DUBLIN CITY UNIVERSITY
SCHOOL OF ELECTRONIC ENGINEERING

Enhancing steganography for hiding pixels inside audio signals

Jaume Ros Alonso

May 2022

BACHELOR OF ENGINEERING

IN

DATA SCIENCE AND ENGINEERING

Supervised by Kevin McGuinness

Acknowledgements

I would like to thank my tutors Xavier Giro-i-Nieto, Jordi Pons and Kevin McGuinness for giving me the opportunity to work on this project and for all the feedback and support they offered during the development of this work.
I also want to express my gratitude towards the Polytechnic University of Catalonia (UPC) for providing the computational resources that enabled me to run the dozens of experiments required to make this thesis possible.
I wish to thank Cristina Puntí, Margarita Geleta and Teresa Domènech for the previous work done on this project which I had the pleasure to continue; especially Cristina, that had the patience to help me in setting up everything for the first time.
Finally, I would like to thank Pau Bernat, who worked alongside me for these past few months and gave me crazy ideas when crazy ideas were needed.

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.
I have read and understood the DCU Academic Integrity and Plagiarism at `https://www.dcu.ie/system/files/2020-09/1_-_integrity_and_plagiarism_policy_ovpaa-v4.pdf` and IEEE referencing guidelines found at `https://loop.dcu.ie/mod/url/view.php?id=1401800`.

Jaume Ros Alonso
May 2022

Abstract

Multimodal steganography consists of concealing a signal into another one of a different medium, such that the latter is only very slightly distorted and the hidden information can be later recovered. A previous work employed deep learning techniques to this end by hiding an image inside an audio signal's spectrogram in a way that the encoding of one is independent of the other.
In this work we explore the way in which images were being encoded previously and present a collection of improvements that produce a significant increase in the quality of the system. These mainly consist in encoding the image in a smarter way such that more information is able to be transmitted in a container of the same size. We also explore the possibility of using the short-time Fourier transform phase as an alternative to the magnitude and to randomly permute the signal to break the structure of the noise. Finally, we report results when using a larger container signal and outline possible directions for future work.

Contents

List of Figures

List of Tables

Chapter 1 - Introduction

Steganography is the practice of hiding a secret message inside another one, which can be sent in plain sight. As opposed to cryptography, that aims to make a message unintelligible to third parties, steganography tries to hide the presence of such a communication in the first place. Since the container signal appears to be well-formatted, it benefits from the fact that potential attackers do not suspect about the presence of a hidden message inside.

Traditionally, steganography in digital signals has used least significant bit (LSB) approaches [1], where the last bits of the container signal are used to encode the hidden message, thus producing minimal distortion in the container. The number of bits used to encode the hidden signal is a hyperparameter that determines the trade-off between the container signal's distortion and its capacity to carry information.

Recently, deep learning techniques have been used as an alternative approach to LSB ones, offering more flexibility in the encoding method, since it is the network that is able to learn which redundancy to exploit in the container. While steganography is typically used to encode in a uni-modal way (*e.g.* hiding images inside images [2] or audio inside audio [3]), multi-modal architectures are also possible.

1.1 State of the art

Deep convolutional neural networks are widely used to process image data. These are usually trained in an end-to-end fashion and the architecture is often composed of an encoder, that learns to hide the image in the host signal, and a decoder, that aims to reveal its contents and get the original data back.

[2] and [4] (the latter is shown in Figure 1.1) split the encoding process in two different networks: one for preprocessing the secret image, independently of the host, and a second one for hiding it.

