Convolution by h in the time domain becomes multiplication by H in the frequency domain. Given input $y(n) = (x * h)(n)$, then, for any ω:

$$Y(\omega) = X(\omega)H(\omega).$$

Proof. We will start of by using the one-dimensional DFT on $y(n)$.

$$\begin{aligned}
Y(\omega) &= \sum_{n\in\mathbb{Z}}\sum_{k\in\mathbb{Z}} h(k)x(n-k)e^{-jn\omega} \\
&= \sum_{k\in\mathbb{Z}} h(k)\sum_{n\in\mathbb{Z}} x(n-k)e^{-jn\omega} \\
&= \sum_{k\in\mathbb{Z}} h(k)e^{-jk\omega}\sum_{n\in\mathbb{Z}} x(n-k)e^{-j(n-k)\omega} \\
&= \sum_{k\in\mathbb{Z}} h(k)e^{-jk\omega}X(\omega) \\
&= X(\omega)\sum_{k\in\mathbb{Z}} h(k)e^{-jk\omega} \\
&= X(\omega)H(\omega).
\end{aligned}$$

□

3.3.1.1. Low-pass filter

One example of a low-passs filter, is the filter that, at time n, returns the average output of the input signal at time n, $x(n)$ and the output of the input signal at the last timestep $n-1$, $x(n-1)$. That is:

$$y_L(n) = \frac{1}{2}x(n) + \frac{1}{2}x(n-1). \tag{3.2}$$

The filter coefficients for this filter are $h_L(0) = \frac{1}{2}$, $h_L(1) = \frac{1}{2}$. Note that this filter corresponds to a moving average, where we only use the current and last element to find the moving average.

Intuitively, this filter should be less sensitive to one-off jumps, since it uses multiple values to find the average and thus one-off jumps should be less noticeable in the moving average than in the original signal. Thus such a filter should be able to capture low-frequency information that is present in the signal of interest.

To formalize this notion of a filter that is less sensitive to one-off jumps and is better able to capture low-frequency information present in a signal, we will introduce the signal $x(n) = e^{in\omega}$, which has a pure frequency of just ω, rather than multiple frequencies present in the unit impulse. This input has the property that for any filter, the output vector becomes a multiple of the input vector.

$$\begin{aligned}
y(n) &= \frac{1}{2}x(n) + \frac{1}{2}x(n-1) \\
&= \frac{1}{2}e^{in\omega} + \frac{1}{2}e^{i(n-1)\omega} \\
&= (\frac{1}{2} + \frac{1}{2}e^{-i\omega})e^{in\omega}.
\end{aligned}$$

We refer to $(\frac{1}{2} + \frac{1}{2}e^{-i\omega})$ as the frequency response function, $H(\omega)$. Notice that the for $\omega = 0$, we have that the frequency response function of our filter is equal to 1, since $H_L(0) = 1$. If our input were to be $x = e^{in0} = (\ldots, 1, 1, 1, 1, \ldots)$, then the moving average filter would return exactly the input, $y_L = (\ldots, 1, 1, 1, 1, \ldots)$, as the average will always be one.

On the other hand, for $\omega = \pi$, we have that the frequency response function of our filter is equal to 0. If our input were to be $x(n) = e^{in\pi} = (\ldots, -1, 1, -1, 1, \ldots)$, then the moving average filter would return, $y_L(= (\ldots, 0, 0, 0, 0, \ldots)$, as the moving average will always take the difference between 0 and 1. Note how all of the information of the original signal got lost here.

For other frequencies, such as $\omega = \frac{1}{2}$, the value of $H_L(\omega)$ is a complex exponential, so we can use magnitude and phase plots of the function to have a better understanding of what happens for different values of ω. The magnitude is $|H_L(\omega)| = \cos\frac{\omega}{2}$. We have added a plot of $|H_L(\omega)|$ in Figure 3.4, which shows that the filter has less and less magnitude when approaching higher frequencies.

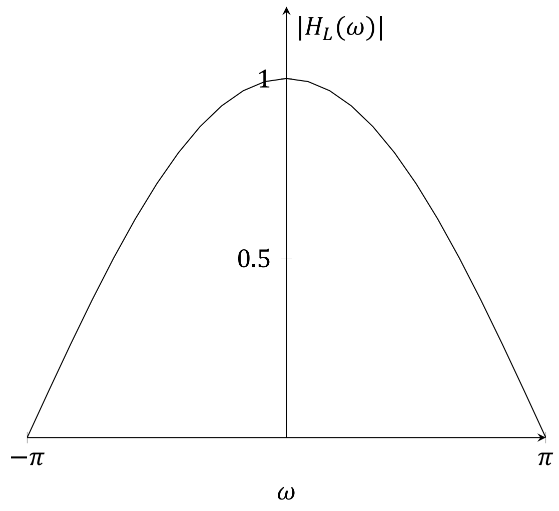


Figure 3.4: Plot of $|H_L(\omega)| = \cos(\frac{\omega}{2})$

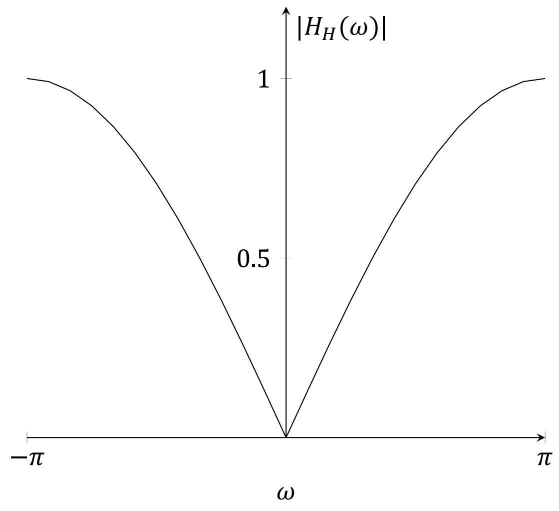


Figure 3.5: Plot of $|H_H(\omega)| = |\sin(\frac{\omega}{2})|$

Here we can see why the moving average filter is referred to as a low-pass filter: For low-frequencies apparent in the input signal, the filter is close to 1 in magnitude and will not impact the frequencies too much with respect to the output signal, whereas for high-frequencies apparent in input signal, the filter is close to 0 in magnitude and will almost completely remove them from the output signal.

The second and last type of filter we will look at, does the complete opposite of what we have just seen: This new type of filter will try to keep high-frequencies present in the original signal while trying to remove the low-frequencies present.

3.3.1.2. High-pass filter

Low-pass filters reduce or remove high frequency components and instead focusses on low frequency components present in the signal. By doing this, such a filter removes or reduces jumps in a signal, as we have seen in the moving average filter. A high-pass filter does the opposite: It removes or reduces low-frequency components.

An example of a high-pass filter, is the counterpart of the moving average filter: the moving difference filter, which returns at time n half the difference between the original signal at time n and the original signal at time $n-1$. That is:

$$y_H(n) = \frac{1}{2}x(n) - \frac{1}{2}x(n-1). \tag{3.3}$$

The filter coefficients of this filter are $h_H(0) = \frac{1}{2}$ and $h_H(1) = -\frac{1}{2}$. Whereas the moving average filter would be less sensitive to one-off jumps, the moving difference filter is more sensitive to these one-off jumps: Suppose the input signal is $x = (..., -1, 1, -1, 1, ...)$, the moving average filter would just return all zeros, whereas the moving difference filter will return exactly the input signal.

We can formalize this notion of being better able to capture high-frequency information by means of looking at $H_H(\omega)$. To this end, we can once again analyze the results of applying the filter to input signal $x(n) = e^{in\omega}$.

$$\begin{aligned}
y_H(n) &= \frac{1}{2}x(n) - \frac{1}{2}x(n-1) \\
&= \frac{1}{2}e^{in\omega} - \frac{1}{2}e^{i(n-1)\omega} \\
&= (\frac{1}{2} - \frac{1}{2}e^{-i\omega})e^{in\omega}.
\end{aligned}$$

We can see that for the moving difference filter, $H_H(\omega) = \frac{1}{2} - \frac{1}{2}e^{-i\omega}$, which is exactly equal to 0 for $\omega = 0$ and 1 for $\omega = \pm\pi$, the precise opposite of our moving average filter. However, once again, the frequency response function is complex and thus we can consider the magnitude $|H_H(\omega)|$ to have a better understanding of what effect the filter will have on frequencies different from 0 and $\pm\pi$. The magnitude of this filter is $|H_H(\omega)| = |\sin(\frac{\omega}{2})|$, which we have plotted in Figure 3.5. This figure shows that indeed, the moving difference filter has large magnitude for frequencies close to π, whereas it has near zero magnitude for frequencies close to 0.

If we are to convolve some input signal x with our moving difference filter, the frequencies present in the output signal can be found exactly by using $Y(\omega) = H_H(\omega)X(\omega)$. This relation shows exactly why our moving difference filter can be considered a high-pass filter: low-frequencies present in the input signal, close to 0, will be multiplied by some $H(\omega)$ which is close to 0 in magnitude and thus will be reduced, whereas high-frequencies apparent in the input, which are close to $\pm\pi$ will be multiplied by a value close to one in magnitude, and thus will not be affected as severely.

3.3.2. Filter banks

Our introduced moving average and moving difference filters are not invertible. This can be seen by the fact that both $(\ldots, -1, 1, -1, 1, \ldots)$ and $(\ldots, -2, 2, -2, 2, \ldots)$ lead to the same output for the moving average filter: that is the zero vector. Similarly, we can see that the moving difference filter applied to the all ones or on the all twos vector returns the zero vector, thus is also not invertible. This can also be understood by considering Theorem 3.3.1: since some frequencies are completely wiped out by the filters, it is not possible to retrieve the original frequency.

However, we notice that the two filters complement each other quite well: The frequency removed by the moving average filter, is not removed by the moving different filter and vice versa. The usage of multiple filters in conjunction such that the filters have common input or output is referred to as a filter bank. The use of filter banks will not allow us to reconstruct an original signal, but the use of filter banks will also allow for simultaneous analysis of a signal in terms of different frequencies.

Two important tools that are important for designing useful filter banks, are the downsampling and upsampling operators. We will see that it is possible to only keep half of the outputs of the moving average filter and of the moving difference filter to reconstruct the original input.

Definition 3.3.2 (Downsampling operator)**.** Downsampling $y = (\ldots, y(-2), y(-1), y(0), y(1), y(2), \ldots)$ by some factor n, and offset $i \in \{0, \ldots, n-1\}$ is an linear operator that works in the following way:

$$(\downarrow n, i)(y) = (\ldots, y(i - 2 \cdot n), y(i - n), y(i), y(i + n), y(i + 2 \cdot n), \ldots).$$

In the remainder of this thesis, we will only use downsampling operators with factor 2 and with an offset of 0. If not mentioned otherwise, it is to be assumed that when we are referring to the downsampling operator, the downsampling operator with factor 2 and offset 0 is used.

Because we want to downsample the results of using our high-pass and low-passs filter, we should use a normalized version of the filters, which can be achieved by simply multiplying the filters with $\sqrt{2}$: For the moving average filter this would change the filter to $h(0) = \frac{1}{\sqrt{2}}$, $h(1) = \frac{1}{\sqrt{2}}$.

The interested reader can read the work by Strang and Nguyen [42] for why this normalization constant is needed, but in summary: it causes the filter bank to become energy preserving.

The downsampling operator is not invertible: There is no way to know what the odd-valued inputs were in the input signal. Thus, in the upsampling operator, we will just insert zeroes at positions of unknown coefficients.

Definition 3.3.3 (Upsampling operator)**.** Upsampling $y = (\ldots, y(-2), y(-1), y(0), y(1), y(2), \ldots)$ by some factor n is a linear operator that inserts $n - 1$ zeros between each pair of consecutive elements in the input sequence. Formally, the upsampling operator, denoted by $(\uparrow n)$, can be defined as follows:

$$(\uparrow n)(y) = (\ldots, 0, y(0), 0, 0, y(1), 0, 0, y(2), 0, 0, \ldots).$$

The output of the upsampling operator will have n times more elements than the input sequence, with zeros inserted between each pair of consecutive elements. More specifically, if the input sequence has length M, the output sequence will have length $n \cdot M$.

In the context of this thesis, we will only use upsampling with factor 2. Therefore, when referring to the upsampling operator, it is assumed to be the upsampling operator with factor 2.

Now that we have introduced both downsampling and upsampling, we can dive into our first filter bank. To this end, we will use the normalized moving average filter, where $h(0) = \frac{1}{\sqrt{2}}$ and $h(1) = \frac{1}{\sqrt{2}}$.

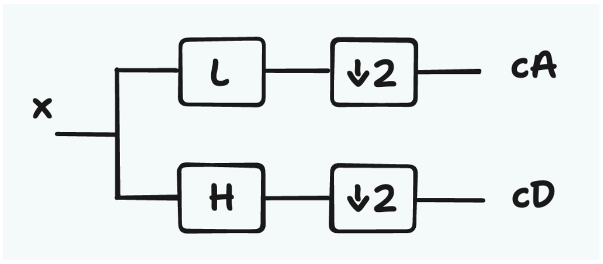


Figure 3.6: A two-channel analysis filter bank, where input signal x is put in and two filters, L and H are applied to x. Afterwards, the result of applying both L and H is downsampled, yielding in coefficients cA and cD respectively. Usually L is chosen to be some type of low-pass filter, such as the moving average filter and H is chosen to be some type of high-pass filter, such as the moving difference filter.

Notice that, since our input signal is assumed to have finitely many non-zero values, we can write the normalized moving average filter as a matrix, which we will refer to as L. Similarly, we can introduce the normalized moving difference filter, where $h(0) = \frac{1}{\sqrt{2}}$ and $h(1) = -\frac{1}{\sqrt{2}}$ as H.

The usual way of representing filter banks is by means of a block diagram. We will follow this convention and have introduced Figure 3.6, where we break down our original signal into coefficients cA and cD, the former of which is the result of applying our example low-pass filter L and afterwards downsampling the low-passed output and the latter the result of applying our example high-pass filter H on the input x and afterwards downsampling the high-passed signal. Such a filter bank where a single input signal is used by multiple filters to return multiple coefficients, is called an analysis bank.

The filter bank shown in Figure 3.6 returns us two sets of coefficients, cA and cD. Since the cA coefficients are the result of using a low-passs filter, we can also understand it as an approximation of the original signal. In the same vein we can understand cD to be detail coefficients of the original signal. Rather than focusing on more general patterns in the signal that are apparent in the low-frequencies of the signal, the result of a high-pass filter shows the details that are more apparent if we are to consider higher frequencies.

Since we are using the moving average and moving difference filters, we can make use of the following property to reconstruct the original signal x from the results of the filter bank shown in Figure 3.6:

Lemma 3.3.1. For any n:

$$x(2n) = \frac{1}{\sqrt{2}}(cA(n) + cD(n)),$$

and

$$x(2n-1) = \frac{1}{\sqrt{2}}(cA(n) - cD(n)).$$

Proof. Since $cA(n) = \frac{1}{\sqrt{2}}(x(2n) + x(2n-1))$, and $cD(n) = \frac{1}{\sqrt{2}}(x(2n) - x(2n-1))$, we can find the desired identities by direct application of $cA(n)$ and $cD(n)$.

$$\begin{aligned}\frac{1}{\sqrt{2}}(cA(n) + cD(n)) &= \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}(x(2n) + x(2n-1)) + \frac{1}{\sqrt{2}}(x(2n) - x(2n-1))) \\ &= \frac{1}{2}(x(2n) + x(2n-1) + x(2n) - x(2n-1)) \\ &= \frac{2x(2n)}{2} = x(2n).\end{aligned}$$

Similarly, we can recover $x(2n-1)$:

$$\begin{aligned}\frac{1}{\sqrt{2}}(cA(n) - cD(n)) &= \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}(x(2n) + x(2n-1)) - \frac{1}{\sqrt{2}}(x(2n) - x(2n-1))) \\ &= \frac{1}{2}(x(2n) + x(2n-1) - x(2n) + x(2n-1)) \\ &= \frac{2x(2n-1)}{2} = x(2n-1).\end{aligned}$$

□

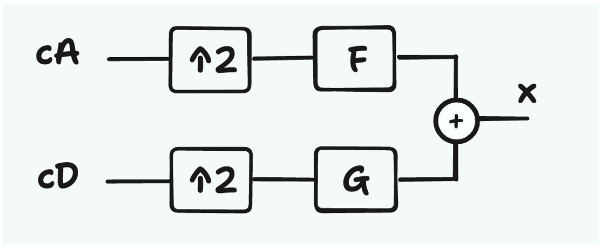


Figure 3.7: A two-channel synthesis bank. Usually, input coefficients cA and cD are the resulting outputs of a two-channel analysis bank such as shown in Figure 3.6. This filter bank upsamples the two coefficients, then applies filter F and G on the upsampled coefficients and finally sums up the results to return signal x. Under appropriate circumstances, x is exactly the input signal given to the filter bank shown in Figure 3.6.

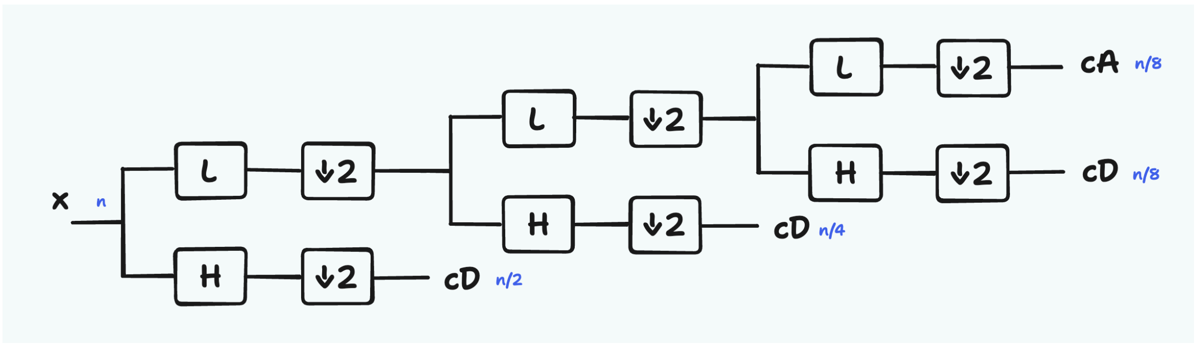


Figure 3.8: A three-level analysis bank, which can be understood as a generalization of the filter bank shown in Figure 3.6. After every level, the result of L, the low-pass filter, can be subject to another filter bank, which yields another set of approximation coefficients and detail coefficients, cA and cD. Notice how at every level the length of the signal is halfed.

Using Lemma 3.3.1, we can design a synthesis bank, which takes as an input multiple coefficients and returns a single signal. Usually, a synthesis bank has a counterpart analysis bank for which the synthesis bank is the inverse: That is, if we apply the filter bank to some input x, the synthesis bank applied on the resulting coefficients should return the same input x back.

To this end, we will introduce filter F of length two with non-zero coefficients $f(0) = f(1) = \frac{1}{\sqrt{2}}$. Besides, we will introduce filter G of length two which has non-zero coefficients $g(0) = -\frac{1}{\sqrt{2}}$ and $g(1) = \frac{1}{\sqrt{2}}$ Notice how the signs are the other way around relative to the moving difference filter.

To reconstruct the original signal given the resulting coefficients of filter bank fig. 3.6, we will first need to upsample the coefficients, then we apply filter F on the upsampled cA coefficients and we apply filter G on the upsampled cD coefficients, finally, by adding the two resulting signals, we will retrieve the original input. This process has been visualized in Figure 3.7.

An important observation can be made that we can use our filter bank from Figure 3.6 in a multilevel fashion. This is done by realizing that the filter bank can be applied to any signal, and thus recursively applying the whole filter bank to the cA coefficients, as shown in Figure 3.8.

In Figure 3.8, we can see that the original signal x is split into high frequency coefficients cD, which we will refer to as the first level detail coefficients and the approximate coefficients are once again given as an input for a two-channel filter bank. This yields us a second level of detail and approximation coefficients, of which we insert the approximation coefficient once again into a two-channel filter bank, yielding us third level approximation and detail coefficients. In blue we have added the sizes of all signals. Notice how the signals get halved in size at every level.

The advantage of this type of filter bank is that it allows for a multiscale analysis. This offers several advantages over a regular filter approach. Firstly, it allows us to more easily discern different sources of frequencies. We show an example of such a case in Figure 3.10, where we plot the different level coefficients of an input signal that we find by considering $f(x) = \sin(x) + \cos(8x)$. In the fourth level average coefficient, we can see that that the low-frequency $\sin(x)$ is visible, whereas that would not be that easily visible if we were to only consider the first level coefficients, which correspond to the filter bank Figure 3.6.

Secondly, it allows for analysis of the same signal across multiple scales. An example of this phenomenon can be found in Figure 3.9, where we are considering a step function, its noisy variant and different level detail coefficients that are found when applying the a multilevel filter bank to the noisy y signal. Notice that the simultaneous analysis of multiple detail coefficients at different levels allow us to get a better idea of where the jumps were in the original signal, rather than if we were to only consider the first level detail coefficient.

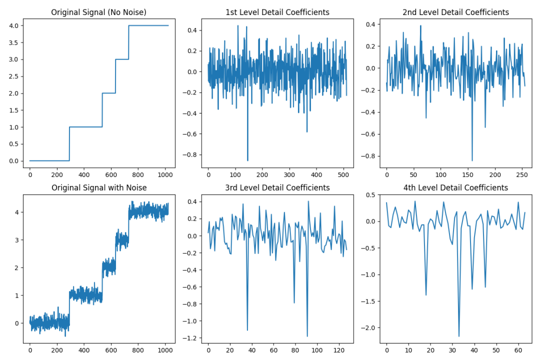


Figure 3.9: In the top-left corner we see some step function, of which the bottom-left corner is a noisy variant. The top-middle plot is a visualization of the cD coefficients of a single level filter bank such as shown in Figure 3.6, where we use the moving average and moving difference filters as low-pass and high-pass filters respectively. The bottom-middle and all plots on the right correspond to cD coefficients of the higher levels of a multi-level filter bank such as shown in Figure 3.8. Notice how the step jumps are easier to notice when analyzing multiple levels of detail at once.