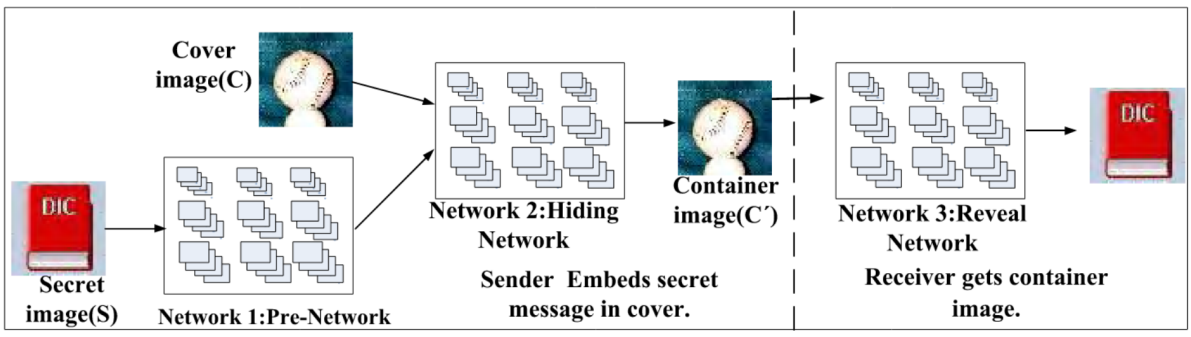

Figure 1.1: Architecture proposed by Duan *et al.*, to hide images inside images. Note that the encoding of the secret image is conditioned on the container.

While steganography is usually performed in digital data and it is often assumed that the transmission will be lossless, other works focus on the transmission of hidden signals through a physical medium. StegaStamp [5] is a deep learning-based system that is able to encode hyperlinks (encoded as bitstrings) into an image, which is later printed and scanned by a camera. Such setups need to account for the large amount of distortion that physical transmission introduces.

A closely related field to steganography is that of steganalysis, which aims to detect the presence of a hidden message in a signal; many techniques exist to tackle this problem [6, 7], with new approaches also relying on convolutional neural networks to analyse the data. This naturally leads to an adversarial framework for steganography, in which a discriminator is trained in parallel to the hiding network. SteganoGAN [8], depicted in Figure 1.2, relies on this approach to obtain a high capacity steganography system.

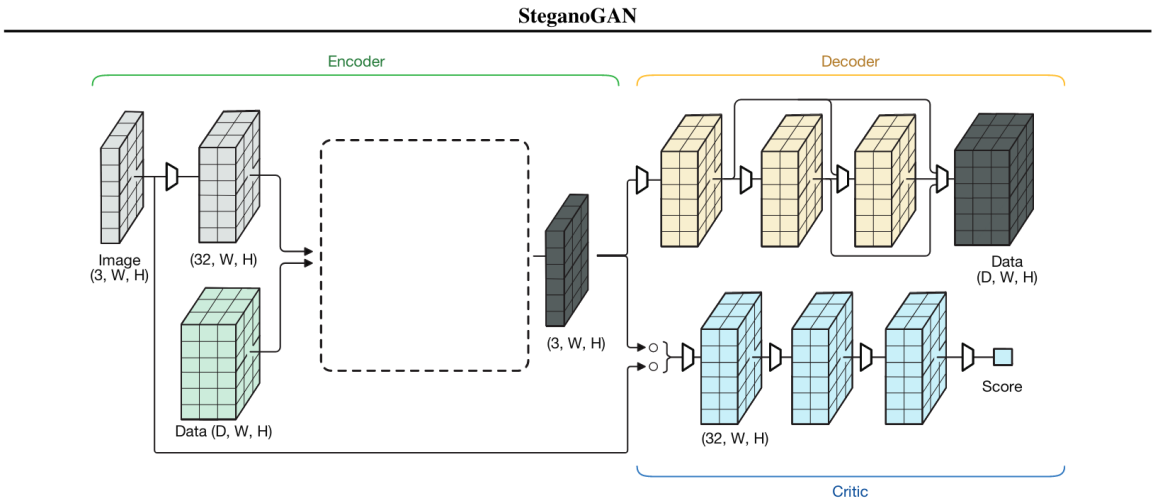

Figure 1.2: Architecture proposed by Zhang *et al.*, that, in addition to an encoder and decoder networks, uses a critic to perform steganalysis. The blank in the encoder diagram corresponds to the connection between the preprocessed image and the data, for which they propose multiple possibilities.

Another use for steganography is to perform backdoor attacks on systems such as neural networks. In these attacks the input (*e.g.* an image to be classified) is slightly distorted in an imperceptible manner from a human perspective, but causes the victim model (*e.g.* a convolutional neural network) to completely misclassify the data. [9] present two approaches that can trick state-of-the-art backdoor detection approaches.

Audio signals are also widely used in the steganography field. While some process the audio in the waveform domain [10], in order to hide an image within it is more common to apply a transform to the 1-dimensional audio signal to convert it into a 2-dimensional one, where previous image-based approaches can be used. [3] used a short-time Fourier transform for this purpose, while [11, 12] preferred a wavelet transform. A generic schema for the last one is depicted in Figure 1.3.