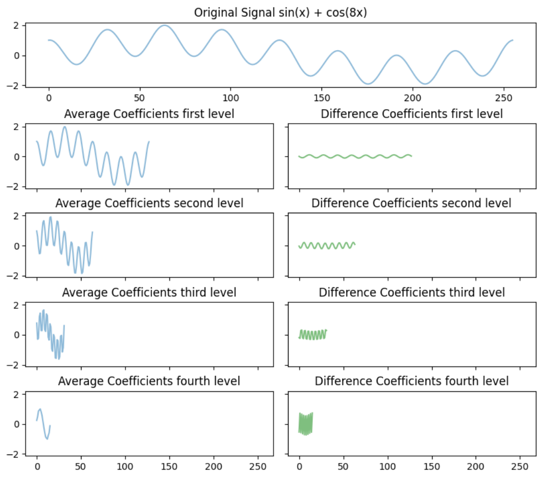


Figure 3.10: Multi-level filter bank analysis of function $f(x) = \sin(x) + \cos(8x)$. For every level of the filter bank, we plot both the approximate coefficients as well as the detail coefficients. Notice how we can clearly see the original $\sin(x)$ function appear in the fourth level approximate coefficients.

Given that we can reconstruct a two-channel filter bank such as Figure 3.6, we can also easily find how to reconstruct a recursive filter bank such as Figure 3.8: Recursive application of the highest level detail and approximation coefficients yields the average coefficient of the penultimate level. This procedure can be applied recursively until there are no detail coefficients anymore and we find our original input x.

We can use any signal x in the filter banks shown in Figures 3.6 and 3.8 and use their respective synthesis banks to recreate the original signal. This means that we can understand the output of the analysis banks as an alternative representation of the original signal, similar to the Fourier Transform of a signal.

Notice that this representation is dependent on our choices of filters L, H. Moreover, due to the existence of the synthesis bank, we know that these coefficients truly are a one-on-one representation. Suppose for example that L and H were chosen in such a way that the synthesis bank that returns the original input given the output of the analysis bank, then this representation might not be one-on-one.

This leads us to the question: Are there other choices of L and H that lead to the existence of a synthesis bank F and G and also split a signal into a low frequency signal and a high frequency signal? To answer these questions, introduce wavelets, the discrete wavelet transform and its connection to filter banks in the upcoming section.

3.3.3. Continuous Wavelet Transform

Wavelets are, as the name suggests, “small” waves. As can be seen in Figure 3.11, where we plot a sine function next to a Mexican Hat function, a type of wavelet, this smallness is expressed in terms of the oscillatory behaviour of the wave. Whereas the sine function oscillates infinitely, the wavelet function has a small local oscillation.

Many signals and images display sections of piecewise smooth behaviour in between irregular short jumps. In images, these short jumps can be seen in edges, in one-dimensional signals such as financial time series these can be understood in economic upturns or downturns. Fourier analysis breaks up signals of interest into sine waves of different frequencies and thus is less suited to represent such

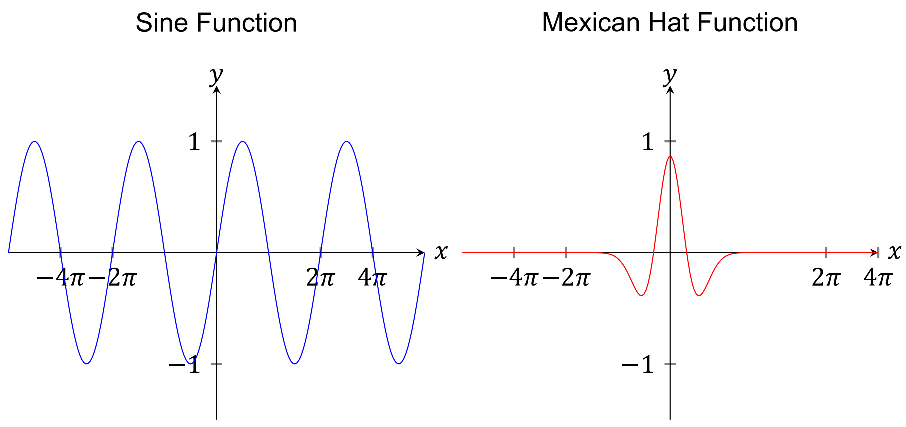


Figure 3.11: Sine function and Mexican Hat function with extended domain

localized frequency changes. Wavelet analysis on the other hand breaks up a signal of interest in diluted and translated versions of a so-called mother wavelet. These diluted and translated versions are referred to as daughter wavelets.

Not any function can be used as a mother wavelet for the goal of breaking a signal of interest into the daughter wavelets. We will consider Definition 3.3.4 taken from Blatter [2].

Definition 3.3.4 (Mother wavelet)**.** A mother wavelet is a function ψ that adheres to the following requirements:

1. The function ψ is absolutely integrable, that is:

$$\int_{-\infty}^{\infty} |\psi(t)| dt < \infty.$$

2. The function ψ is square integrable, that is:

$$\int_{-\infty}^{\infty} \psi(t)^2 dt < \infty.$$

3. The function ψ has mean-zero:

$$\int_{-\infty}^{\infty} \psi(t) dt = 0.$$

Using this mother wavelet, we can introduce the translated and diluted versions of the mother wavelet, the so-called daughter wavelets. Since these functions are translated and diluted, we will introduce scaling factor $a \in \mathbb{R}\backslash\{0\}$ and translation factor b. The daughter wavelet found by scaling the original wavelet with factor $b \in \mathbb{R}$ and then scaling with factor a is:

$$\psi^*_{a,b}(t) = \frac{1}{\sqrt{|a|}}\psi(\frac{t-b}{a}). \tag{3.4}$$

Scaling means stretching if $|a| > 1$ or compressing if $|a| < 1$ the signal, whereas translation by b just means shifting the position over time.

Given some finite-energy signal, $s(t)$ we can analyze how well a certain daughter wavelet aligns with the signal by convoluting the two and considering the resulting value. This value will be high when $s(t)$ aligns with the daughter wavelet and low if not. Using this idea, we can introduce the Continuous Wavelet Transform (CWT), which uses this idea to transform a signal $s(t)$ such that we can analyze the signal simultaneously in frequency and location.

Definition 3.3.5 (Continuous Wavelet Transform)**.** [4] Given signal $s(t)$, mother wavelet ψ, scaling factor $a \neq 0$, and translation factor b, then the Continuous Wavelet Transform is defined as:

$$W(a,b) = \int_{-\infty}^{\infty} s(t)\psi^*_{a,b}(t)dt. \tag{3.5}$$

[4]We do not consider complex wavelets in this thesis, so we simplify the result to real wavelets.

In Figure 3.12 we are using the CWT to analyse a chirp signal function, which starts off with a low-frequency and gradually has an increasing frequency. This chirp signal is shown in the top plot of Figure 3.12. The CWT using the Mexican hat wavelet is shown in the bottom plot of Figure 3.12. Since the CWT is a two-dimensional function, we plot the translation term on the x-axis and the scaling factor on the y-axis. Note that y-axis of the bottom plot is inverted, that is, smaller scales are plotted higher. This is done because we want signals with higher frequencies - the more squashed versions of the mother wavelet - to correspond to points higher in the plot.

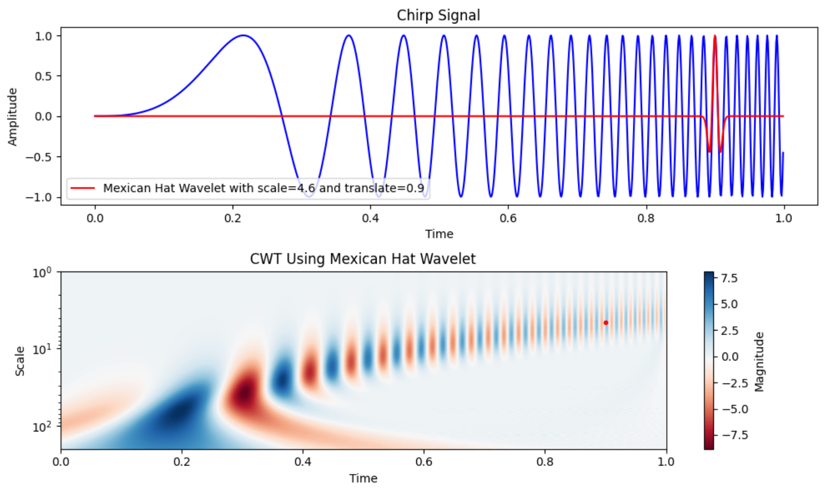


Figure 3.12: On top: a chirp signal. Below: visualization of the CWT of this chirp signal using the Mexican hat wavelet. Such visualizations are called scalograms. The values shown in this scalogram have been log-scaled to more clearly visualize the properties. Notice how the y-axis is inverted. Higher values of scale (and thus lower on our y-axis) correspond to lower frequency signals, whereas lower values of scale correspond to higher frequency signals. The scalogram clearly shows that earlier in time the Chirp-signal can be better described by low-frequency signals, and later in time the signal can be better described by higher frequency signals. The red dot in the scalogram matches the daughter wavelet of the Mexican hat function plotted in the top plot.

Any point in the bottom plot corresponds with some dilated and translated daughter wavelet of the Mexican hat function. To visualize this, we have added a red dot in the CWT plot and the corresponding Mexican Hat wavelet. Notice how well this specific daughter wavelet coincides with one of the jumps of the chirp signal. This was to be expected of course, because of the fact that we placed the red dot in an area with high magnitude.

Using Figure 3.12 we can understand how the continuous wavelet transform allows for localized frequency analysis of the chirp signal: It shows us that at times closer to 0 the chirp signal is better described with a signal that is stretched more (i.e. higher scale) and at times closer to 1 the signal is described better by daughter wavelets with lower scale. Notice that this is exactly what we would expect: In the begin of the chirp signal the frequencies apparent are relatively lower than at the end of the signal.

3.3.4. Discrete Wavelet Transform

The CWT creates a lot of redundant information. It is not needed to calculate the coefficient for every scale and translation value to get a useful representation of a signal of interest: If we know the value $W(a, b)$ then we can expect $W(a + \epsilon, b)$ to be similar for small enough ϵ.

One natural way of having less redundant information, is by using a dyadic grid, rather than allowing any value for a and b, we restrict them to powers of two (2^j for scale and $2^j k$ for translation), where j is an integer and k is an integer offset within the chosen scale. To this end, we introduce the following function $\psi_{m,n}$:

$$\psi_{m,n}(t) = \frac{1}{\sqrt{2^m}}\psi(\frac{t - n2^m}{2^m}). \tag{3.6}$$

Such discrete dyadic grid wavelets are commonly chosen to be an orthonormal basis for $L^2(\mathbb{R})$. That means that the wavelets are orthogonal to each other and have normalized unit energy. This orthonormality means that the following property has to hold:

$$\int_{-\infty}^{\infty} \psi_{m,n}(t)\psi_{k,l}(t)dt = \begin{cases} 1, & \text{if } m = k \text{ and } n = l, \\ 0, & \text{otherwise.} \end{cases}$$

The requirement for $\{\psi_{m,n}\}_{m,n\in\mathbb{Z}}$ to be a basis for $L^2(\mathbb{R})$ is quite restrictive. It causes a number of mother wavelets that can be used in the Continuous Wavelet Transform to be unusable in this setting, for example, the Mexican hat wavelet which we have seen in Figure 3.11. The interested reader can learn more about the additional properties a mother wavelet has to satisfy such that $\{\psi_{m,n}\}_{m,n\in\mathbb{Z}}$ does indeed form an orthonormal basis for $L^2(\mathbb{R})$ in the work by Christensen [5].

Definition 3.3.6 (Discrete Wavelet Transform)**.** Given a signal $s(t)$ and an orthonormal basis for $L^2(\mathbb{R})$, $\{\psi_{m,n}\}_{m,n\in\mathbb{Z}}$, the Discrete Wavelet Transform is defined as:

$$T(m,n) = \int_{-\infty}^{\infty} s(t)\psi_{m,n}(t)\, dt. \tag{3.7}$$

Using the fact that $\{\psi_{m,n}\}_{m,n\in\mathbb{Z}}$ is an orthonormal basis, we can reconstruct the original signal $s(t)$ given $\{T(m,n)\}_{m,n\in\mathbb{Z}}$ in the following manner:

$$s(t) = \sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} T(m,n)\psi_{m,n}(t).$$

An example of a mother wavelet for which $\{\psi_{m,n}\}_{m,n\in\mathbb{Z}}$ is an orthonormal basis for $L^2(\mathbb{R})$ is the Haar mother wavelet, which we define in Definition 3.3.7.

Definition 3.3.7 (Haar mother wavelet)**.** The Haar mother wavelet $\psi(t)$ is defined as:

$$\psi(t) = \begin{cases} 1, & 0 \le t < \frac{1}{2}, \\ -1, & \frac{1}{2} \le t < 1, \\ 0, & \text{otherwise.} \end{cases}$$

The orthonormality of the set $\{\psi_{m,n}\}_{m,n\in\mathbb{Z}}$ based on the Haar mother wavelet can be verified by noticing that for daughter wavelets on the same level m, the non-zero domains do not overlap and thus the their convolution is zero. For daughter wavelets on different scales, but having some overlap in terms of the domain, the convolution will be zero as the lower scale daughter wavelet will be constant where the higher scale daughter wavelet is non-zero and the mean is zero of any daughter wavelet.

Notice that even using the discrete dyadic wavelets, we will still need an infinite number of daughter wavelets to calculate the transform. Since images are two-dimensional, finite signals, we will need to understand how we can use the discrete wavelet transform for finite signals.

The translations of the wavelets are limited by the length of our signal of interest, thus we have an upperbound on that. The difficult part is in how many scales we should use. We gain more insights into this problem by considering the behaviour of the daughter wavelets in the frequency domain.

Suppose we start of with a daughter wavelet at some level m. The daughter wavelet at level $m+1$ is a stretched out version of that wavelet with factor 2, that is[5]:

$$\psi_{m+1,0}(t) = \frac{1}{\sqrt{2}}\psi_{m,0}(\frac{t}{2}). \tag{3.8}$$

Given the time scaling property of the Fourier transform, we can use the time-scaling property of the Fourier transform to find the following identity:

$$\mathcal{F}(\psi_{m+1,0}(t)) = \sqrt{2}\mathcal{F}(\psi_{m,0}(2t)), \tag{3.9}$$

where we us $\mathcal{F}$ to denote the operation of taking the Fourier transform.

[5]Without loss of generality we assume n=0.

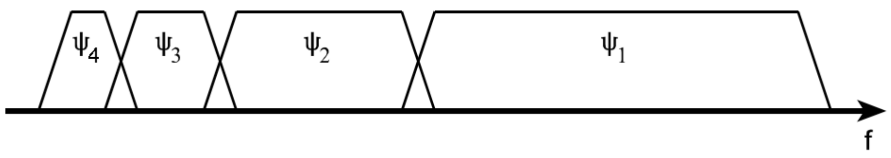


Figure 3.13: Figure that shows the frequencies attenuated by daughter wavelets of higher levels m. Notice how a higher level daughter wavelet corresponds to both a smaller width as well as a smaller position on the x-axis. Every next level daughter wavelet covers up half the remaining space until the zero axis. Taken from [44]

An additional, important piece of information we have on the Fourier transforms of our daughter wavelets, is the fact that their mean is zero, thus the Fourier transform of the daughter wavelets for frequency $\omega = 0$, is zero.

We can thus understand these daughter wavelets as some kind of high-pass filter, which only attenuates certain frequencies of the input signal. Using this understanding in combination with Equation (3.9), we can understand how different level wavelets can be understood as high-pass filters that are covering different parts of the frequency spectrum, where the daughter wavelets with higher values of m represent wider and larger bins in the frequency domain. Notice that these daughter wavelets do need to have some overlap, otherwise the transform wont be able to capture all possible frequencies. Correct design of the wavelets ensures this.

Notice that just using the daughter wavelets, it is not clear how we can cover the whole frequency spectrum until frequency zero, as every next level daughter wavelet only covers half the remaining space in Figure 3.13. Therefore, to cover the whole frequency spectrum, we would need an infinite number of wavelets.

Suppose we are to only use a finite number of wavelets, then there is a hole left behind in the frequency spectrum around the zero frequency. One solution is to use an additional function that can represent the frequencies present in the signal around this zero frequency. This has been visualized in Figure 3.14. This function is referred to as the father wavelet or scaling function and is a function that is unique to a mother wavelet.

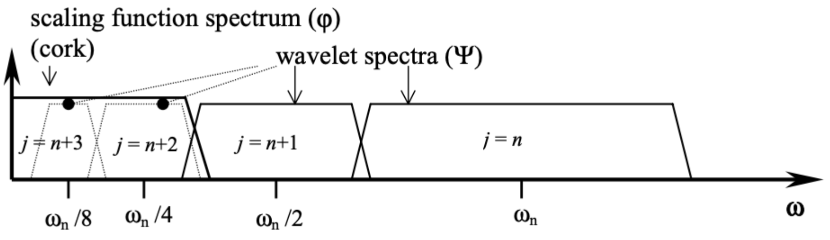


Figure 3.14: Figure that shows the frequencies attenuated by daughter wavelets of higher levels m. Notice that by using a scaling function that acts as a low-pass filter, the whole frequency spectrum can be covered wiht a finite number of functions. Taken from [44]

Definition 3.3.8 (Haar father wavelet)**.** The father wavelet corresponding to the Haar mother wavelet is:

$$\phi(t) = \begin{cases} 1, & \text{if } 0 \leq t < 1, \\ 0, & \text{otherwise.} \end{cases}$$

Similar to Equation (3.6), we can introduce $\phi_{m,n}(t)$, substituting ψ for ϕ in the equation. This will be useful in the upcoming section, where we will look at how the DWT can be implemented by filter banks.

3.3.4.1. DWT and filter banks

The discrete wavelet transform can be implemented by means of a filter bank. The schematic representation of this filter bank, is the same as shown in Figure 3.8. Notice that this filter bank relies on low-pass, high-pass filters and its corresponding analysis bank relies on two reconstruction filters. Moreover, if the DWT can be implemented by a filter bank, then the synthesis bank exists.

We will not consider the specifics on how to find these filters and their coefficients, but we will notice that these filters are heavily dependent on the choice of wavelet and scaling function. The interested reader can read the work by Vallens [44] to learn more about the specific relation between the mother wavelet function, scaling function and the filters used in filter banks. For our purposes it is enough to understand that any DWT can be implemented as a filter bank using filters, based on the specific wavelet choice. For our purposes, thus we can reduce the choice of DWT to these specific filters. In the upcoming sections we will talk about different choices of these filters.

3.3.4.2. Haar wavelet for DWT

We can use the Haar wavelet to transform signals using the DWT. Notice that we exactly know the form of the mother and father wavelet in this setting, definitions 3.3.7 and 3.3.8 respectively. In Figure 3.15 we show the decomposition filters for the low-pass and high-pass filter of the Haar wavelet. Notice that these filters exactly correspond with our moving average and moving difference filter introduced Sections 3.3.1.1 and 3.3.1.2.

Notice that the filter length is just two, and thus changes in the signal that are more gradual and span over more than two elements, might be less quickly picked up on by Haar-based representations.

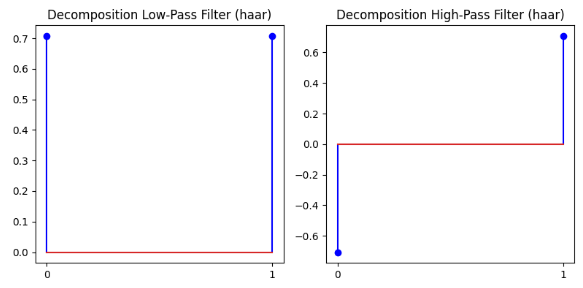


Figure 3.15: Decomposition filter coefficients for the Haar-wavelet.

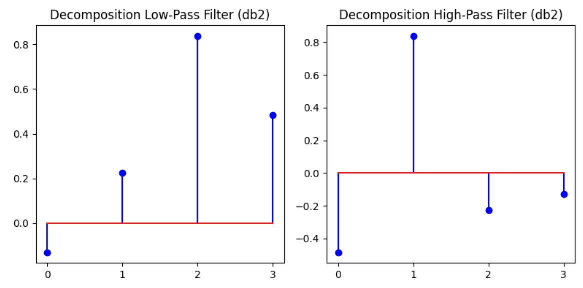


Figure 3.16: Decomposition filter coefficients for the db2, the Daubechies wavelet with two vanishing moments.

3.3.4.3. Daubechies Wavelets

One way of measuring the quality of a wavelet is by considering how many important functions the wavelet is orthogonal with [18]. One class of important functions one might want wavelets to be orthogonal with, are polynomials. By considering a wavelet with A vanishing moments, we know that the wavelet is orthogonal to all polynomials up to degree $A - 1$. Daubechies wavelets are the wavelets that, for any filter length of $2N$ have the most vanishing moments, exactly equal to N. For $N = 1$, the Daubechies wavelet is equal to the Haar wavelet. We will refer to any element of the Daubechies wavelet family by their number of vanishing moments: e.g. db2 will refer to the Daubechies wavelet with filter length 4 and two vanishing moments.

An interesting aspect of general Daubechies wavelets is that they do not have close-formed mother wavelet or scaling function[6]. However, these can be approximated as is done in Figure 3.17. Notice how the wavelet gets smoother for higher vanishing moments, thus by approximating functions with higher Daubechies wavelets, we are approximating signals with smoother functions.

Once again, these wavelets can be implemented by a filter bank such as shown in Figure 3.8. We have visualized the db2-filter coefficients needed for both the low-pass filter and the high-pass filter in Figure 3.16.

3.3.4.4. Boundary extensions

When using a filter of length larger than two, such as db2, issues can rise around the boundary of finite signals. Because the filter needs to have a number of elements as input, and obviously at the boundary of a finite signal signal, there might not be enough elements to calculate the resulting coefficient.

[6]Except for the db1, the Haar wavelet.

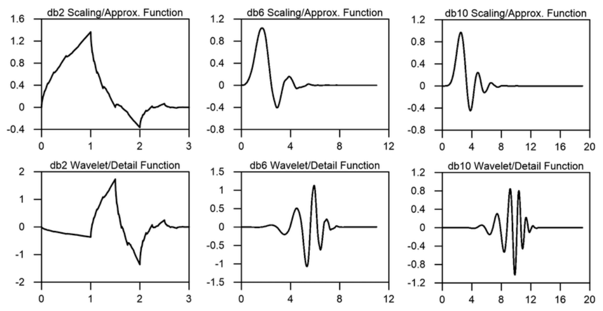


Figure 3.17: Scaling and (mother) wavelet functions for a number of members of the Daubechies wavelet family. Taken from [26]

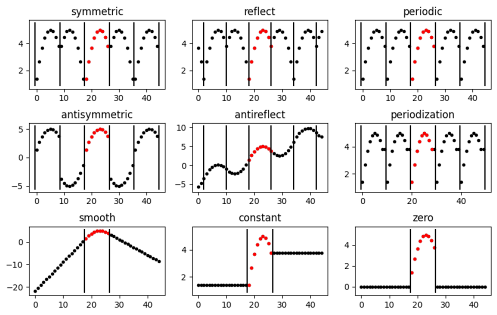


Figure 3.18: An overview of different types of boundary extension modes.

In all of our earlier writing, we simply use zeros when such a coefficient is undefined. This can be understood as padding the signals with zeroes on the boundaries. However, such an approach might be problematic if, for example the border values are all quite high, let's say 255. Boundary artifacts would be introduced by this choice of extension.

There are a number of different extension modes which can be useful for different type of signals. You can find a number of these extension modes in Figure 3.18.

3.3.5. Wavelet packets

When we constructed our multi-level analysis filter bank shown in Figure 3.8, we chose to apply the output of the low-pass filter to additional filter banks recursively. Wavelet packets approaches are a generalization of this idea, that allows for the filter bank to any outputting coefficient as you see fit.

An example would be where the filter bank is applied recursively on only the outputs of the high-pass filter. Wavelet packets can even be level specific: For example, one might apply the filter bank on the output of the first level low-pass coefficients, but then apply the filter bank on the second level high-pass coefficients.

In this thesis, we will only consider full wavelet packets, that is, wavelet packets for which every coefficient is used as input for a filter bank. Thus the wavelet packet transform will be used to refer to the full wavelet packet transform. The filter bank for such a full wavelet packet transform, is shown in Figure 3.19, where we show the transform for three levels.

Notice that for choice of L and H that is in accordance to some discrete wavelet, then we know that the two-channel synthesis bank exists and we can find the full synthesis bank of the wavelet packet transform by a simple level-wise synthesis.

A possible advantage of these wavelet packets approach is that it further decomposes high-frequency coefficients. This could therefore allow for better analysis of these high-frequency coefficients.

3.4. Two-dimensional wavelets

Using the previous section, we have gained an understanding of how the one-dimensional discrete wavelet transform works. However: our signals of interest, namely images, are not one-dimensional,

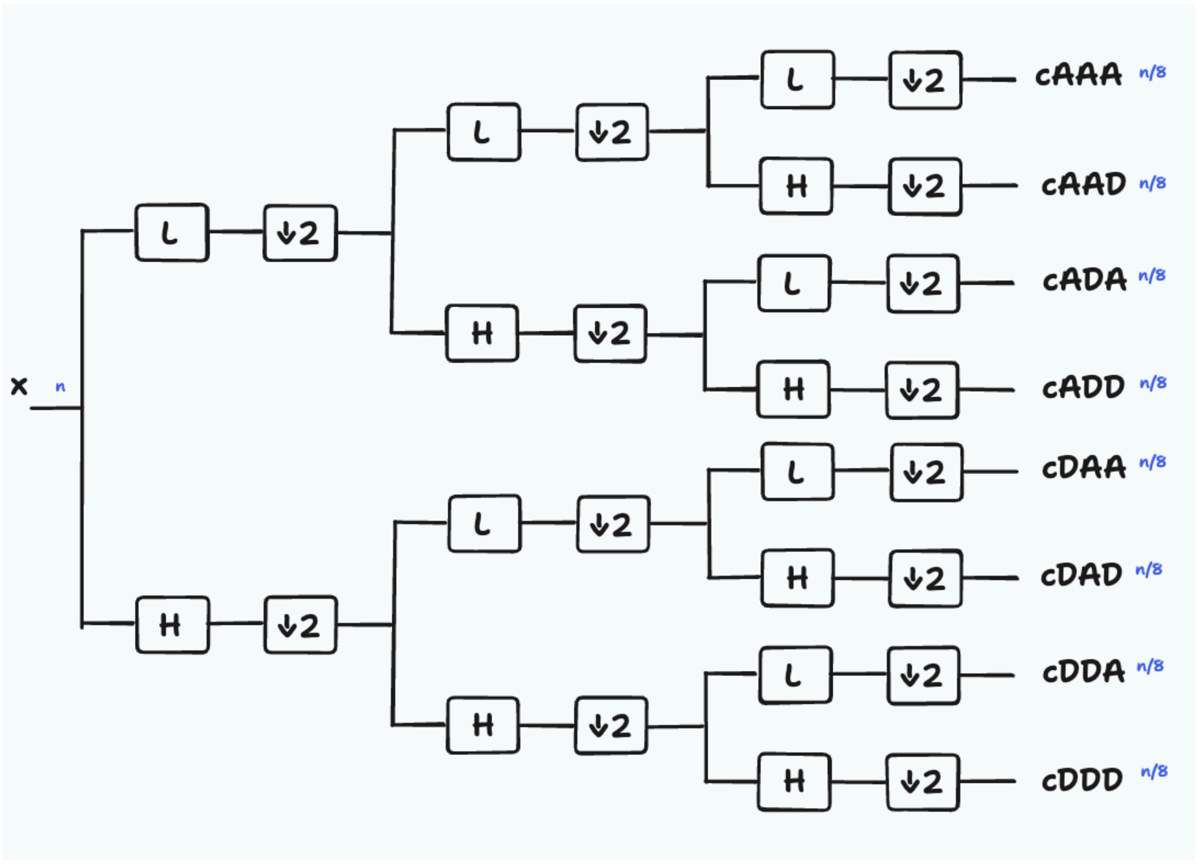


Figure 3.19: Wavelet packet transform of three levels. Note how every output coefficient has size n/8 and is characterized by the order of filters it has passed through. For example, cADA are the coefficients that have first been low-passed and downsampled, then high-passed and downsampled and finally that have been low-passed once again and downampled.

rather, they are two-dimensional. In this section we will explain how the one-dimensional wavelet transform can be used to decompose a two-dimensional signal similar to approximate coefficients and detail coefficients.

3.4.1. Two-dimensional DWT

To decompose a two-dimensional signal, such as a grayscale image of dimension nxn using the discrete wavelet transform, we can simply do the following. First, we apply the one-dimensional wavelet transform on all rows of the signal. Then, for any row of an image, the row exists out of cA and cD coefficients. We can stack all of the coefficients on top of each other as to form a two-dimensional signal again, but one that has half the width of the original signal. We have visualized such a transform in Figure 3.20, where we use the Haar-wavelet as our one-dimensional wavelet transform and where we use a grayscale alpaca image as our two-dimensional signal. Note that the output images of the one-dimensional DWT over the rows, are two signals with half the width of the original image, where the output of the approximation coefficients seem like a squished version of the original image, and the detail coefficients seem to clearly show horizontal edges, where especially the end of the background and start of the alpaca fur is noticeable.

After applying the one-dimensional DWT on the rows of the original image, we additionally apply one-dimensional DWT on both the approximation coefficients and the detail coefficients. This time, we apply this over the columns, such that we will get four square signals in total. The final column in Figure 3.20 corresponds to these final signals. Notice how the resulting image of taking the approximation coefficients over both the rows and columns corresponds to an approximation of the original image, which is four times smaller than the original image. The second image, corresponding with taking the average over the rows and taking the details over the columns, shows vertical edges, especially noticeable in the background, where the image switches from sky to a forest-esque background.

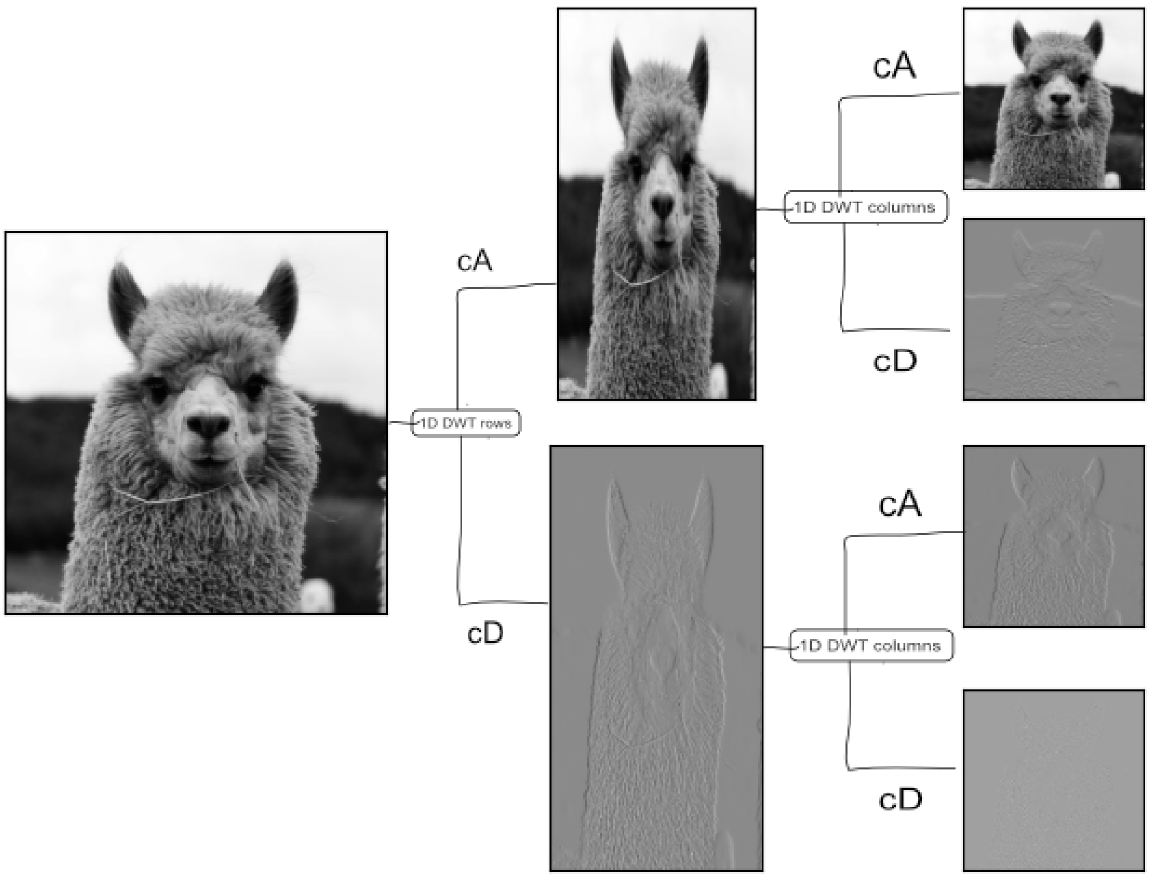


Figure 3.20: Example of how a two-dimensional DWT can be done using three one dimensional transforms.

The third image in the column is the result of taking the details over the rows and afterwards taking the average coefficients over the columns. Here we can clearly see the vertical edges that mark the end of the background areas and the start of the alpaca. Besides that, it is interesting to see that the horizontal edges that were apparent in the switch from sky to forest background are not visible at all here.

It is also notable that in both images the fur of the alpaca is apparent, although in very different ways: The image focusing on horizontal edges, shows a lot of edges around those parts of the alpaca fur where the hair is horizontal, e.g. around the eyes, whereas the image focusing on the vertical edges shows more edges around the parts of the alpaca fur which are more vertical, more around the neck of the alpaca below its face.

The last image of the final column corresponds to taking detail coefficients over both the columns and rows. These coefficients correspond to places where there is both variation in the horizontal and vertical directions, thus capturing fine-scale variation. It is noticeable that these coefficients capture only the fur of the alpaca, because other parts of the image do not have simultaneous variations in both directions.

Rather than decomposing a signal into two signals, the two-dimensional discrete wavelet transform decomposes a signal into four signals, exactly the four elements in the final column of Figure 3.20. The signal following approximations over both axes, is referred to as cA. The signal following approximation over the rows and details over the columns, is fittingly referred to as cV, where V refers to the vertical edges that this signal displays. The signal following details over the rows, but approximation over the columns, is referred to as cH, because it is suited to display horizontal edges. Lastly, the signal following taking approximation over both axes, is referred to as cD, since it displays diagonal edges.

Note that by the invertability of the one-dimensional DWT, we can get the original two-dimensional signal given cA, cH, cV and cD. Given that we have the decomposed coefficients on the right on Figure 3.20, we can apply the inverse DWT on the columns on the coefficients cA and cV to get the image on top of the middle column of Figure 3.20. Similarly, we can get the bottom image of Figure 3.20 by applying the same inverse DWT on the columns of the coefficients of cH and cD. Finally by applying the inverse DWT on the rows of the two resulting signals, we can get our original image back.

Parallel to the one-dimensional case, we can once again recursively decompose the approximation signal, cA, after a single two-dimensional DWT. this would decompose cA into four different signals cAA, cAD, cAH, cAD respectively. Of course, it is possible to apply the two-dimensional DWT on cAA and so on. This has the possible advantage of possibly allowing for analysis of signals on different scales: It might very well be that some structures only appear in a certain level of the wavelet decomposition.

For visualization purposes, an additional transform is used to display the wavelet coefficients, namely, rather than just using the coefficients cD, we take the absolute value of the coefficients, add one and apply a logarithmic transform. Since the wavelet coefficients can become quite large, this scaling causes differences to be more visible.

Moreover, noticing the fact that a single DWT step decomposes an image into four smaller images that are exactly half the size of the original image in both height and width, we can concatenate these four smaller images in the following form as to get an image back. This has been visualized in the left part of Figure 3.21. The coefficients have been concatenated in such a way that the cA and cH coefficients compromise the first row and the second rows is exactly the cV and cD coefficients. Moreover, on the right of Figure 3.21, we can see how recursive application of the DWT steps on the approximation coefficients can also be visualized as a two-dimensional signal: We just replace cA with the four signals cAA, cAH, cAV and cAD that cA decomposes into. Of course it is possible to recursively apply the transform on the newest level approximation coefficients and replace the corresponding coefficients in the signal with the four new decomposed signals.

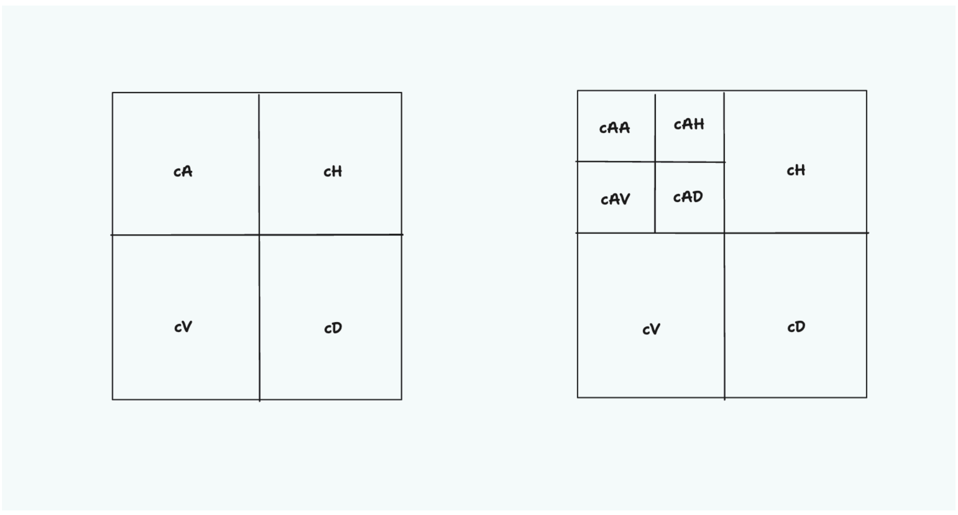


Figure 3.21: Visualization of the common convention of representing the wavelet-coefficients as a two-dimensional signal by means of arranging the signals into a specific pattern. On the left we see this convention for a single step of the DWT and on the right we see it for two steps. Notice how cA is placed in the left-upper corner, cH in the right-upper corner, cV in the left-bottom corner and cD in the right-bottom corner on the left. On the right, we see that this same structure follows, except for in the upper-left corner, where the decomposition has been done recursively on the cA signal.

3.4.2. Two-dimensional wavelet packet transform

The wavelet packet transform also has a two-dimensional counterpart. However, since we every signal gets decomposed into four channels rather than two, we can apply this one-level decomposition on every one of the output signals, cA, cH, cV and cD. Once again, we will only consider full wavelet packet transforms until a certain level. That is we decompose every output signal using our four-channel analysis filter bank as shown in Figure 3.20. Notice that, similarly to in Figure 3.19, we will end up with a number of coefficients that can all be characterized by which specific filters have been applied to them.

The three-level wavelet packet decomposition refers to the following transformation: We take our original signal, decompose it, then decompose every signal that comes out of the first step and then decompose every signal that came out of the second decomposition. We have visualized this in Figure 3.22. For sake of comparison, we have added the regular three-level wavelet decomposition. Notice that there are exactly $4^3 = 64$ different coefficients. Moreover, everyone of these 64 representations, is exactly the same size.

It is important to note that there is no set-in-stone custom on how to order these 64 inside the larger image. In this thesis, whenever an image representation of wavelet packets is shown similar to the right image of Figure 3.22, then the order shown in image Figure 3.22 will be the oe used.

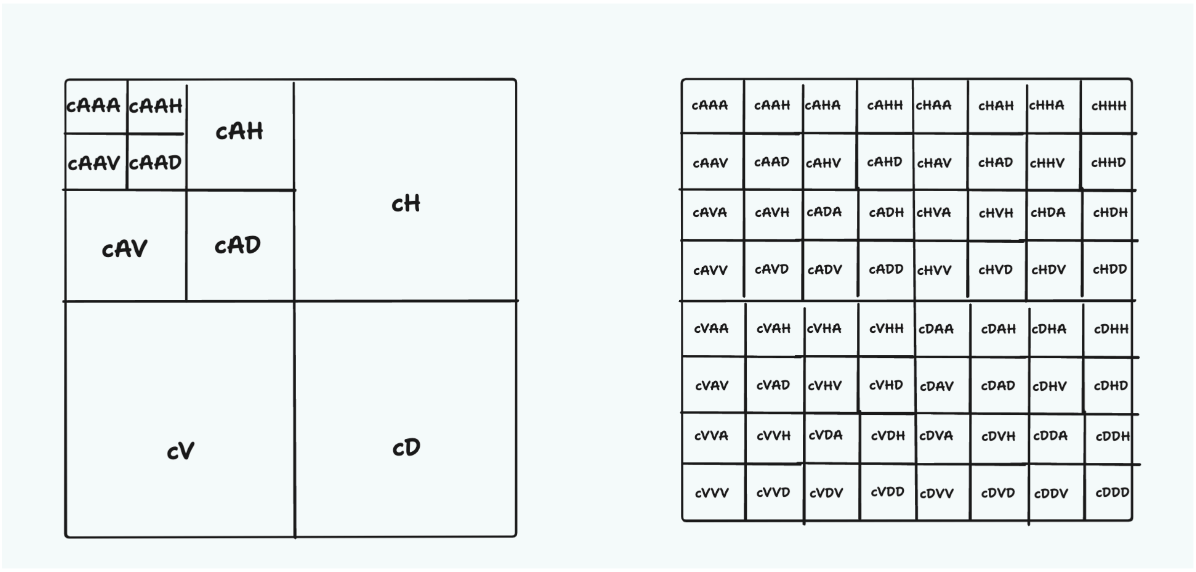

cAAA
cAAH
cAAV
cAAD
cAH
cAV
cAD
cH
cV
cD
cAAA cAAH cAHA cAHH cHAA cHAH cHHA cHHH
cAAV cAAD cAHV cAHD cHAV cHAD cHHV cHHD
cAVA cAVH cADA cADH cHVA cHVH cHDA cHDH
cAVV cAVD cADV cADD cHVV cHVD cHDV cHDD
cVAA cVAH cVHA cVHH cDAA cDAH cDHA cDHH
cVAV cVAD cVHV cVHD cDAV cDAD cDHV cDHD
cVVA cVVH cVDA cVDH cDVA cDVH cDDA cDDH
cVVV cVVD cVDV cVDD cDVV cDVD cDDV cDDD

Figure 3.22: On the left: Three-level wavelet decomposition. On the right: Three-level wavelet packet decomposition

We have visualized the effect of the three-level wavelet packet decomposition using the Haar-wavelet in Figure 3.23. The wavelet packets seem to be an eight by eight tile of all resized representations of the original image. Notice how this makes interpretation more difficult than in the case of the regular wavelet decomposition: What does it mean for a packet to be the resulting set of first using the diagonal coefficients, then taking the horizontal coefficients and finally taking the vertical coefficient $cDHV$? This is quite unclear, whereas for the regular wavelet decomposition we can understand all coefficients as being either an average or detail coefficients in some direction on a different level.

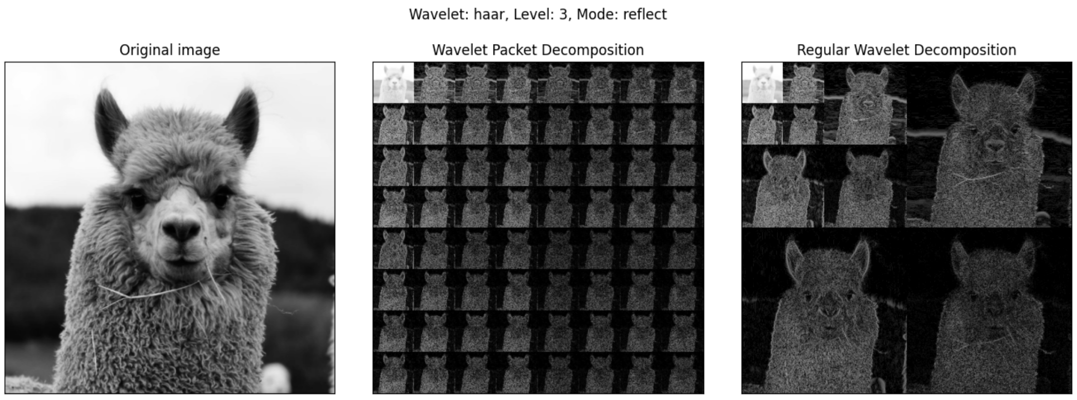

Wavelet: haar, Level: 3, Mode: reflect
Original image
Wavelet Packet Decomposition
Regular Wavelet Decomposition

Figure 3.23: An image, its three-level wavelet packet decomposition and its three-level regular wavelet decomposition. The Haar-wavelet was used. Both transformation representation have been log-scaled. The order of the wavelet packet decomposition is exactly equal to the order described in Figure 3.22.