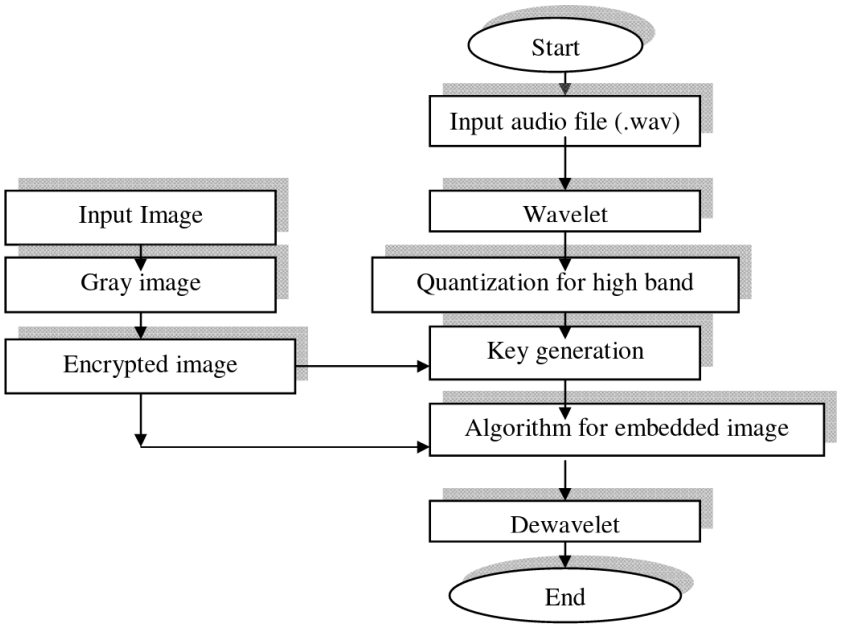

Figure 1.3: Architecture proposed by Hmood *et al.*, where a wavelet transform is applied to the audio waveform to obtain a 2D representation in which the image can be directly embedded.

1.2 PixInWav

PixInWav [13] introduced a novel application of using end-to-end deep steganography to encode image data into an audio signal. It used a short-time discrete cosine transform (STDCT) to get a 2D representation of the audio which allowed to add the encoded image in a residual manner.

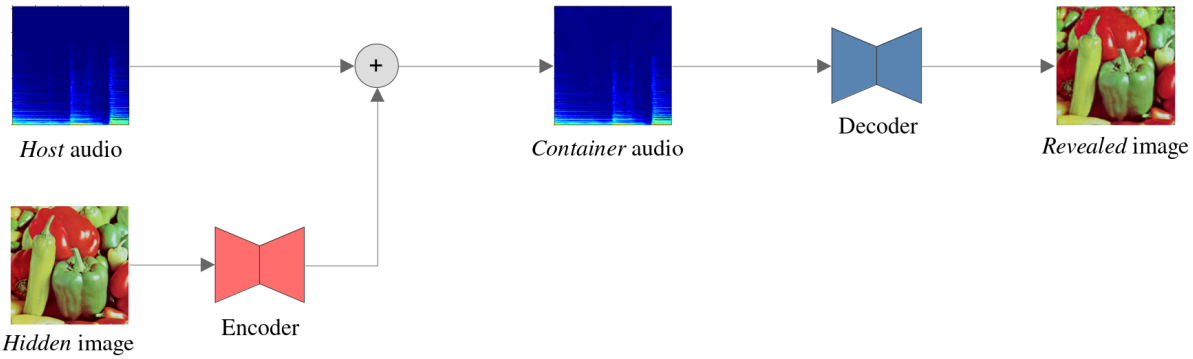

Figure 1.4: Basic schema of the PixInWav system: the hidden image is preprocessed and residually added onto the spectrogram, producing a distortion in the audio that should be unnoticeable. The decoder is then able to reveal the original image from the container spectrogram.

Chapter 2 - Technical background

The purpose of this chapter is explaining some of the concepts necessary to fully understand the thesis and that the reader is assumed to know on later chapters.

2.1 Convolutional neural networks

At the core of the system presented in this thesis is a deep learning architecture composed of convolutional neural networks (CNNs). These are a type of artificial neural networks that are widely used to process visual data, such as images or spectrograms.