4

Literature Review of the synthetic image detection field

To work towards our goal of analysing wavelet-based techniques for synthetic image detection, it will be instructive to spend attention to previous research done in the field of synthetic image detection. The importance of this section is twofold.

Firstly, it will allow us to learn how we can investigate the usefulness of wavelet-based techniques for synthetic image detection by considering how previous literature does this. We will see that the field of synthetic image detection uses experiments as its main tool to show usefulness of different techniques for the goal of reliably detecting synthetically generated images. We will let us be inspired by these previous done experiments for the design of our own experiments.

Secondly, our analysis of the synthetic image detection field will allow us to gain a better understanding of discrepancies between synthetic and real images. We will see that these discrepancies give additional reasons to why we should consider approaches that consider frequency aspects of images, such as our proposed wavelet-based approaches.

We will first give a broad literature review of relevant works in the synthetic image detection field. Afterwards, we will investigate frequency-based techniques for synthetic image detection more in-depth. One specifically important paper which we will analyse, is the paper by Frank et al. [13]. Afterwards, we will consider literature on wavelet-based approaches. For which the paper by Wolter et al. [48] is the first to utilize multi-level wavelet-based approaches for synthetic image detection. We will consider their work in-depth. We cannot finish this section without properly understanding the relevant criticisms and limitations on the field of synthetic image detection. Finally, we will conclude this section by stating our key findings and how we will use these findings in the analysis of wavelet-based techniques for synthetic image detection.

Before starting, we should introduce exactly what we mean with synthetic image detection. We will do this shortly in the upcoming section.

4.1. Definition of synthetic image detection

Synthetic image detection is a binary classification problem. We have a set of objects which we want to divide in one of two classes, those being synthetic or real. The specific definition of what we use for these classes, is quite important.

There are many ways images can be manipulated synthetically. We will use the following definition, which is mentioned as content generation in the work of Verdoliva [45].

Definition 4.1.1 (Synthetic and real images)**.** An image is referred to as **synthetic** if it is the end-product of some kind of model which takes as an input noise and outputs an image, such as a GAN or a DM. All images which were not created as the process of such a process, are referred to as **real**.

We should clarify that this definition does not separate images shot of e.g. a natural landscape and an image created by a digital artist in terms of what is synthetic or real. As long as an image is not the end-product of a noise-to-image model such as a GAN or a DM, we refer to it as real. In the scope of this thesis, we will only be considering real images which were shot by some sort of camera due to availability of data. Moreover, the relevance of detecting such images is higher, as can be understood from the Pentagon bomb example Figure 1.1.

Some interesting corollaries arise from this definition. One of those is the following: The definition does also not properly deal with the question on images which are partially synthetically generated. An example of this, is the “inpainting” possibilities which numerous synthetic image generators offer [34, 37]: Given a real image, a prompt can be used to adjust the image to add additional objects in the image. Should this image be classified as synthetic or real? What if we only use the inpainting to change a single pixel value? This quickly becomes a ship of Theseus-esque thought experiment.

The reason why we choose to proceed with this definition is that we can try and get insights into the characteristics of the generating models and what traces they might leave behind in the synthetic images relative to other non-synthetic images.

4.2. Literature Overview

The majority of the literature focussing on synthetic image detection, has been focussed on GAN-generated images instead of DM-generated images. This is because Generative Adversarial Networks have enjoyed a lot more attention and usage prior to the paper by Dhariwal et al. [8], where it was shown that Diffusion Models outperform Generative Adversarial Networks in terms of image sample quality.

The question to what extent synthetic image detection results that were obtained on GANs, generalize to Diffusion Models, is highly relevant because of the relative large body of work done for GANs and has been considered by Ricker et al. [35]. Ricker et al. found that detectors trained on GANs that claimed to be universal, failed to effectively recognize DM-generated images. They do speculate, however, that patterns contained in GAN-generated imagery, are also contained in DM-generated imagery, but in a less prominent fashion. Thus, they conclude that the development of synthetic image detectors that work well on DMs, should also work well on GANs. In the remainder of this chapter we will focus on synthetic image detectors based on GANs, unless otherwise specified.

GANs can create some obvious discrepancies which can easily be noted by visual inspection. These visual discrepancies have been used to create easy-to-implement deepfake detectors in the domain of synthetic faces, such as in [30] where features such as iris radii across the two eyes of a face and the geometrical modelling of teeth are used. The obvious limitation of this type of classifier is that they are heavily domain dependent.

Another interesting branch of GAN-based synthetic image detection is based on the discrepancies in terms of how color is synthesized. An example of this can be found in [25], where the authors use a high-pass filter to extract residuals of an image and conclude that there is a systematic mismatch between some colour-based correlation metric of images, which is especially notable for the chrominance components of the YCbCr color space.

Lastly, there is a branch of SID which focus on discrepancies in terms of mismatches with regard to frequencies in images. Similar to how digital cameras leave behind Photo-Response Non-Uniformity patterns, GANs can leave behind fingerprints, GANs are found to leave behind artificial fingerprints over multiple datasets [50]. These fingerprints can, for example, be observed as a checkerboard-esque pattern in the DCT domain and can thus be leveraged to create effective classifiers [13].

Since wavelets allow for both spatial and frequency analysis of signals, the branch of SID focusing on frequencies is most interesting to us. We will discuss frequency-based techniques for SID in-depth in the upcoming section.

4.3. Frequency-Based Techniques for synthetic image detection

The first time a frequency based perspective was considered for synthetic image detection, was by Zhang et al. [51]. They showed that unique artifacts are introduced in the frequency domain by the deconvolution up-sampling component of GANs. These artifacts are visible as a grid in the frequency domain. These findings were reproduced by Wang et al. [46]. Frank et al. [13] built further on their works in two ways, firstly by showing that these artifacts are also found in DCT-based representations and secondly by showing that direct training of images on the frequency domain can lead to higher accuracy and more efficient learning over pixel-domain representations. We will dive deeper into their work in Section 4.3.1.

An interesting discrepancy between the works of Zhang et al. and Frank et al. on the one side and that of Wang et al. on the other side, is the question on whether to apply the DFT or DCT on the whole image or on the residuals respectively. Residuals are the result of applying a high-pass filter on the image. Zhang et al. and Frank et al. apply the first approach, while Wang et al. use the second approach. This latter approach is inspired by photo-response non-uniformity-based (PRNU) forgery detection, first proposed by Lukavs et al. [28], where PRNU noise, which can be thought of as a camera fingerprint, is used to detect whether some image is digitally forged. This process has been visualized in Figure 4.1. Notice that for such PRNU-based approaches, the exact choice of denoising is highly relevant. Moreover, such methods could possibly allow for more generalizable classifiers, as the residuals are less domain-dependent than pure image representations are.

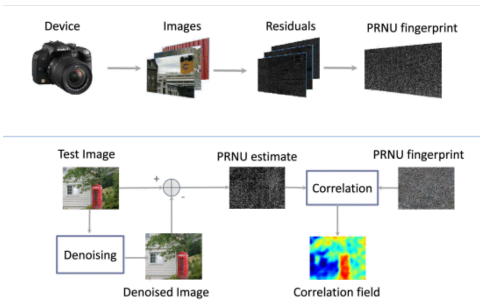


Figure 4.1: Example of PRNU-based analysis of images by taking a set of images captured by a camera, finding the residuals of those images, a PRNU fingerprint can be created. Afterwards during test time, we can test to what extent the residuals of the test-image correlate with our PRNU estimate to get an idea of how likely it was that this picture was taken by a device. Taken from [45].

Other works in frequency-based approaches to synthetic image detection, showed that besides these grids appearing in the frequency domain, GAN-based generative images in addition have noticeable discrepancies in replicating the attributes of high-frequencies [12]. Some work shows that upsampling methods cause these discrepancies to appear [11], but other work shows that this is not enough to fully explain these discrepancies [38].

Moreover, recent work has found the same higher-frequency discrepancies to also hold for DM-generated images [35]. This indicates that these high-frequency discrepancies might be an interesting avenue of further research, because of the fact that both GANs and DMs have noticeable discrepancies in this regard. A possible universal synthetic image detector could make use of such a common discrepancy.

In conclusion, the literature shows that the frequency domain can be useful for synthetic image detection. The high-frequency discrepancies are especially promising, due to their prominence in both DMs and GANs.

4.3.1. DCT-based SID in GANs: Frank et al. [13]

The motivation of the work of Frank et al. lies in their observation that GAN-generated imagery “*exhibit severe artifacts that can easily be identified*” [13]. The artifacts that they are referring to, are visualized in Figure 4.2, which shows an example of the mean discrete cosine transform (DCT) spectrum of both the Flickr-Faces-HQ (FFHQ) dataset [21], which are real images of faces, and the mean DCT spectrum of StyleGAN [21], a type of GAN, trained on the FFHQ dataset. More specifically, to produce the leftmost image, Frank et al. have taken 10,0000 real images of the FFHQ dataset, grayscaled all of those images and taken the DCT transform of every of those 10,000 images and plotted the mean DCT over the batch-dimension. The same procedure has been done on 10,000 synthetic images that were generated using StyleGAN [21].

It can be seen that there are peaks in the StyleGAN, generated spectrum, which have no corresponding peak in the FFHQ spectrum. We have tried to recreate this exact image, but were not successful in doing so. According to the original author of the paper, this could be explained by the fact that the images used in their experiments were trained using a Tensorflow implementation of StyleGAN, whereas the pretrained FFHQ dataset which is widely used online, is implemented with PyTorch, which could possibly implement image resizing techniques differently, which can have quite big effects

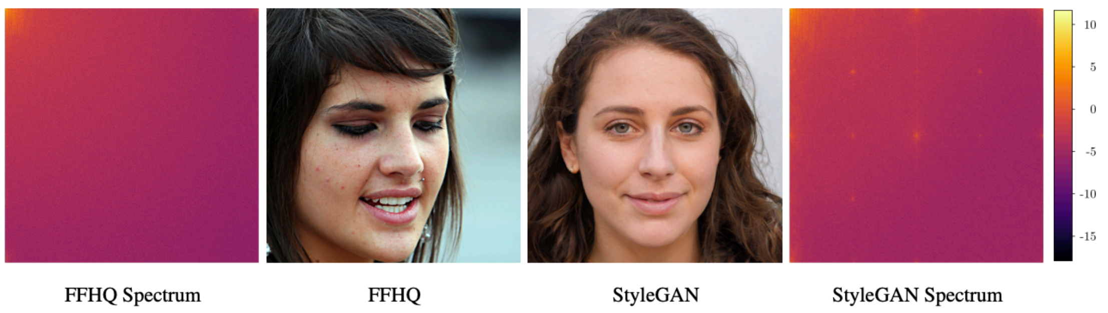


Figure 1: **A side-by-side comparison of real and generated faces in image and frequency domain.** The left side shows an example and the mean DCT spectrum of the FFHQ data set. The right side shows an example and the mean DCT spectrum of a data set sampled from StyleGAN trained on FFHQ. We plot the mean by averaging over 10,000 images.

Figure 4.2: Figure taken from [13], where it can be seen that there are artifacts in the frequency domain of GAN generated images.

in generative modelling [33]. However, even though we could not visually recreate this image, this does not mean that these differences are not useful for classifiers.

Frank et al. [13] showed that a linear classifier can easily learn to recognize this checkerboard-esque pattern and achieve linear separability - i.e. test set accuracy of 100% - for classifying FFHQ images versus StyleGAN generated images at a resolution of 1024 x 1024. Using the same regression classifier on raw pixel values gives a test accuracy of just 76%. In a subsequent experiment, it is shown that a simple regression classifier does not achieve full linear separation when the resolution of images is less (i.e. 256 x 256) or when more elaborate upsampling mechanisms are used. A Convolutional Neural Net (CNN) classifier can still reach 100%-test accuracy in these more challenging scenarios.

Frank et al. afterwards turn their attention to **source attribution problem**. The source-attribution has been introduced by Yu et al. [50] and concerns itself on the following question: Given images that are real, or are generated with one out of four different type of GAN models we have trained: Which out of these five distinct classes does the image belong to (Real, from the first GAN-classifier, the second GAN classifier etc.). That is: given an image predict whether it is real or generated by one of four selected GANs.

These source-attribution experiments are conducted on two different datasets, The first dataset they use is the LSUN[1] Bedroom [49] which is a dataset of bedrooms, which we will also use and dive deeper into in a later section. The second dataset they use is the CelebA dataset [27], which is a face dataset based on images of celebrities. The resolution of images used in these experiments is smaller than before, namely 128 x 128. Frank et al. conclude that the usage of frequency domain images over pixel domain images consistently improves the accuracy of the classifiers.

Frank et al. argue that the reason for why their frequency-based approach works so well, is due to the fact that GANs usually have a generator for which the noise generating distribution has a way lower dimension than the object of interest. Thus, to create an higher dimensional image based on this lower dimension latent space, we need to necessarily apply some kind of operation to get to this higher dimension, which can be done by means of upsampling operators.

4.4. Wavelet-based Approach for synthetic image detection

If we are to recall that both GANs and DMs fail to recreate high-frequency content of real images and recall how wavelets can be used as a multi-scale analysis tool, the question arises to what extent wavelet-based representations can be useful for synthetic image detection. Moreover, wavelet-based representation allow for conservation of spatial relations, which are not considered in frequency-based representations, which could be of interest for partial synthetically generated images, such as the in-painting possibilities offered.

Tang et al. [43] are the first to consider using wavelets for synthetic image detection. Their approach only considers a single level approach, which does not make use of the multi-scale potential of wavelets. Moreover, they only use wavelets as a pre-processing step with the goal of analyzing

[1]Stands for Large-scale Scene UNderstanding.

spectral correlations between color channels of images.

There is only one paper which concerns itself on the usage of multi-level wavelet-based approaches for synthetic image detection. This paper, written by Wolter et al. [48], specifically investigates the use of wavelet-packets to this end, without considering the more efficient and easier interpretable regular wavelet approach. In the upcoming section we will dive deeper into their paper.

4.4.1. Wavelet-packet-based SID: Wolter et al. [48]

Wolter et al. motivate their usage of wavelet-packets for synthetic image detection by noticing that this approach allows for a level of conservation of spatial relations, that is not possible for Fourier transforms. Since Fourier transforms are based on sine and cosine waves, which have infinite energy, the Fourier-based representation does not tell us where in a signal these frequencies are most apparent. Wavelets, on the other hands are localized waves that drop to zero rather than oscillating forever. This property allows wavelet-based representations to not only represent frequency information, but it also allows for connecting this frequency information to the space where it is most fitting.

Note that the combination of spacial and temporal relations can allow for new types of analysis and insights, which are not possible with just either of the two. Such analysis could tell us for example that generated images have a systematic mismatch in terms of frequencies in certain areas of images.

Figure 4.3 shows an example of such an analysis. By considering the mean and variance ln-scaled three-level Haar wavelet-packet representation of 10,000 FFHQ images vs. 10,000 StyleGAN images that were generated by training on the FFHQ images. We can see that the synthetic images show overestimate the high-frequency content in the background in this setting. Moreover, the variance in the background of real images is significantly higher than for the generated images. These images were created by transforming all 10,000 individual elements to the wavelet-packet representation, then calculating the mean and variance of both the real and synthetic images and then comparing these values by considering how much they deviate from one another.

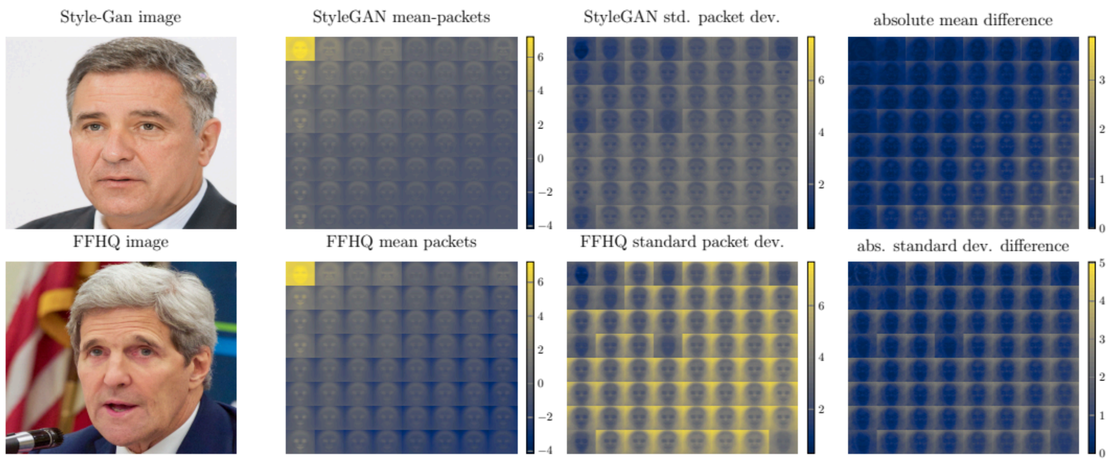


Figure 1: This figure compares single time-domain images and absolute ln-scaled mean wavelet packets and their standard deviations. The first column shows two example images, the second mean packets, the third standard deviations, and the fourth mean and standard deviation differences. Mean values and standard deviations have been computed on 5k 1024×1024 FFHQ- and StyleGAN-generated images each. The order of the packets has the frequency increasing from the top left to the bottom right. We observe significant differences. The mean is significantly different for higher frequencies and at image edges. The standard deviations differ in the background across the entire spectrum. In the supplementary, Figure 22 shows the exact filter sequences for each packet. Best viewed in color.

Figure 4.3: Figure by Wolter et al. showing their motivation for using wavelet packets for the goal of synthetic image detection. Taken from [48]

Following these observations, Wolter et al. try to linearly separate StyleGAN packets and FFHQ packets. To this end, they use 63k training images, 2k validation images and 5k testing images per class. Each image is of size 128-128. Using a simple linear classifier, they achieve a mean test accuracy of 99.75% when trying to seperate the images in the Haar packet coefficient space, whereas the test accuracy for the same classifier trained on raw pixels, is just around 83%.

After linear separation, Wolter et al. move their attention to a more standardized problem, for which they use the source attribution problem, exactly reproducing the setup from Frank et al. To this end, they introduce a CNN based on raw-pixel values and CNNs that are trained on different wavelets. Interestingly, they find that the Haar wavelet packet approach, is outperformed by the DCT-CNN used by Frank et al. but that other, more complex wavelets such as db3 and db4 perform better on average than DCT.

Wolter et al. additionally compare their wavelet-packet approach on Diffusion Model generated imagery by designing the following experiment: LSUN bedroom images are used as the real images, and synthetic images generated by DDPM, a type of Diffusion Model, are trained on LSUN bedroom images. The resolution of the images is 128-by-128. Interestingly, a CNN is used for this experiment, and the CNN based on the Haar-wavelet packet representation has 96.75% accuracy, whereas CNN based on raw pixel values just scores 80.03%, most peculiar is the fact that a CNN based on the Fourier coefficients just scores 73.55%, worse than the pixel based approach.

4.5. Conclusion

In conclusion, we have seen that frequency-based techniques for synthetic image detection are very relevant because of the systematic discrepancy in frequency content between synthetic images and real images. Wavelet-based approaches are a related approach whose usefulness for synthetic image detection has been relatively little researched. Only a single paper manages to use a multi-level wavelet-based approach for synthetic image detection. However, this paper focuses on the wavelet packet approach without placing any consideration on the more efficient and more interpretable regular wavelet approach.

Moreover, it is noticeable that authors use linear models and achieve very high accuracies and then immediately abandon these models in favour of more complex models. It would be of interest to consider how far we can take these simple linear models: When will their performance deteriorate? In which scenarios do they work well and when do they fail?

Given both of these ideas, it makes very much sense to stay close to the experiments done by Wolter et al. Since their code is publicly available[2], we can leverage their code to both faithfully reproduce results as well as expand by introduction of new methods. Our changes to the code of Wolter et al. can be found in `https://github.com/kanarian/frequency-forensics`. In Chapter 5 we will dive into our experimental design and into the model used by both Wolter et al. and us.

[2]`https://github.com/gan-police/frequency-forensics`

5 Experimental setup

In this chapter we will zoom into specifics of the experimental setup that is used by in this thesis. To this end, we will firstly consider the datasets on which we are applying the analysis. Afterwards, we will look at the model we will be using for classification, and then we will look at more specifics of the exact training procedure.

5.1. Datasets

One challenging aspect of this thesis was to find datasets. A lot of the papers in the literature assume that we have access to large amounts of computing power and thus it should be easy to generate synthetic images yourself based on just a synthetic image generator and its weights, but not only is this infeasible if you do not have access to a lot of computing power. It is also very unsustainable to spend large amounts of computational power to generate these synthetic images, even more if that amount of compute has already been spent by some researcher to generate these images.

5.1.1. FFHQ and FFHQ-based StyleGAN

The first dataset used in this thesis is the face dataset released by Wolter et al. [48]. This dataset consists out of 70,0000 resized[1] 128-by-128 RGB-coloured Flickr-Faces-HQ faces and 70,000 128-by-128 RGB-coloured StyleGAN generated images, trained on the Flickr-Faces-HQ faces.

To train the classifier, both of the image sets are divided into three disjoint sets: The first being the training set, the second being the validation set and the final set being the test set. The training set is used to update weights of the classifier, the validation set is introduced so that overfitting could possibly be detected and we can stop the training procedure in time and lastly the test set is introduced to get an estimate of how the test set would perform in terms. The training set contains 63,000 FFHQ images and 63,000 StyleGAN images, the test set contains 5,000 FFHQ images and 5,000 StyleGAN images and lastly, the validation set contains 2,000 FFHQ Images and 2,000 StyleGAN images. These settings have been chosen to be the same as Wolter et al. [48], as to stay close to their setting.

As can be seen in Figures 5.1 and 5.2, these images are quite consistent in their form, as they are all images of faces. Also note that the synthetically generated images here are only StyleGAN-based. So if one would like to consider how well wavelets work for the detection of Diffusion Models generated imagery, that would require additional data.

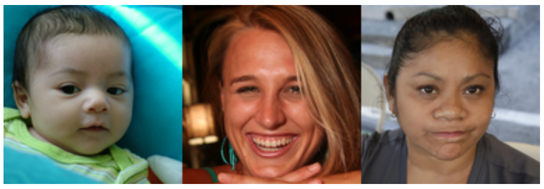


Figure 5.1: Images of FFHQ dataset.

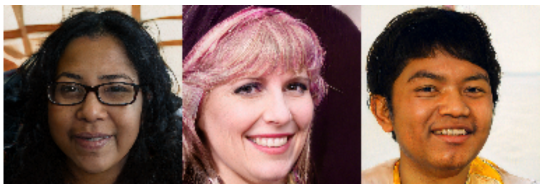


Figure 5.2: Images trained by StyleGAN on FFHQ.

[1]Originally, all FFHQ images are 1024-by-1204.

5.1.2. LSUN Bedroom

The second dataset we consider in this thesis, is the LSUN bedroom dataset released by Ricker et al. [35], which contains 50000 samples for a number of different synthetic image generators, both GAN-based, in StyleGAN and DM-based in DDPM, trained on the LSUN Bedroom dataset. To get 50,000 real samples, we had to make use of the LSUN Bedroom dataset itself [49].

In our experiments, we will only consider DDPM and StyleGAN generated images, as these two models are the two most prominent and widely recognized DM and GANs respectively. Moreover, since the the face dataset also uses StyleGAN images, this allows for a comparison of performance on two different datasets using the same generator.

Since we only have access to 50,000 samples per class, we will use 43,000 images per class in the training set, 5000 per class in the test set and 2,000 per class in the validation set, similar to the setting for the FFHQ dataset. Note that we have just shrank our training set with 2,0000 images.

The LSUN Bedroom dataset contains images of bedrooms, which is a less “regular” set of objects than faces, that is there is way more variety in images of bedrooms than there is in images of faces, as can be seen in Figures 5.3 to 5.5. Moreover, bedroom images might contain more different types of patterns than images of faces, e.g. there might be distinct patterns on the duvet cover and the wallpaper of an image, which allows for interesting high-frequency behaviour of such images, whereas in images of faces such high-frequency behaviour might only appear in the background or perhaps in the facial hair or a haircut of the face.

Figure 5.3: Images of LSUN Bedroom dataset.

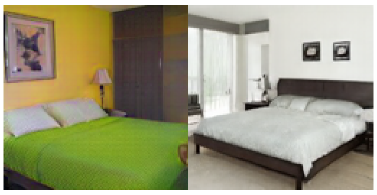


Figure 5.4: Images trained by StyleGAN on LSUN Bedroom dataset.

Figure 5.5: Images trained by DDPM on LSUN Bedroom datasets.

5.2. Linear Model for Binary Image Classification

In this thesis, we will be using the linear model introduced by Wolter et al. In this section we will work towards understanding this linear model. We will show that this model uses linear combinations of the flattened input features to get two values, one representing the probability of the image being real and the other representing the probability of the image being synthetic. To train the weights of this linear model, we will see that the negative log likelihood loss is used. The model has been visualized in Figure 5.6.

We will see that we can add additional interpretation of the weights by means of reordering weights. We will see that our linear model attempts to learn some kind of representative image for each of the two classes and will assign any image to either of the two classes based on how well the image corresponds to the representative image for each class.

5.2.1. Algebraic interpretation

Our goal is to assign a given image, x to one of the two classes, ‘real’ or ‘synthetic’. We can achieve this by creating a function that assigns a real-score and synthetic-score for any image we give it and then assign the image to the highest scoring class. To this end, we will introduce learnable weights W, which we will use to learn which features of the image are more or less important to decide whether an image is synthetic or real.

Our model will be of the following form:

$$f(x, W) = Wx. \tag{5.1}$$

In this equation, we can understand x to be a flattened representation of the image. That means that if our input image were to have dimensionality 3x128x128, then we transform it into a column vector that has 49152 elements.

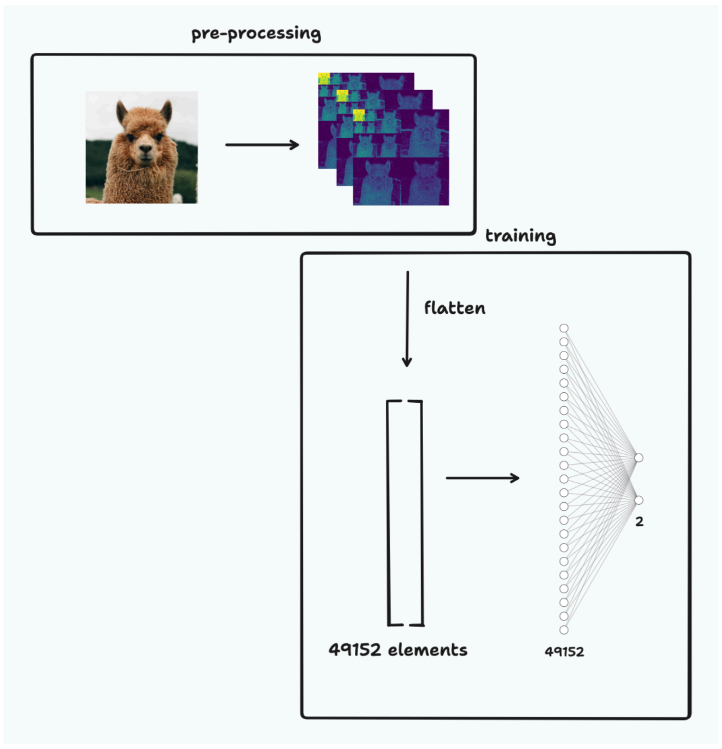


Figure 5.6: Visualization of the training procedure for a regular wavelet decomposition. The image is transformed into three wavelet decompositions corresponding to the RGB color channels. The wavelet representation is flattened and used as input to a linear classifier with 49152 inputs and 2 outputs. The classifier's outputs are linear combinations of the wavelet coefficients, representing the probabilities of the image belonging to the synthetic or real class. A logarithmic softmax function is then applied to the outputs for probabilistic interpretation.

Since we have two classes for which we want to calculate a score, we want the output of the model to give us exactly two values. This, in combination with the given dimensionality of the input image, decides the dimensionality of the weights. If we were to once again assume that our input image is of dimensionality 3x128x128, then W would be of dimensionality 2x49152. Note that the matrix-vector product Wx would then give us a column-vector of exactly two.

A small visualization for how a linear model can assign real and synthetic scores to a 2-by-2 input image is shown in Figure 5.7. Notice how the first, green, row corresponds to the weights for finding the real-score of the image, whereas the second, red row corresponds to the weights for finding the synthetic score of the image. The final scores vector $f(x, W)$ contains the real score and synthetic score. Notice that in this case the first score is higher and thus we would classify the image as being real.

Linear models might have an additional learnable term, b, added to the equation eq. (5.1). In our case, such a term would always be a column vector with two values. This term can act as an offset for any of the output classes. We can incorporate the bias term by adding an additional column to our weights W and always adding an additional one-valued feature at the end of our flattened input vector.

Until now we have not considered how we get the values of the weights, but the whole linear model heavily depends on these exact values for its performance. If the weights are not very good, then the scores will not be very good and consequently our classification will be not very good. In the upcoming section we will explain how we can use data to learn these weights.

5.2.2. Learning weights

To learn the weights of our linear model, we should first start off by thinking of what it means for our model to perform well. To this end, we will introduce a loss function, which is high when our model performs badly and low when it performs well. Since we are in the setting of supervised learning, we have access to a dataset existing out of images x_i, and their labels y_i, that is whether they are real or synthetic. Obviously, we would like our classifier to classify real images as real and synthetic images

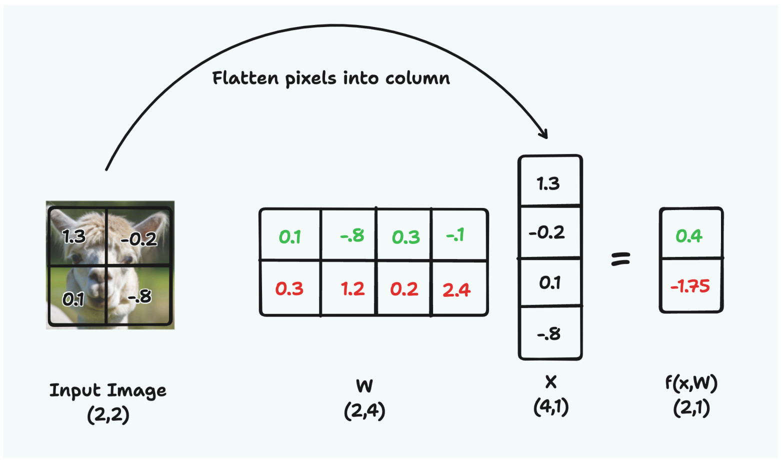


Figure 5.7: Example of how a linear model can be used to assign scores to an image. The image is flattened into a column vector and then a matrix-vector, Wx product is calculated. Note that the first, green, row of W contains the weights for the real score, and the second, red, row contains the weights for the synthetic score. The final scores vector, $f(x, W)$, contains both the real and synthetic score.

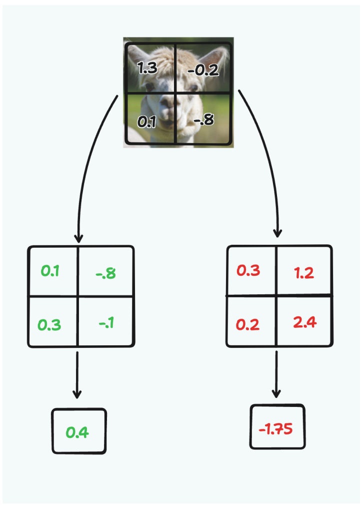


Figure 5.8: Template interpretation of a linear model. Notice how each output classes has a specific template.

as synthetic.

Transforming our outputting class scores to probabilities, allows for a number of benefits. Most notably, this allows us to interpret the scores as some kind of confidence in the prediction: If our mode says that it believes that the image is real with 99% probability, then that might lead us to different conclusions than if it were to say that an image is real with just 51% probability.

This can be achieved by application of the softmax function.

Definition 5.2.1 (Softmax function)**.** Given some vector $z = [z_1, \ldots, z_m]$, where z_i represents the score for class i, the softmax function outputs a vector of probabilities $p = [p_1, \ldots, p_m]$, where p_i is the probability of the input belonging to class i. The softmax function is defined as:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{m} e^{z_j}}, \tag{5.2}$$

for $i \in \{1, \ldots, m\}$.

In the case of Figure 5.7, the softmax operator would yield: $(\frac{e^{.4}}{e^{.4}+e^{-1.75}}, \frac{e^{-1.75}}{e^{.4}+e^{-1.75}}) = (0.90, 0.10)$, showing that the model places 90% confidence in that the image is real and 10% that the image is synthetic.

Our loss of choice will be the following: We will firstly apply the softmax operator on the score-vector, then we will take the probability corresponding to the true class-label. We will then take the logarithm of that probability and finally we use the negative of this logarithm as our loss function. This notion has been formalized in Definition 5.2.2. Optimization of the negative log-likelihood in this setting is equivalent to maximizing the likelihood of our model and to minimizing the KL-divergence between our empirical distribution and our model distribution [14].

Definition 5.2.2 (Negative log-likelihood loss)**.** Given some input-label pair (x, y) and model $f(x, W)$ which outputs a probability vector $p = [p_1, \ldots, p_m]$ where probability p_i corresponds to input x belonging to class i. The probability the model $f(x, W)$ assigns of x belonging to class y is exactly p_y. Now the negative-log-likelihood loss, $\mathcal{L}$, is:

$$\mathcal{L} = -\log(p_y).$$

If we were to have a set on input-label pairs, rather than just a single input-label pair, the negative log-likelihood loss can be calculated similarly per input-label pair and the total loss is just the sum of the individual losses. Moreover, notice how smaller the probability of the input being classified to the correct label by the model, how bigger the loss.

The model we will be using can be seen as a tiny neural network, but one that does not use non-linear activation functions and therefore we will also not use multiple layers. This is because a network using multiple layers without a non-linear activation function reduces to a one-layered, linear model, such as the one we have.

In the upcoming section we will introduce an alternative interpretation of the linear model. This alternative interpretation allows for a more intuitive explanation of what the weights mean and what our linear model tries to do.

5.2.3. Template interpretation

Templates can be understood as quintessential representations of what it means for an image to belong to a particular class. We can reshape every row of our weights to the dimensionality of the inputting image to get such a template. For example, we can reshape the four, green weights corresponding the true-score in Figure 5.7 to be an image of dimensionality 2-by-2. Then we can get the real-score by an dot product between the reshaped weights and the input image. We can do this for both the real and synthetic score, netting us two sets of reshaped weights. This has been visualized in Figure 5.8.

Notice that we are simply taking the dot product of the image with a template. Since the dot product is highest when two elements match exactly, we can understand why these reshaped weights are called templates: For any input image, we compare how well they correspond to the real template and the synthetic template. A better correspondence, gives us a better score. These templates allow us to interpret what pixels of the original image are important for saying an image belongs to either of the two classes.

It is important to understand that we are only using a single template per class and so necessarily, features of classes that are multi-modal are difficult to capture. Imagine for example that our outputting class is not real or synthetic, but e.g. alpaca or football and we would be using raw-pixel values rather than wavelet-representaion-based coefficients. An alpaca can have different colours of fur, but the template can only tell us what an alpaca is supposed to look like by using a single image. Thus the template will not be able to show us that indeed there are multiple colours of fur an alpaca can have.

Since we are training our classifier on wavelet-based representations, the templates will also be templates in terms of wavelet-based coefficients. That means that the templates will learn what kind of wavelet-based representation is most fitting for each of our two classes of interest, real and synthetic.

5.3. Training procedure specifics

In this section we will reflect upon some specifics of the training procedure. That is, first we will look at the pre-processing step we are doing. We will see that we we will not directly use the wavelet coefficients, rather we will use normalized, log-scaled values of the absolute wavelet coefficients. Afterwards, we will consider the training loop. that will exactly show us how the weights of the model are trained.

5.3.1. Pre-processing

Before training, we will apply the appropriate transform (i.e. wavelet packet or wavelet) to each of the images in the training, validation and test set. For both the packet approach and the regular approach, we will mainly use the three-level Haar-wavelet decomposition. Not only is this the choice also made by Wolter et al., but this choice also allows for no boundary issues. When not specified which wavelet is used, it is to be assumed that the Haar-wavelet was used. This pre-processing saves a lot of time during training, as otherwise we would have to transform an image anytime we want to use it. Since we will be doing multiple runs over the whole dataset and thus we will be using the same image multiple times, this approach saves us some time.

Because of the fact that the wavelet coefficients can be of very different scales, we will need to take the natural logarithm to make the coefficients to be of similar order. Since we cannot take the logarithm of negative values and the coefficients can be negative, we will use the absolute values of the coefficients and adding a small ϵ value before transforming, to prevent zeroes from occurring inside the natural logarithm. Mathematically speaking, given that x is a wavelet-coefficient representation of our signal of interest, we will apply the following transform in the pre-processing step:

$$x \mapsto \ln(|x| + \epsilon), \tag{5.3}$$

finally before any coefficient is used as input in the model, the wavelet coefficients will be normalized by subtracting the mean wavelet coefficient over the dataset and by dividing over the variance of that wavelet coefficient over the dataset.

The dimension of our image is exactly $3 \cdot 128 \cdot 128 = 49152$, since it's a RGB image of 128-by-128 pixels. As both the wavelet packet decomposition and wavelet decomposition are orthonormal basis, their dimensionality is also 49152. An interesting consequence of this approach is that both transformed representations of the images take a lot more space than the original image, as every coefficient of the image is an integer ranging from 0-255 and can thus be stored by using 8-bit unsigned integer, whereas the coefficients of the wavelet-based decompositions are floating point values, which need to be stored with 32 or 64-bits.

5.3.2. Training loop

Our weights will be a matrix of size 49153-by-2, since we have inputs of dimension 49152, a bias term and two output classes. In total, we will have 98306 weights. We will specifically train these weights by applying a softmax function over the outputting two classes and then calculate the negative log likelihood loss over a batch of 512 training images, then update the weights by doing a gradient step using the Adam optimizer [23] with step size 0.001. After every 200 training steps, the model is validated by running through the validation set and noting the accuracy. During this validation loop, weights are not updated. If we see that the validation accuracy is 1.0, then we end the training. Our ending condition is either that the validation accuracy is 1.0, or that we have done 20 epochs, which are full passes through the whole training set. When training is finished, we use the test set to get a final estimate of performance of the model.

The full procedure has been visualized in Figure 5.6 for a regular wavelet decomposition. For the wavelet packet decomposition, the procedure is the same, but we use the wavelet packet decomposition in the pre-processing step. We can see that a single image is transformed into three wavelet decompositions, corresponding to the three colour channels. During training, the wavelet representation is flattened and put as an input to a linear classifier that has 49152 inputs and 2 outputs.

Notice that we will be using a training dataset of 126,000 images in the FFHQ case, and a training dataset of 86,000 images in the LSUN Bedroom case. Since the total number of weights is 98,306, we see that the number of inputs is just barely higher, or lower than the number of inputs for the FFHQ and LSUN bedroom datasets respectively. Additionally, we do not use any regularisation. The relative low number of weights in contrast to the number of datapoints and the fact that we are not using regularisation, means that we should be aware that overfitting could occur.

6 Results

In this section, we will seek to answer our questions of interest by means of by running a number of experiments as described in Chapter 5.

We will start off this result section by comparing the StyleGAN vs. FFHQ challenge. For this setting, we will use the wavelet-packet approach proposed by Wolter et al. In addition, we will use our regular wavelet approach and consider performance on this setting. This allows for comparison of the regular wavelets versus wavelet packets. We will see that, in this setting the wavelet packets very slightly outperform the regular wavelets, but both approaches yield very high accuracies. This allows us to

Afterwards, we will consider to what extent this result generalizes to different datasets, where we will use the challenge of detecting LSUN bedroom images vs. StyleGAN generated bedrooms. Here we will see that contrary to our previous experiment, the regular wavelet approach outperforms the wavelet packet approach. Additionally, we will see that both approaches still achieve high performance.

By considering the LSUN bedroom images vs. DDPM generated bedrooms as the next challenge, we can compare the performance of our classifier with the previous experiment to get an idea of what the effect of using a Diffusion Model is. We will see that performance drops quite a bit for both the regular wavelet and the wavelet packet approach. Moreover, we will see that this discrepancy between the two approaches also becomes bigger.

By noticing that most of the experiments already achieve their best performing accuracy quite early on in the training process, we will consider training the model with just 10% of the training data. We will see that performance stays relatively stable. However, since this allows for quicker experiments and therefore quicker iterations, this is quite a nice result. Additionally, by showing that even this relative simple classifier can achieve high performance on a relative small subset of the data, where we are able to extend the notion of robustness of training data reduction found by Wolter et al. in the setting of CNNs to the setting of the much simpler classifier.

Finally, we will finish this chapter with a section with some steps toward generalizability. We will consider performance of the classifiers across the three differente datasets and we will see that for most settings this performance deterioriates to near random performance. By noticing that this might be caused by the domain-reliance of the classifiers, we propose to train our classifiers on the residuals of the images rather than the whole image, which should be less domain dependent. We will see that this does not lead to notable performance increase.

6.1. StyleGAN vs. FFHQ

Notice that this is the experiment exactly done by Wolter et al. [48], where we add the usage of regular wavelet approach. Thus, this section will both act as a reproduction of their work as well as a comparison of the packet method to the regular wavelet method.

6.1.1. Visual mean and variance deviations

We will start off with considering both the means and variances of both representations. Since we have access to the 1024-by-1024 images, we will start off our first analysis with images of this dimension, as has also been done by Wolter et al. Once again, this is done in an attempt to stay close to their experiments and show that even following their argumentation, the regular wavelet approach is an approach that can be useful.

Before applying the wavelet transforms, we grayscale the RGB image. We do this for illustrative purposes, as the analysis per colour channel is similar to our analysis on the grayscaled images. After application of the transforms, we scale the resulting images by taking the absolute values of the coefficients, then we take natural logarithm of these coefficients.

Afterwards, we take both the mean and variance of the log-scaled transformed images. This procedure is done for both real and synthetic images, in this case thus for both images from FFHQ and images generated by StyleGAN based on FFHQ, netting us in the end two means and two variances. Finally, we take the absolute deviation of the means and variances respectively. By looking at the deviations, we might get an idea of what features could be used by a classifier to detect whether an image is fake or real. Mention however, that if we cannot see any deviations, it does not mean that a classifier might not use them.

Figure 6.1 is added to give the reader a better understanding of what the aforementioned process looks like. This figure only visualizes the pipeline for the regular wavelet approach, but the pipeline is analogous for the wavelet packet approach.

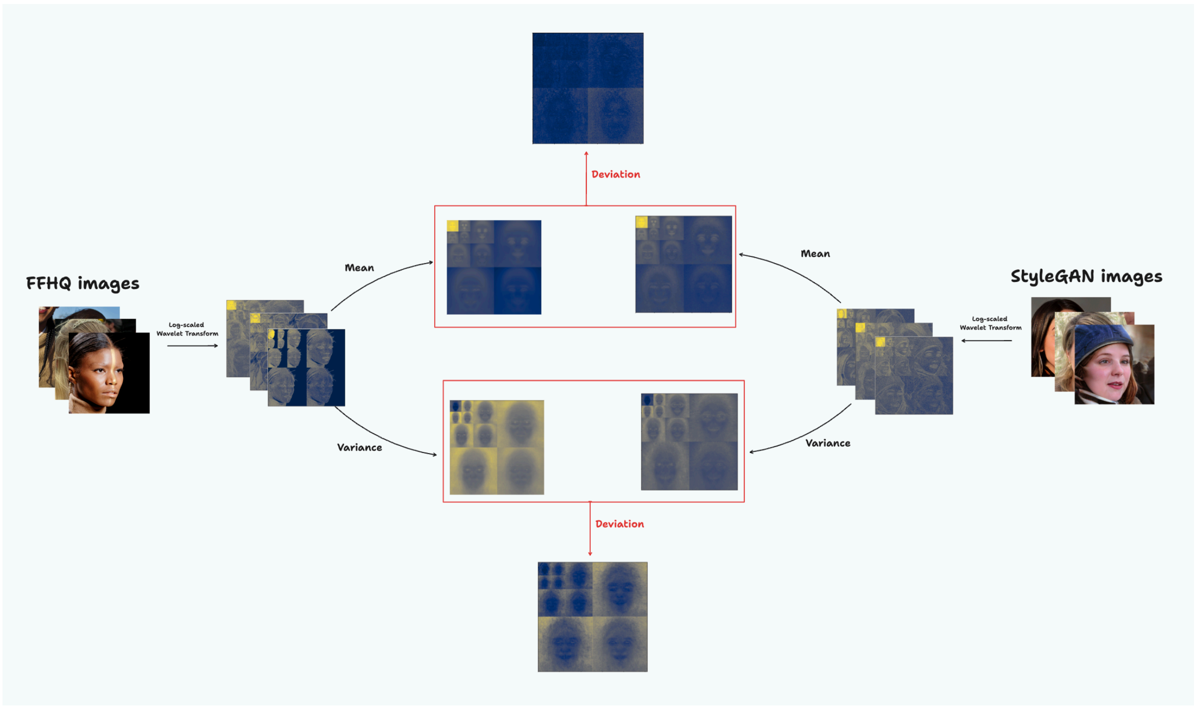


Figure 6.1: Pipeline for creating mean deviation images and variance deviation images for FFHQ images and StyleGAN images using the regular wavelet transform. For both sets of images, the wavelet transform is applied to all images after which the coefficients are then scaled by using Equation (5.3). The mean and variance of both sets of transformed images are then taken and finally the deviation between the real and synthetic means are taken, as well as the deviation between the real and synthetic variances.

Figure 6.2 contains the results of the aforementioned pipeline when using the wavelet packet decomposition. Figure 6.3 contains the results when using the regular wavelet decomposition. We note that Figure 6.2 is a reproduction of the work by Wolter et al. Our results match their results.

There are a number of interesing observations we can make. The first of which is that both the wavelet packet approach and the regular wavelet approach show that in terms of mean deviations, the first-level diagonal wavelet coefficients are the ones that have the largest deviations between real and synthetic datasets. Most interestingly, in the wavelet packet mean deviation, we can see that the most lit up packet, is the packet corresponding to taking diagonal coefficients once and then taking approximate coefficients twice.

If we focus on the variance deviations, we can once again see that both approaches allow us to come to the same conclusion: Most of the variance deviations are in the background of the image rather than in the faces. Notice how the packet corresponding to *cDAA* is the packet with the least variance deviation. Meaning that even though there is a relatively high deviation in terms of means, the deviation in terms of variance is small. Notice that such analysis cannot be done if we were only to consider the wavelet approach.

We can additionally note that the variance in the background of the real images is higher than those of the fake images. That means that images generated by StyleGAN consistently underestimate the variance that appears in backgrounds.

Another interesting phenomenon that we observe, is the fact that synthetic faces seem to have more variance with respect to their teeth, if we look at the variances of real and synthetic images. This is especially noticeable in the first-level vertical coefficients. Note that such analysis cannot easily be made with the wavelet packet based approach. The deviations might still be present, but not easily noticeable by the human eye.

Note that our experimental analysis will be done on images of size 128-by-128, for which we have also added the figures in the Appendix, Figure A.1 and Figure A.2. The results match our described results for the 1024-by-1024 images. As mentioned earlier, this is done because of computational constraints. However, this also shows that even if the images are resized to smaller dimensions, the differences can still be spotted.

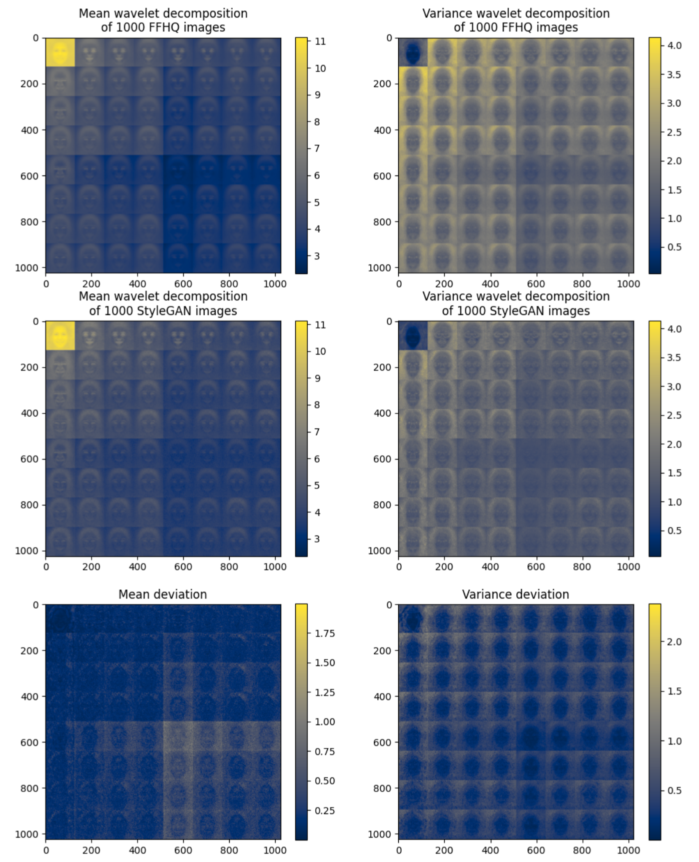


Figure 6.2: Three-level Haar wavelet packet decomposition of real and synthetic images. Real images used are 1024-by-1024 images of the FFHQ dataset, synthetic images used are 1024-by-1024 images generated by StyleGAN, trained on the FFHQ dataset.

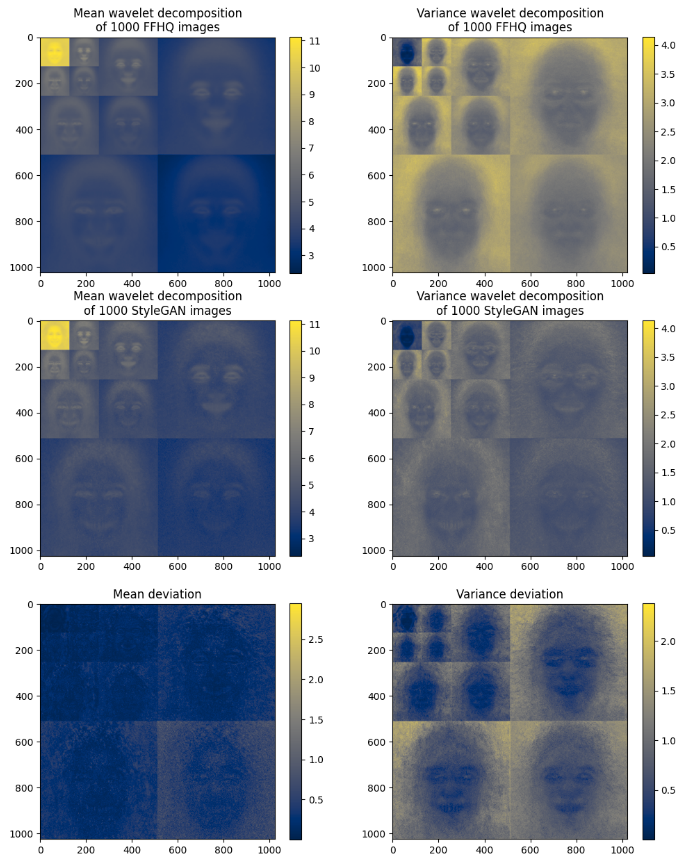


Figure 6.3: Three-level Haar wavelet decomposition of real and synthetic images. Real images used are 1024-by-1024 images of the FFHQ dataset, synthetic images used are 1024-by-1024 images generated by StyleGAN, trained on the FFHQ dataset.

In conclusion, both wavelet-based approaches show systematic differences between real and synthetic faces in terms of these deviations. This means that both approaches could be interesting if we are to use them as input for a classifier. Moreover, we might understand both approaches better if we are to look at how well they perform in terms of classification.

6.1.2. Experimental Results

Figure 6.4 contains the results of our experimental setup. We have run the experiment 5 times with different initial seeds, with the goal of getting a more accurate view of the performance of the model. For the validation accuracy, we have plotted the mean validation accuracy as a line and we have added the minimum and maximum values accuracies of the runs per training steps as shaded areas. For the test accuracies, we have added their mean in the plot as a dot and we show the spread of the values by the vertical line through the dot.

Throughout all of our experiments, we will concern performance in terms of accuracy. This is the ratio of correctly sampled elements to total elements. We will mostly be using test accuracies to compare elements, as this is an indication of how well the classifier would do on unseen data.

Notice how both models get an test accuracy of near 1.0, but the packet approach consistently scores higher: Even the worst test accuracy out of all wavelet packet test accuracies, is higher than the best regular wavelet test accuracy. Interestingly, both approaches seem to start the first validation step with very high accuracy. Moreover, the regular wavelet approach seems to have more variance over the different experiments.

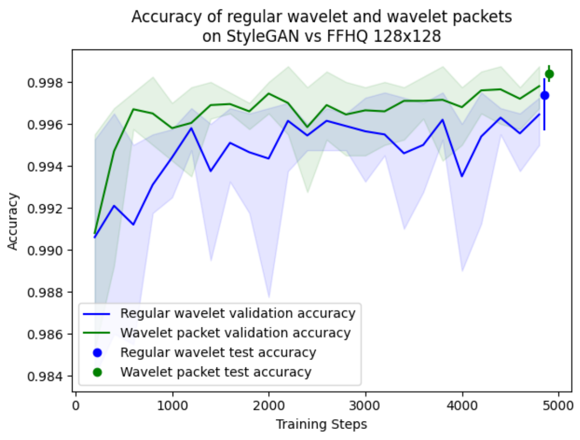


Figure 6.4: Validation and test accuracy for SID on StyleGAN vs FFHQ using our classifier. For both wavelet packets and regular wavelets, the experiment has been run five times, where the means are shown as a line and dot for the validation and test accuracy respectively. The shaded areas and the vertical line through the dot correspond to the range of accuracies found in the experiments.

In other experiments, both of ours and of Wolter et al., we have seen that using raw-pixel values as input rather than either wavelet-based decomposition, gives much lower accuracies, of around 80%. This shows that usage of wavelet-based representations does really offer an advantage over simple usage of raw-pixel values.

Another interesting conclusion of other experiments we have found is that the logarithmic scaling step of both the regular wavelets and the wavelet packets is crucial, otherwise the linear classifier performs similarly to the raw-pixel representation. This can be explained due to the fact that since the scale is so large between the different coefficients, gradient updates are not very useful for finding a better classifier, as the weights would need to scale with the input coefficients, but we are not doing that.

We have also considered whether usage of logarithms on raw-pixel values could be useful, but found this not to be the case. This can be explained because of the fact that pixel values just range from 0 to 255, so not a multitude of different scales, such as the wavelet-based coefficients, and thus not being very useful. This tells us that these wavelet-based approaches truly offer an advantage over just considering raw-pixel values, since they increase the accuracy and the difference can only be assigned to the usage of the wavelet-based representation.

6.1.3. Understanding the important features

Since we are making use of a simple linear classifier, we can simply rearrange the weights of the classifier into the shape of the original image to get an idea of what features are used to come to a conclusion. These are exactly the templates that we introduced in Section 5.2.3.

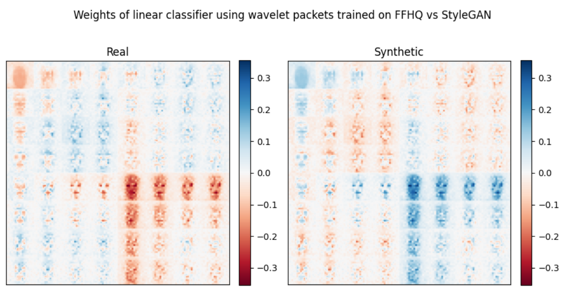


Figure 6.5: Weights learned by linear classifier when applying synthetic image detection on a wavelet packet representation, using three-levels and the Haar wavelet, represented as an image.

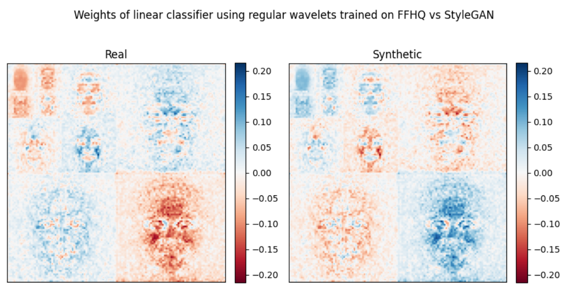


Figure 6.6: Weights learned by linear classifier when applying synthetic image detection on a regular wavelet representation, using three-levels and the Haar wavelet, represented as an image.

We have done this in Figure 6.5 for the wavelet-packet based classifier and in Figure 6.6 for the regular wavelet based classifier. For both figures, we have transformed the weights of the model back to an image, to get a template. We have done this for both output classes, real and synthetic.

The weights are visualized in such a way that zero-valued weights appear as white, negative weights appear as red and positive weights appear as blue. By applying such a visualization, we can more easily understand the importance of elements for classification. Larger values in absolute sense correspond to more important features.

The real weights of Figure 6.5 show that the packet corresponding to taking the diagonal coefficients once and then averaging twice, lights up as red. This should not be very surprising, if we take into account our observations of Figure 6.2, where we also saw this exact packet light up.

Moreover, the same package appears very blue in the synthetic representation of the image. This is because of the fact that our model just learns two classes: Real or synthetic. If some coefficient tells us that presence of it means that the image is less likely to be real, this obviously means that the image is more likely to be synthetic if these coefficients are present. Thus, our addition of both Real and synthetic plots is redundant, we could have made the same conclusions using a single plot, but for the sake of completeness, we choose to add them both. This is only the case because we have exactly two classes.

Notice that is also visible in Figure 6.3, where we see that the first-level diagonal coefficient, of which the *cDAA* is just an approximation, is also mainly responsible for saying whether an image is real or synthetic. Moreover, the coefficients corresponding to the *cDAA* are all implying that the image is fake with a larger scale than can be seen for the first-level diagonal coefficients in the wavelet-based approach. This could be an explanation for why the wavelet-packet based approach works better: In this specific setting, this packet coefficient contains more useful information for deciding whether an image is real or synthetic.

An interesting observation that we can understand from Figure 6.6, is that a multi-scale analysis does actually allow for different type of insights. We can see that coefficients large in absolute value in the lowest-level diagonal coefficients imply strongly that an image is synthetic, whereas coefficients apparent in the second-level diagonal coefficients do the exact opposite: They imply that an image is more likely to be real.

Another interesting observation is that whereas we saw in Figures 6.2 and 6.3 that the mean deviations are most notable in the background of the images, we see in Figures 6.5 and 6.6 that the most important used features are not the background, but rather features in the faces themselves.

The sharp reader has noticed that both figures only contain the results of a single wavelet (packet) transform for both the real and synthetic weights, whereas we would expect to have such images for all three of the colour channels, as we apply the decomposition for any colour of the image, as seen in Figure 5.6. We have averaged over all colour channels to get the weight representations shown in

Figures 6.5 and 6.6. The analysis per colour channel leads to the same conclusions as the analysis over the means.

Moreover, since our experiment yielded five different classifiers each based on a different seed for both the regular wavelet decomposition as well as the wavelet packet decomposition, we have selected just a single classifier for each of the wavelet-based approaches. We do this because of the fact that all of them lead to the same conclusions.

6.1.4. Conclusion

In conclusion, we can say that both approaches can be used to separate faces from the FFHQ dataset with high accuracy. Both approaches rely a lot on the first-level diagonal coefficients for their analysis: the regular wavelet approach directly and the wavelet packet approach in an indirect manner. The wavelet-packet based approach outperforms the regular wavelet approach slightly, which could possibly be explained by the usefulness of the *cDAA* packet.

However we notice that the templates are domain dependent for both wavelet-based approaches: It seems to rely heavily on features in the images that are only apparent for other images of faces. If we would like to use this classifier for another set of images, i.e. detection of synthetic alpaca's, performance will be quite poor, as this classifier relies on facial features to come to its conclusion.

These results lead us to several questions: First and foremost it leads us to the question to what extent such classifiers can be used successfully in different settings than the setting for just faces: Is there a more generalizable discrepancy between synthetic images and real images in terms of these wavelet-based representations that can potentially be leveraged for synthetic image detection? Sine the accuracy of both models is extremely high and we are just using a simple linear classifier, it is of great interest to see if similar results can be found over a different dataset.

Another interesting question is whether the packet approach has higher accuracy than the wavelet approach across different datasets: We have seen that this is the case for StyleGAN vs FFHQ, but does it also hold in other settings? If we can run these experiments and find that indeed this holds, then we can make steps towards validating the assumption of Moritz et al. that wavelet-packet based approaches might be more suitable for discerning synthetic and real images. If we cannot find that this holds, it begs the question what are the specific instances in which either of the two approaches are more suitable.

To gain additional insights into the aforementioned questions, we will consider our second set of experiments, where we compare LSUN Bedroom images with StyleGAN generated images. We will follow the same structure as in this section, first we will look at computed dataset characteristics, then we will look at performance of a linear classifier trained on both synthetic and real images and lastly we will consider what the important features were by looking at the model activation. Once again all of these steps will be done with both the regular wavelet decomposition as well as the wavelet packet decomposition.

6.2. StyleGAN vs. LSUN Bedroom

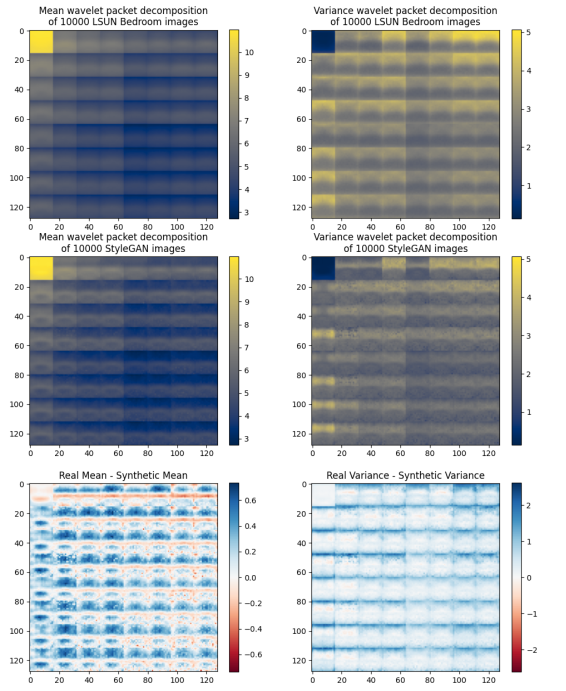
6.2.1. Visual mean and variance deviations

We will firstly analyze the visual meand and variance deviations of LSUN Bedroom images and StyleGAN synthetics based on the dataset. To this end we will use a similar procedure as in Figure 6.1. However, rather than taking the absolute deviations of the representations, we will consider signed deviations, so we can more easily tell which aspects are more or less prominent in real images relative to synthetic images. These signed deviations are shown in the bottom row of both Figures 6.7 and 6.8, where blue corresponds with absolute coefficients being larger in the real mean/variance, and red corresponds to the absolute coefficients being larger in the synthetic mean/variance. Once again, we will apply these analyses for both the wavelet packet as well as the regular wavelet approach, we can find these in Figures 6.7 and 6.8.

One very interesting observation, is how in Figure 6.8 the variance of the wavelet decomposition of the real images, seems to introduce a very prominent grid-esque, which images generated by the StyleGAN seem unable to recreate properly. Rather, StyleGAN produces boxy artifacts in the variance. A possible explanation for the appearance of such a grid-esque structure might be the fact that LSUN bedroom images have been JPG compressed with factor $Q = .75$. Corvi et al. [7] find a similar grid-esque pattern appearing in the auto-correlation function of residuals of the LSUN bedroom dataset, which they assign to JPEG compression. They similarly find that synthetic image generators try to learn these JPEG artefacts with varying success. However, to truly say that our observed artefacts are due to JPEG compression of the LSUN bedroom dataset, further research is needed.

Similar behaviour also is present in the means of the image sets: There is a grid structuring apparent in the mean wavelet representation, which is not visible for the generated images, rather the generated images introduce some blocky pattern, which are especially noticeable in the first-level diagonal coefficients.

All of these observations we made are not as easy to be made in the wavelet packet representations. Perhaps the information is still there, but we cannot easily notice it. To gain more insights into an experiment in the upcoming section where we use both of the approaches to train a linear classifier.



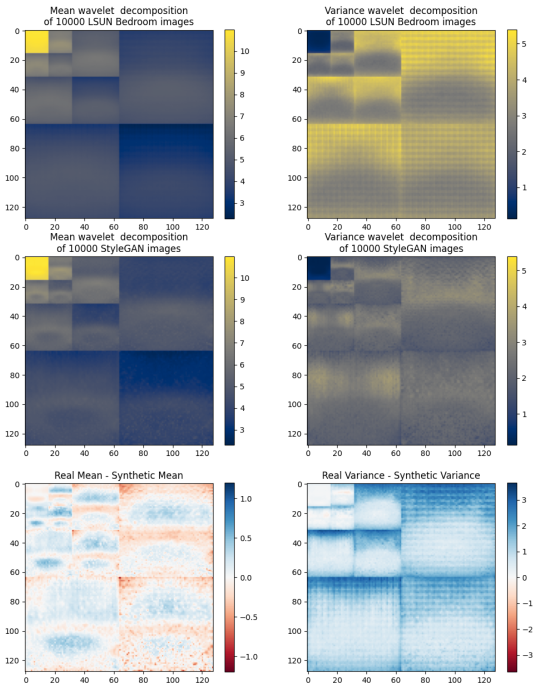


Figure 6.7: Three-level Haar wavelet packet decomposition of real and synthetic images. Real images used are 128-by-128 images of the LSUN Bedroom dataset, synthetic images used are 128-by-128 images generated by StyleGAN, trained on the LSUN Bedroom dataset.

Figure 6.8: Three-level Haar wavelet decomposition of real and synthetic images. Real images used are 128-by-128 images of the LSUN Bedroom dataset, synthetic images used are 128-by-128 images generated by StyleGAN, trained on the LSUN Bedroom dataset.

6.2.2. Experimental Results

Our experimental setting has been described in Chapter 5 and is the same as in Section 6.1.2. That means that we have ran five differently seeded classifiers on both regular wavelet representations and wavelet packet representations of the dataset in a supervised learning setting. We have put the mean accuracy score, worst accuracy score and best accuracy score per training step in Figure 6.9. We have done this for both the validation set over training and also for the test set after training.

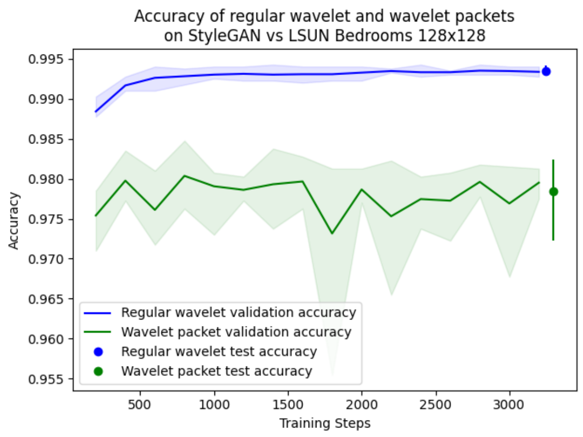


Figure 6.9: Validation and test accuracy for SID on StyleGAN vs LSUN Bedroom using our classifier. For both wavelet packets and regular wavelets, the experiment has been run five times, where the means are shown as a line and dot for the validation and test accuracy respectively. The shaded areas and the vertical line through the dot correspond to the range of accuracies found in the experiments.

Once again, it is quite interesting to see that both of the experiments achieve accuracies that are near perfect, both being higher than 97% in even the worst test run. It is interesting to see that this time, the regular wavelet approach outperforms the packet approach instead of the other way around as in the setting of FFHQ vs. StyleGAN.

The accuracies are a bit lower than in the setting of StyleGAN vs FFHQ. Both approaches consistently had around 99.7% accuracy for the StyleGAN vs. FFHQ approach, showing near perfect performance. In this experiment, the average of the wavelet approach is around 99.4%, whereas the packet approach has an average test accuracy of 97.8%.

We validated that results on this datasets could not simply be achieved by application of the same classifier on raw-pixel values. If the performance on raw-pixel value were already to be 100%, it would not make very much sense to try and use alternative representations to detect synthetic images. In those experiments we found that the performance was around 80%.

The difference in performance of the regular wavelet approach and wavelet packet approach is interesting, since not only is there a bigger gap than in our previous experiment, the order has also been reversed. This experiment shows that there is reason to believe that the wavelet packet approach is not always the superior approach.

This signifies the importance of understanding the different behaviour of both approaches even more: If there is reason to believe that neither of the two is always superior, in which situations should one prefer one over the other?

6.2.3. Understanding important features

Figures 6.10 and 6.11 show an image representation of the weights, similar to images Figures 6.5 and 6.6. Based on these figures we can try and understand how the classifiers come to their conclusions.

Notice that there is no clear pattern visible which is desirable, because it might learn high-frequency relations which are not as domain dependent as we had seen in the FFHQ vs. StyleGAN case.

6.2.4. Conclusion

Once again, we have seen that linear classification of synthetic and real images based on both wavelet-based representations can have high accuracy. Interestingly enough, the regular wavelet approach

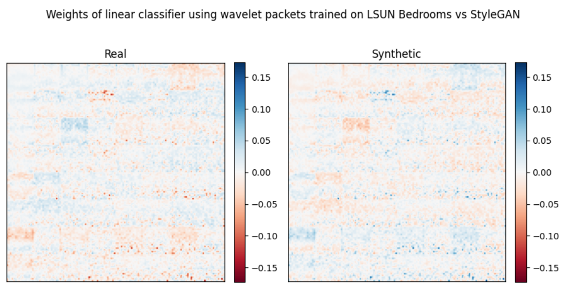


Figure 6.10: Weights learned by linear classifier when applying synthetic image detection on a wavelet packet representation, using three-levels and the Haar wavelet, represented as an image.

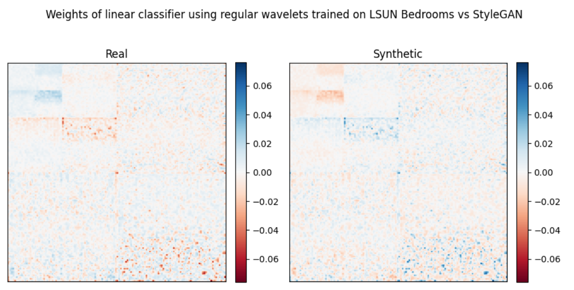


Figure 6.11: Weights learned by linear classifier when applying synthetic image detection on a regular wavelet representation, using three-levels and the Haar wavelet, represented as an image.

outperformed the wavelet packet approach in this experiment, whereas in the StyleGAN vs. FFHQ setting, the packet approach seemed better.

Notice that both of our previous experiments relied on StyleGAN, a GAN-based generator. With the rise of Diffusion Model generated images, the need for tools to detect DM generated imagery, increases. How will our approach fare under Diffusion Models? This question introduces our upcoming section, where we will investigate DDPM vs. LSUN Bedroom images.

6.3. DDPM vs. LSUN Bedroom

6.3.1. Visual mean and variance deviations

Once again, we will start off by analysis of visual mean and variance deviations of both the regular and wavelet-packet representations. We have added both outputs of the procedure described in Figure 6.1 in Figures 6.12 and 6.13. Notice that the deviations are a lot smaller than for the StyleGAN vs. LSUN Bedroom setting.

Moreover, as we can see in Figure 6.13, similar grid-esque patterns as in the StyleGAN setting are visible. Once again, these are most notable in the variance setting. Even though DDPM does not introduce blocky artefacts such as StyleGAN does, it still underestimates the grid-pattern. This could be understood by the analysis of Corvi et al. [6], who showed that for their noise-residual auto-correlation values, different models achieve different levels of reconstruction performance.

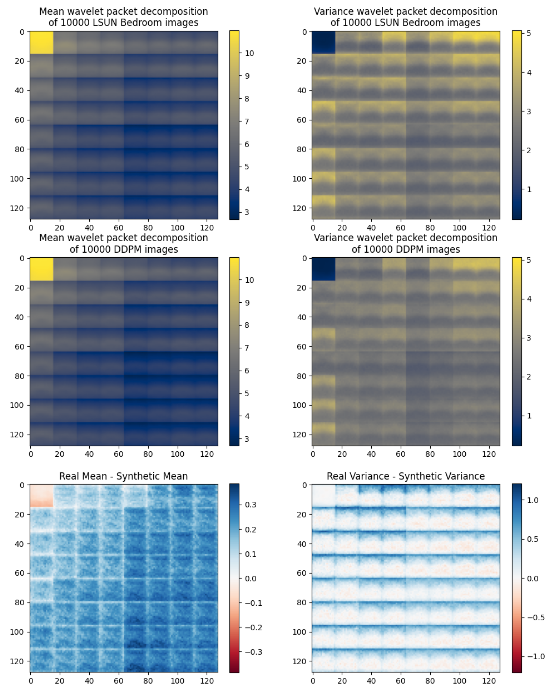


Figure 6.12: Three-level Haar wavelet packet decomposition of real and synthetic images. Real images used are 128-by-128 images of the LSUN Bedroom dataset, synthetic images used are 128-by-128 images generated by DDPM, trained on the LSUN Bedroom dataset.

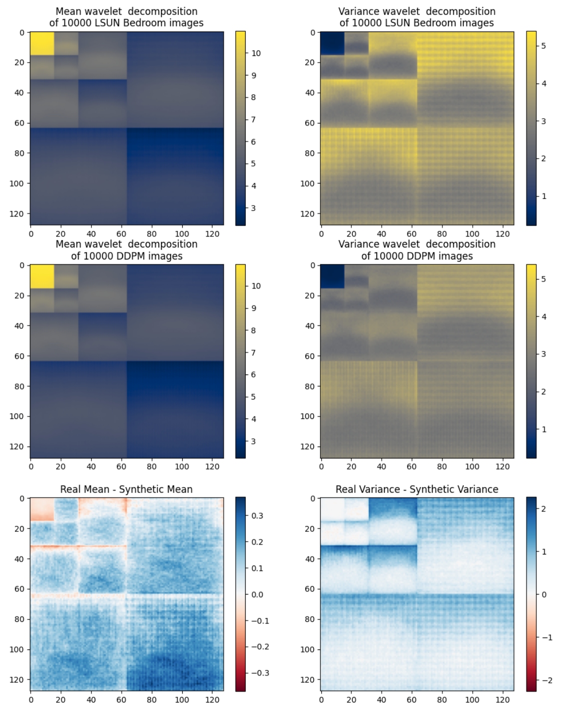


Figure 6.13: Three-level Haar wavelet decomposition of real and synthetic images. Real images used are 128-by-128 images of the LSUN Bedroom dataset, synthetic images used are 128-by-128 images generated by DDPM, trained on the LSUN Bedroom dataset.

6.3.2. Experimental Results

Once again, our experimental setting is described in Chapter 5. The results are plotted in Figure 6.14.

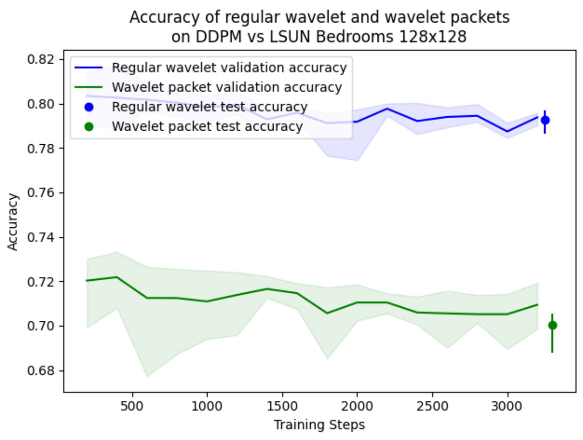


Figure 6.14: Validation and test accuracy for SID on DDPM vs LSUN Bedroom using our classifier. For both wavelet packets and regular wavelets, the experiment has been run five times, where the means are shown as a line and dot for the validation and test accuracy respectively. The shaded areas and the vertical line through the dot correspond to the range of accuracies found in the experiments.

In contrast to previous experiments, average performance has deteriorated quite a bit for both regular wavelets and the wavelet packet approach: Whereas test accuracy was above 95% for any of the previous experiments, even the best performing model in this setting does not achieve an accuracy performance higher than 80%.

The astute reader recalls that we had similar accuracy in the previous two experiments when using raw-pixel values to train our linear classifier. However, when running that same experiment in this setting, we found that such a classifier only achieves a performance of roughly 55%, just a bit better than random guessing.

Since we are exactly in the same setting as our experiments of section Section 6.2.2, but are using DDPM rather than StyleGAN, this gives us additional evidence that Diffusion Models are more difficult to detect than GANs, inline with conclusions of Ricker et al. [35].

Notably, the wavelet packet approach has an average accuracy of approximately 70.0%, which is a lot lower than the regular wavelet approach, which has an average accuracy of 79.2%. This reinforces our belief that special attention should be placed on the choice of using regular wavelets or wavelet packets, since it can lead us to quite a difference in accuracy.

An additional interesting observation based on Figure 6.14 is that the validation accuracy seems to have decreased over the training steps, which might indicate that there is some overfitting going on. We will later return to this point.

To try have a better understanding of why this discrepancy exists, we will once again take a look at the features used by the classifier in the upcoming section.

6.3.3. Understanding Important Features

The weights for both classifiers have been resized and plotted in Figures 6.15 and 6.16. Once again, we have taken the mean over the colour axes and are just choosing one of the five models for both wavelet-based approaches, as the analysis does not differ based on those factors.

It's interesting to see that in Figure 6.16 the second level diagonal coefficients seem to be impotant for indicating that an image is synthetic. Additionally, the third-level vertical coefficients seem to be important for classification if an image is real.

The weights of the first-level coefficients in Figure 6.16 do not seem to follow a certain pattern, which we mentioned earlier might be a sign of less domain dependence.

In Figure 6.15, we can see that surprisingly, the coefficients corresponding to the packet *cHAA* is the most important one to decide whether an image is synthetic or not. If we relate this to Figure 6.16, this is unexpected, as the wavelet based approach does not place special attention to the first level horizontal coefficients, of which the *cHAA* packet is an approximation.

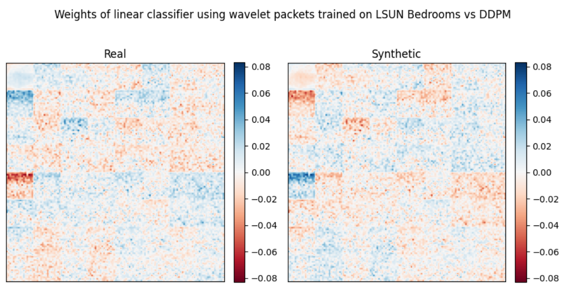


Figure 6.15: Weights learned by linear classifier when applying synthetic image detection on a wavelet packet representation, using three-levels and the Haar wavelet, represented as an image.

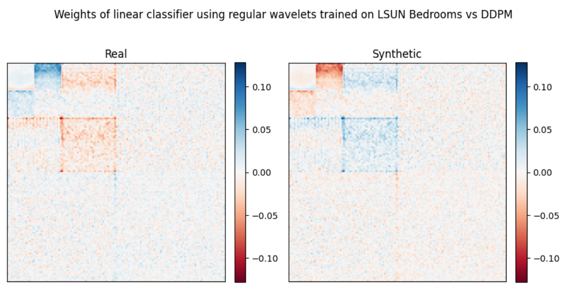


Figure 6.16: Weights learned by linear classifier when applying synthetic image detection on a regular wavelet representation, using three-levels and the Haar wavelet, represented as an image.

This shows that in this setting both approaches are not necessarily looking at the same features to decide whether an image is synthetic.

6.3.4. Conclusion

In conclusion, the linear classifier degrades quite a bit in performance when using StyleGAN over DDPM in the LSUN Bedroom setting. This is in line with the observation by Ricker et al. [35] that DM-generated imagery is more difficult to detect than GAN-generated imagery.

Additionally, we can conclude that the degradation of the wavelet packet approach is larger than that for the regular wavelet approach. This might be due to the fact that the wavelet-based approach allows for more direct incorporation of multi-scale information.

6.4. Training using 10% of the training data

In all of our previous experiments, we have seen that the classifiers do not seem to benefit a lot from additional training after a couple of hundred training steps. This seems to indicate that perhaps the classifiers do not need to have access to this many samples to perform at their capacity. This would be a nice property to have because it allows for more rapid iterative model development. Moreover, it would show that the classifier can perform even in situations where there is little data available.

In Table 6.1 we present the mean test accuracies over five seeds for the different classifiers over the different datasets. Under the 100% column we show the mean test accuracies for models using 100% of the training data and accordingly we show the mean test accuracies for models using 10% of the training data under the 10% column. Under the Δ column we show the difference between the 100% accuracy column and the 10% accuracy column.

Table 6.1: Mean test accuracies over five different seeds for the three experimental settings described in the columns. The rows correspond to the mean test accuracies of the dataset trained and tested on 100% or 10% of the data. The Δ column shows the difference between these two. Moreover, the columns are grouped into ‘Packets’, under which the results are noted when using packet-based transform, and ‘Wavelets’, under which the results are noted when using the regular wavelet-based transform.

	Packets			Wavelets		
	100%	10%	Δ	100%	10%	Δ
FFHQ-StyleGAN	0.9971	0.9907	0.0065	0.9932	0.9864	0.0068
LSUN-StyleGAN	0.9867	0.9597	0.0270	0.9785	0.9727	0.0057
LSUN-DDPM	0.7002	0.6835	0.0167	0.7748	0.7926	-0.0178

Notice that overall the mean test accuracy is relatively similar, the deterioration is relatively small. Even though we are removing 90% of the training data, the worst deterioration in terms of mean test accuracies, is not even 3% points. For the wavelet based approaches, the worst deterioration is not even 0.7%.

Interestingly, the packet approach seems to deteriorate more than the wavelet approach on both

LSUN models. A quite remarkable result is the fact that using wavelets on just 10% of the data in the LSUN-DDPM setting achieves a higher mean test accuracy than using 100% of the data in that setting. This is, nonetheless, not that surprising given the fact that we saw a decline in validation accuracy in Figure 6.14 over the training steps. However, noticing that we also saw this decline for the packet approach in the figure, it is even more interesting to see that the packet approach does worse when using just 10% of the data.

Only on the FFHQ-StyleGAN dataset does the packet approach deteriorate less than the wavelet approach. This might be due to the importance of the *cDAA* wavelet packet and still being noticeable when using a small number of weights.

Robustness under training set reduction have been found by both Frank et al. as well as Moritz et al. However, their results were only found when using a more complex CNN classifiers. Our results show that even in much simpler classifiers these wavelet-based approaches are surprisingly robust.

6.5. Steps towards generalizability

The results we have seen are interesting, but are all done in pristine conditions. In this section, we will first ponder the question to what extent the performance of these classifiers can be generalized across datasets. We will see that performance deteriorates gravely across datasets. Afterwards, we will use a concept from the Synthetic Image Detection field not considered by both Frank et al. and Wolter et al. that could possibly allow for better generalization across datasets. Rather than using full images as inputs, we will only consider the residuals in the image as an input.

6.5.1. Performance across datasets

Ideally, we would like a classifier to perform well across datasets and models. This would mean that rather than having to train a classifier for any specific type of image, we can use the same classifier for multiple use cases. Similar to how the recent advances in synthetic image generators allow us to do zero-shot synthetic image generation, we would like to have a model to have zero-shot synthetic image detection capabilities.

One step towards this zero-shot synthetic image detection would be to consider how well our trained classifiers perform on different datasets. To this end, we can simply consider the trained model that was trained on e.g. the FFHQ-StyleGAN dataset and consider the accuracy of this model on the test set of the LSUN-StyleGAN dataset. If we then see that the accuracy stays high across the different datasets, we have evidence that the classifier learns features of synthetic generated images that are apparent across different datasets. If this is not the case, then we have evidence that the learned features are specific to the dataset.

We will use the classifiers trained on 10% of the data for these experiments. Since we have five of those classifiers, all trained on different seeds, we will report the mean test accuracy of the classifier. We have noted the results for the packet based classifiers in Table 6.2 and for the regular wavelet based classifiers in Table 6.3. For both tables, the columns correspond with which dataset was used to train the classifier on, and the rows corresponds to which dataset was used to test the performance. One important note we have to place, is that there might be data leakage in the combination between LSUN-DDPM and LSUN-StyleGAN images. Since both datasets rely on the same dataset of real images and for each of the models, we have randomly splitted the LSUN dataset into training and test images, it can be the case that a real image is in the test set of LSUN-DDPM, but in the training set of LSUN-StyleGAN.

Table 6.2: Cross-dataset performance of trained models using the packet-based representation. In the rows we see the model on which the classifier was trained and in the columns over which the trained model was tested. Notice that the diagonal corresponds to pristine experiments, where the classifier is trained on the same dataset as it is tested upon. Every cell is the mean test accuracy of classifier trained on the row-dataset and tested on the test-set of the column-dataset.

		Tested on		
		LSUN-DDPM	LSUN-StyleGAN	FFHQ-StyleGAN
Trained on	LSUN-DDPM	**0.6835**	0.5476	0.0839
	LSUN-StyleGAN	0.4935	**0.9597**	0.5008
	FFHQ-StyleGAN	0.4359	0.4771	**0.9907**

Table 6.3: Cross-dataset performance of trained models using the regular wavelet-based representation. In the rows we see the model on which the classifier was trained and in the columns over which the trained model was tested. Notice that the diagonal corresponds to pristine experiments, where the classifier is trained on the same dataset as it is tested upon. Every cell is the mean test accuracy of classifier trained on the row-dataset and tested on the test-set of the column-dataset.

		Tested on		
		LSUN-DDPM	LSUN-StyleGAN	FFHQ-StyleGAN
Trained on	LSUN-DDPM	**0.7926**	0.7244	0.2270
	LSUN-StyleGAN	0.5199	**0.9864**	0.4993
	FFHQ-StyleGAN	0.4851	0.4440	**0.9727**

Some interesting conclusions can be made based on the result of Tables 6.2 and 6.3. Most notably, we see that, in general, performance deteriorates quite a bit if we are considering performance on another dataset. Most cross-dataset accuracies are around 50%, which is the same a random guessing classifier would yield us. For both regular wavelet and packet based classifiers, the model trained on LSUN-DDPM and tested on FFHQ-StyleGAN performs the worst, with the packet-based approach achieving an accuracy of just 8%. This indicates that the features the classifier learns are good for differentiating LSUN bedroom images and DDPM-based syntetics, are also working quite well for differentiating FFHQ and StyleGAN images, but the other way around. This can also be seen if we would flip the outputting labels, then it would achieve an accuracy of 92%.

Another interesting phenomenon, is the fact that the model trained on LSUN-DDPM performs relatively well on the LSUN-StyleGAN dataset: it just loses 7% accuracy. This could be due to the leakage issue we discussed earlier, but if we are to consider the fact that the LSUN-StyleGAN trained model has an accuracy of just 52%, it makes it more likely that the LSUN-DDPM model has learned features that are generalizable across different synthetic image generators. This supports the hypothesis of Ricker et al. that images generated by DMs contain patterns similar to GAN-generated images, but less pronounced. However, in light of this hypothesis, we would expect that the model trained on DDPM would also perform well on the FFHQ-StyleGAN dataset, but as we have seen, this is not the case. This is probably due to the fact that the learned features are all heavily domain reliant, especially for the FFHQ-StyleGAN dataset, for which we noticed that the classifiers heavily rely on wavelet coefficients that are related to typical positions in space where faces are.

In the upcoming section we will try to counter this domain reliance by introducing a technique not considered by both Frank et al. and Wolter et al.

6.5.2. Residual-based classification

We will apply a PRNU-based approach. That is, rather than training on the full image, we will train our classifiers on the residuals of an image. Noticing that most literature agrees on the fact that both GAN-based imagery and DM-based imagery differ with real images in terms of high-frequency content, we should be able to classify images as real or synthetic based on their high-frequency content. Such an approach has the additional benefit of being less domain-reliant.

To this end, we will need a way to denoise an image. To this end we will use Haar-based hard thresholding: That is, given an image, we transform it into its Haar-based wavelet coefficients and then set coefficients equal to zero if these coefficients are smaller than a certain threshold in absolute values. After setting coefficients beneath the threshold to zero, we can use the inverse discrete wavelet transform to get the denoised version of the image.

There are many ways this thresholding can be done, but we will use the Donoho's universal wavelet denoising threshold [9], $\lambda_{\text{universal}} = \sigma\sqrt{2}\log N$, where N is the data dimensionality and σ being the noise variance, which is to be estimated. We have estimated this value by visual inspection to be 0.23, which sets $\lambda = 4.12$.

After obtaining the denoised version of the image, we can find the residuals by using the absolute deviation between the original image and the denoised image. An example of finding residuals based Haar-wavelet thresholding, can be found in Figure 6.17. Note that in Figure 6.17 we are visualizing wavelet denoising for a single colour channel, but in our experiments we will apply the the denoising for each of the three colour channels and thus for any image, we will get three sets of residuals. Notice how close the denoised image is to the original image and that thus the added values of these residuals in terms of perceptive quality is only small.

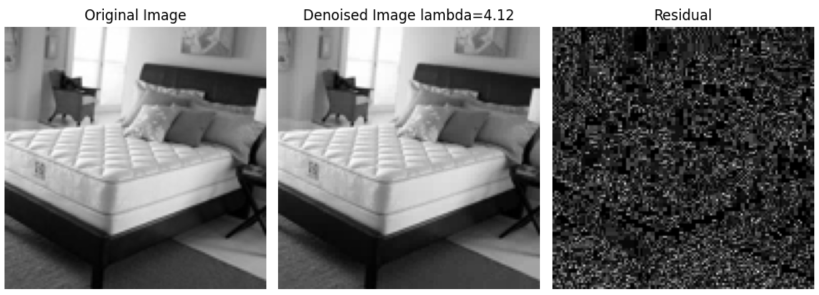


Figure 6.17: On the left: An LSUN bedroom image. In the middle: Denoised version of said LSUN bedroom image, where Haar-based hard-wavelet thresholding is used as a denoiser. On the right: Residual betweeen the original image and the denoised image.

Once again, we will train the classifiers on 10% of the data and we will use five different seeds to train the models where we only report the mean results. These results can be found in Tables 6.4 and 6.5 for the packet-based classifiers and the wavelet-based classifiers respectively.

A couple of interesting observations can be done. First of all, it should be noted that the packet-based approaches keep quite high accuracy for both the LSUN-StyleGAN and FFHQ-StyleGAN dataset if we are to consider the test accuracy on the same dataset. This supports the hypothesis that Style-GAN really does leave behind high-frequency artefacts that can be used to detect synthetic images. However, as can be seen in the cross-dataset tests of these classifiers: The artefacts used by our classifier are not consistent over different datasets.

Another interesting observation, is that the packet-based classifier seems to perform better or the same in general when we are using residuals relative to the whole image. By using residuals, we are using only a relative small amount of data with respect to the total image. However, with regards to cross-dataset performance of the trained models, the packet-based classifiers still perform with near-chance like accuracy. Except for the models trained on LSUN-DDPM and tested on FFHQ-StyleGAN, which achieves once again around 8.3% accuracy.

Table 6.4: Cross-dataset performance of trained models using the packet-based representation of residuals. In the rows we see the model on which the classifier was trained and in the columns over which the trained model was tested. Notice that the diagonal corresponds to pristine experiments, where the classifier is trained on the same dataset as it is tested upon. Every cell is the mean test accuracy of classifier trained on the row-dataset and tested on the test-set of the column-dataset.

		Tested on		
		LSUN-DDPM	LSUN-StyleGAN	FFHQ-StyleGAN
Trained on	LSUN-DDPM	**0.6856**	0.5458	0.0834
	LSUN-StyleGAN	0.5213	**0.9564**	0.5240
	FFHQ-StyleGAN	0.4401	0.4866	**0.9817**

Table 6.5: Cross-dataset performance of trained models using the regular wavelet-based representation of residuals. In the rows we see the model on which the classifier was trained and in the columns over which the trained model was tested. Notice that the diagonal corresponds to pristine experiments, where the classifier is trained on the same dataset as it is tested upon. Every cell is the mean test accuracy of classifier trained on the row-dataset and tested on the test-set of the column-dataset.

		Tested on		
		LSUN-DDPM	LSUN-StyleGAN	FFHQ-StyleGAN
Trained on	LSUN-DDPM	**0.6765**	0.6764	0.4770
	LSUN-StyleGAN	0.5203	**0.9520**	0.5279
	FFHQ-StyleGAN	0.4264	0.4766	**0.9676**

7 Conclusion

We will split the conclusion into two sections: One dealing with conclusions made on the comparison of the regular wavelets approach to the wavelet packet approach and another more general section that deals with conclusions made for both wavelet-based approaches and our used classifier.

7.1. Regular wavelets vs. wavelet packets

We have seen that both the wavelet packet and regular wavelet approaches show high accuracy in detecting synthetic images generated by StyleGAN. We have seen that the wavelet packet approach slightly outperformed the regular wavelet approach in the FFHQ-StyleGAN dataset, whereas the regular wavelet approach outperformed the packet approach on both the LSUN-StyleGAN dataset as well as the LSUN-DDPM dataset. This suggests that the choice of which approach to use, depends on the combination of dataset and model.

In the StyleGAN vs. FFHQ challenge, we saw that the *cDAA* packet was quite different for synthetic images and real images. Due to this discrepancy, the packet-based approach performed very well. In the LSUN bedroom case, where no single packet was that different between the synthetic and real images, the performance was not as good.

Another possible reason for why the regular wavelets outperformed wavelet packets on the LSUN bedroom datasets might be that the multi-scale representation that regular wavelet approach offers can be immediately used: For wavelet packets of course the same information is available, but in a more contrived format, which our classifier might not be able to learn due to its simplicity.

7.2. Wavelet-based approaches in conjunction with our classifier

The performance of the classifiers significantly deteriorated when detecting synthetic images generated by the Diffusion Model, DDPM, relative to StyleGAN. Quite surprisingly, we found that the model trained on LSUN-DDPM, had quite good cross-dataset performance on the LSUN-StyleGAN dataset. This finding supports the claim by Ricker et al. that DM-generated imagery is, in general, more difficult to detect.

Other than that, we found that a reduction of the training data to 10% of the original training set, yielded nearly the same accuracy for all of the classifiers. This suggests that the wavelet-based approaches are robust to data reductions. This is an extension of the findings by both Wolter et al. and Frank et al., who both found the same robustness to hold for wavelet-packets and DCT based-CNNs respectively, to our much simpler classifier. This property can be used to run experiments more quickly and additionally allows for quicker iterations in classifier development.

Lastly, we found that cross-dataset performance of the classifiers is generally poor, indicating that the learned features are not easily generalizable across different datasets and synthetic image generators. We tried a novel way of tackling this issue by training our classifier on residuals rather than full inputs, but this did not significantly improve cross-dataset performance.

Overall, we found that our simple model worked quite well in the pristine setting of GANs, where the task is simply to distinguish between GAN-generated images and the exact set which was used to train the GAN. However, in the setting of Diffusion Models, or cross-dataset performance, we found the classifier to fail.

8 Discussion and Future Recommendations

In this section, we will dive into the weaknesses of this research and put them in a broader context by relating these weaknesses to the more general critiques on the synthetic image detection field. We will consider challenges associated with the experimental protocol and challenges associated with our choice of classifier. During our discussion, we will inform the reader on possible interesting avenues for further research.

8.1. Experimental protocol

Our experiments are fairly limited in their scope: we are only testing a single type of GAN generator and a single type of DM generator, so any results that we ascribe to the whole class of generators, might also be due to specifically StyleGAN or DDPM, rather than the whole class of GANs or DMs. One of the reasons to why our experiments have been limited, is due to the inaccessibility of synthetic data: most of the literature relies on readers of papers to have access to excessive computing power and that they are able to generate data themselves. This is a challenge, especially for Diffusion Models, which are notoriously slow to sample from. This practice of relying on readers to generate datasets themselves also strongly threatens capabilities to reproduce results. As we have seen in the case of trying to reproduce Figure 4.2 of Frank et al., where we failed to do so because, according to the author, the StyleGAN implementation he used was a Tensorflow-based implementation as opposed to the Pytorch-based implementation that is used nowadays.

Not only is this practice inaccessible, it is also very unsustainable, as it requires a great amount of computational power for something that has already been done by someone else. The field would benefit if these created, synthetic datasets were shared. I have had contact with Jonas Ricker, one of the authors of *Towards the Detection of Diffusion Model Deepfakes* [35] with the request to release their synthetically generated images and they were happy to do so. Hopefully this will kick off a trend in the field of synthetic image detection that will make it standard procedure to share generated datasets. To follow in this tradition, I have uploaded my datasets at the following link: `https://www.kaggle.com/datasets/arianjaaa/synthetic-image-detection-dataset`.

More broadly speaking, this issue of everyone possibly using datasets that are different, is a feature of the lack of experimental protocol in the field. Different papers all use different experimental settings, which makes it rather difficult to compare between results between papers. For example, we have focused on the papers by Frank et al. and Wolter et al. where Wolter et al. explicitly follow a very similar structure to the paper by Frank et al. and even then there are some discrepancies: e.g. Wolter et al. use all colour channels, whereas Frank et al. use grayscaled versions of images. To try and truly create work that is comparable, we have tried to stay very close to the work by Wolter et al. but of course, this means that if any of their choices were invalid, ours will be too. The field would benefit from more standardized experimental protocol, which could be an interesting topic for further research.

Another critique on our experimental protocol is that we are considering, a very pristine training and test set. We are assuming that the synthetic images we wish to classify are trained on exactly the real images we have in our training set for our three main experiments. To try and remedy this, we have tried to introduce an approach that classifies only using the high-frequency content of our images, but this did not better performance. An interesting direction for further research could be to try more sophisticated denoising algorithms, such as the one used by Corvi et al. [6]. Moreover, in realistic scenarios such as social media, where the need for synthetic image detection is highest, images are repeatedly resized and JPEG compressed, which can heavily impact quality of detectors as is shown by Gragnaniello et al. [17].

Besides that, our results are also acquired on the specific combination of Haar wavelet and three-levels. Even though we have tested different types of wavelets in other, smaller scale experiments and noticed similar behaviour to the Haar wavelet, we cannot say for sure whether our results generalize to other type of wavelets or levels.

We have only considered accuracy as a metric of interest, but it is likely that in settings where one would like to use such a synthetic image detector, we would like to treat misclassifications differently. Since by using accuracy as our metric of interest, we are implicitly saying that both types of mistakes are equally as bad. Suppose for example that a social media platform wants to use the detector to automatically detect and flag synthetic images uploaded by users to prevent the spread of misinformation. Minimizing false positives is very important in this setting, because false positives can lead to frustration among users and damage to the platform.

8.2. Choice of classifier

Our classifier is a type of linear classifier, which on the one hand means that we can understand more directly how specific inputs affect the output. However, the downside of such an approach is that it is very easy for malicious actors to trick the classifier. E.g. knowing that the *cDAA* coefficients are important for saying that a face is synthetically generated in the packet-based FFHQ-StyleGAN classifier, a malicious actor could make the classifier classify real faces as fake by changing the input image so that this packet has a high value and thus lets the model output synthetic for a real face. Of course, similar techniques exist for CNN-based classifiers, where malicious users can fool a detector by simply perturbing the input [16].

It would be of interest to conclude more complex classifiers such as CNNs. Not only because we see that performance is already stagnating for the DDPM-case around validation accuracy 0.8 whilst the training accuracy is already 1.0[1], but also because we had seen that, in terms of variance, synthetic images and real images are quite different. Since our model could only learn a single template, it was not able to capture this discrepancy in variances.

Moreover, our model depends heavily on spatial position of objects. This does not seem as a desirable property, a synthetic image should not be labelled as real if it is simply flipped vertically or if the image is translated a bit. This problem is heavily related to the challenge in the field of image classification of finding translation invariant representations. An approach that allows for translation-invariant representations that preservers high-frequency information, is the usage of Invariant Scattering Convolution Networks [3], which is an approach that has not yet been used for synthetic image detection and seems very promising given the challenges translation dependence has given us in this thesis.

[1]We have observed this in our results, but not added the training accuracy in the thesis.

Bibliography

[1] Yaniv Benny and Lior Wolf. “Dynamic Dual-Output Diffusion Models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 11482–11491.

[2] Christian Blatter. *Wavelets: a primer*. AK Peters/CRC Press, 2018.

[3] Joan Bruna and Stéphane Mallat. “Invariant scattering convolution networks”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1872–1886.

[4] Ziyi Chang, George A Koulieris, and Hubert PH Shum. “On the Design Fundamentals of Diffusion Models: A Survey”. In: *arXiv preprint arXiv:2306.04542* (2023).

[5] Ole Christensen et al. *Functions, spaces, and expansions: mathematical tools in physics and engineering*. Springer, 2010.

[6] Riccardo Corvi et al. “On the detection of synthetic images generated by diffusion models”. In: *arXiv preprint arXiv:2211.00680* (2022).

[7] Riccardo Corvi et al. “On the detection of synthetic images generated by diffusion models”. In: *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2023, pp. 1–5.

[8] Prafulla Dhariwal and Alexander Nichol. “Diffusion models beat gans on image synthesis”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 8780–8794.

[9] David L Donoho and Iain M Johnstone. “Threshold selection for wavelet shrinkage of noisy data”. In: *Proceedings of 16th annual international conference of the IEEE engineering in medicine and biology society*. Vol. 1. IEEE. 1994, A24–A25.

[10] John Duchi. “Derivations for linear algebra and optimization”. In: *Berkeley, California* 3.1 (2007), pp. 2325–5870.

[11] Ricard Durall, Margret Keuper, and Janis Keuper. “Watch your up-convolution: Cnn based generative deep neural networks are failing to reproduce spectral distributions”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 7890–7899.

[12] Tarik Dzanic, Karan Shah, and Freddie Witherden. “Fourier spectrum discrepancies in deep network generated images”. In: *Advances in neural information processing systems* 33 (2020), pp. 3022–3032.

[13] Joel Frank et al. “Leveraging frequency analysis for deep fake image recognition”. In: *International conference on machine learning*. PMLR. 2020, pp. 3247–3258.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[15] Ian Goodfellow et al. “Generative adversarial networks”. In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

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

[17] Diego Gragnaniello et al. “Are GAN generated images easy to detect? A critical analysis of the state-of-the-art”. In: *2021 IEEE international conference on multimedia and expo (ICME)*. IEEE. 2021, pp. 1–6.

[18] V Hermann. “Wavelets II: Vanishing moments and spectral factorization”. In: *DSP Related, 2016a* 27 (2018).

[19] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6840–6851.

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

[21] Tero Karras, Samuli Laine, and Timo Aila. “A style-based generator architecture for generative adversarial networks”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 4401–4410.

[22] Diederik Kingma et al. “Variational diffusion models”. In: *Advances in neural information processing systems* 34 (2021), pp. 21696–21707.

[23] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[24] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[25] Haodong Li et al. “Identification of deep network generated images using disparities in color components”. In: *Signal Processing* 174 (2020), p. 107616.

[26] Yunhe Liu et al. “Wavelet-based 3-D inversion for frequency-domain airborne EM data”. In: *Geophysical Journal International* 213.1 (2018), pp. 1–15.

[27] Ziwei Liu et al. “Large-scale celebfaces attributes (celeba) dataset”. In: *Retrieved August* 15.2018 (2018), p. 11.

[28] Jan Lukáš, Jessica Fridrich, and Miroslav Goljan. “Detecting digital image forgeries using sensor pattern noise”. In: *Security, steganography, and watermarking of multimedia contents VIII*. Vol. 6072. SPIE. 2006, pp. 362–372.

[29] Calvin Luo. “Understanding diffusion models: A unified perspective”. In: *arXiv preprint arXiv:2208.11970* (2022).

[30] Falko Matern, Christian Riess, and Marc Stamminger. “Exploiting visual artifacts to expose deepfakes and face manipulations”. In: *2019 IEEE Winter Applications of Computer Vision Workshops (WACVW)*. IEEE. 2019, pp. 83–92.

[31] Vasco Nunes, João Dias, and Pedro A Santos. “GAN-Based Content Generation of Maps for Strategy Games”. In: *arXiv preprint arXiv:2301.02874* (2023).

[32] Augustus Odena, Vincent Dumoulin, and Chris Olah. “Deconvolution and checkerboard artifacts”. In: *Distill* 1.10 (2016), e3.

[33] Gaurav Parmar, Richard Zhang, and Jun-Yan Zhu. “On Aliased Resizing and Surprising Subtleties in GAN Evaluation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 11410–11420.

[34] Aditya Ramesh et al. “Zero-shot text-to-image generation”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8821–8831.

[35] Jonas Ricker et al. “Towards the Detection of Diffusion Model Deepfakes”. In: *arXiv preprint arXiv:2210.14571* (2022).

[36] Robin Rombach et al. “High-resolution image synthesis with latent diffusion models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 10684–10695.

[37] Chitwan Saharia et al. “Photorealistic text-to-image diffusion models with deep language understanding”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 36479–36494.

[38] Katja Schwarz, Yiyi Liao, and Andreas Geiger. “On the frequency bias of generative models”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 18126–18136.

[39] Stefano Scotta and Alberto Messina. “Understanding and contextualising diffusion models”. In: *arXiv preprint arXiv:2302.01394* (2023).

[40] Sakib Shahriar. “GAN computers generate arts? a survey on visual arts, music, and literary text generation using generative adversarial network”. In: *Displays* 73 (2022), p. 102237.

[41] Yang Song et al. “Score-based generative modeling through stochastic differential equations”. In: *arXiv preprint arXiv:2011.13456* (2020).

[42] Gilbert Strang and Truong Nguyen. *Wavelets and filter banks*. SIAM, 1996.

[43] Guihua Tang et al. “Detection of GAN-synthesized image based on discrete wavelet transform”. In: *Security and Communication Networks* 2021 (2021), pp. 1–10.

[44] Clemens Valens. “A really friendly guide to wavelets”. In: (1999).

[45] Luisa Verdoliva. “Media forensics and deepfakes: an overview”. In: *IEEE Journal of Selected Topics in Signal Processing* 14.5 (2020), pp. 910–932.

[46] Sheng-Yu Wang et al. “CNN-generated images are surprisingly easy to spot... for now”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 8695–8704.

[47] Lilian Weng. “What are diffusion models?” In: *lilianweng.github.io* (July 2021). URL: `https://lilianweng.github.io/posts/2021-07-11-diffusion-models/`.

[48] Moritz Wolter et al. “Wavelet-packets for deepfake image analysis and detection”. In: *Machine Learning* (2022), pp. 1–33.

[49] Fisher Yu et al. “Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop”. In: *arXiv preprint arXiv:1506.03365* (2015).

[50] Ning Yu, Larry S Davis, and Mario Fritz. “Attributing fake images to gans: Learning and analyzing gan fingerprints”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 7556–7566.

[51] Xu Zhang, Svebor Karaman, and Shih-Fu Chang. “Detecting and simulating artifacts in gan fake images”. In: *2019 IEEE international workshop on information forensics and security (WIFS)*. IEEE. 2019, pp. 1–6.

A
Figures

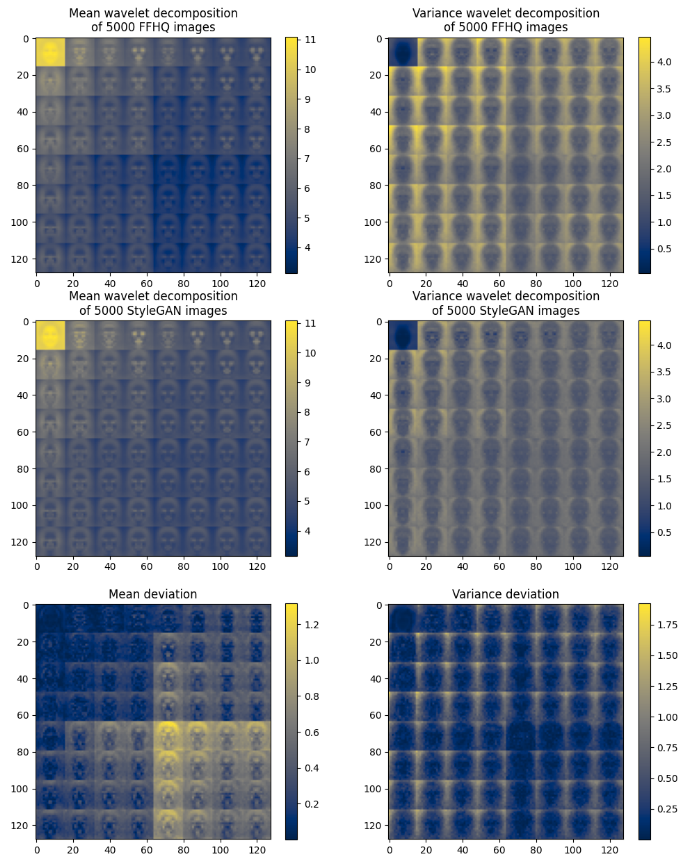


Figure A.1: Three-level Haar wavelet packet decomposition of real and synthetic images. Real images used are 128-by-128 images of the FFHQ dataset, synthetic images used are 128-by-128 images generated by StyleGAN, trained on the FFHQ dataset.

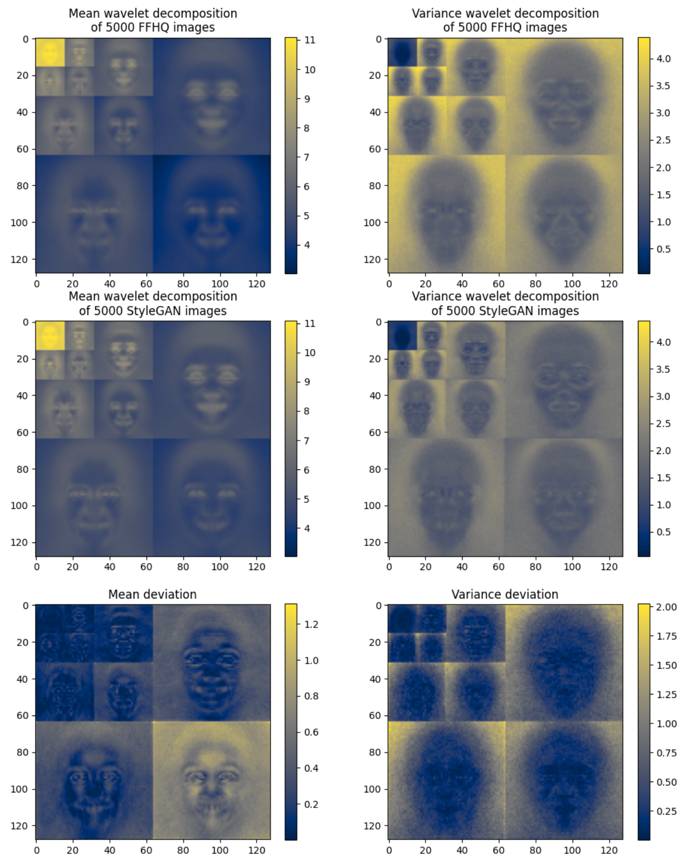


Figure A.2: Three-level Haar wavelet decomposition of real and synthetic images. Real images used are 128-by-128 images of the FFHQ dataset, synthetic images used are 128-by-128 images generated by StyleGAN, trained on the FFHQ dataset.