CNNs are made up of convolutional layers. As opposed to fully-connected layers, where every input (*e.g.* a pixel of an image) would be connected to every output and scaled by a trainable weight, convolutional layers rely on a window of relatively small size (a kernel) that is convolved with the input. The only trainable parameters of a convolutional layer are the values of the kernel, which are much less than would otherwise be in a fully-connected layer; this makes the model lighter and less prone to overfitting.

It is very common for CNNs to be composed of multiple convolutional layers where the input data is downsampled in between, allowing the model to process it in a multiresolution manner.

As with other deep learning models, the system reads and processes a sample of data, computes a loss value and updates its parameters through backpropagation, resulting in a better model that is expected to obtain a smaller loss value on future attempts. This process is repeated many times (tens of thousands in our case) until the model eventually converges and stops improving.

2.2 Audio transforms

The system presented in this work relies on audio transforms to switch between the waveform and frequency domains. An audio signal can be digitally represented as a waveform by storing in a 1-dimensional vector the intensity values at any point in time, along with a frequency that indicates the rate at which the samples need to be played.

These 1-dimensional signals, however, can also be represented as a combination of frequencies by applying a mathematical transform such as the Fourier transform (FT), that results in a complex 1D vector.

Such representation, however, is not able to perceive changes in frequencies across time, which is the case in many audios. In order to achieve this, short-time (ST) versions of such transforms need to be

used; these operations compute a frequency transform every certain number of waveform samples and using only a limited amount of them. This results in a collection of 1D vectors that put together form a 2D structure: the spectrogram.

Such transforms are usually controlled by two hyperparameters: the filter length, that defines how many waveform samples to use when computing the transform every time; and the hop length, that defines the displacement of the window between two consecutive transforms. Changing this values will produce spectrograms of different shapes and characteristics.

Throughout this work the short-time Fourier transform (STFT) is used. It is a complex transform that produces both a magnitude and a phase, each being 2D signals of the same shape.

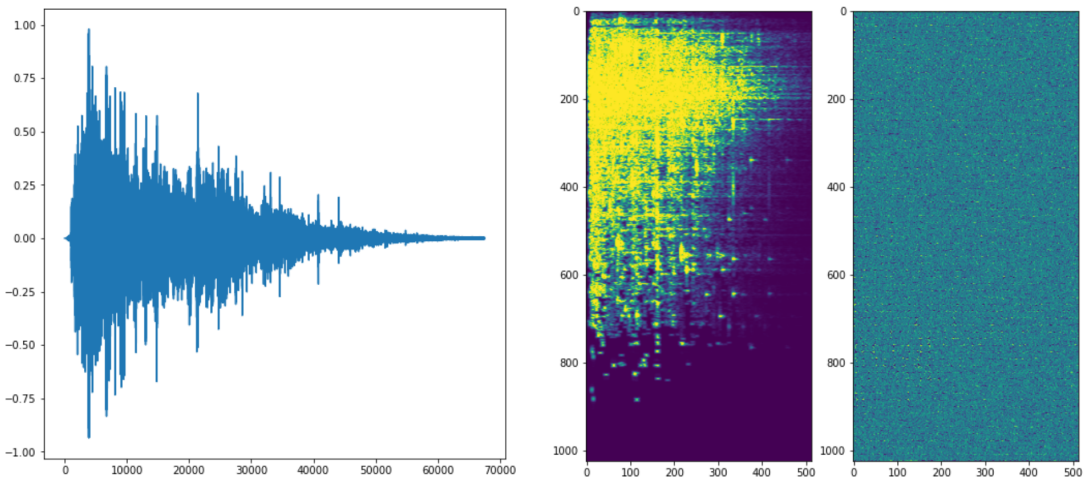

Figure 2.1: Example of audio waveform (left) and the magnitude (middle) and phase (right) obtained after applying the short-time Fourier transform.

An alternative transform which is also mentioned in this work is the short-time discrete cosine transform (STDCT), that works similarly to the STFT, the most relevant difference being that it is a real transform and thus it only produces a single 2D spectrogram.

Note that both transforms are reversible, *i.e.*, they have associated inverse transforms that convert the spectrogram back to the waveform.

2.3 Colour spaces

Colours in the real world can be expressed through a combination of three primary colours. For instance, printers will use different quantities of cyan, magenta and yellow to form any colour required. These sets of primary colours that are able to represent the whole spectrum define a colour space; any possible colour will be represented as a point in such a space.

There are many different colour spaces, each with its own advantages and particular traits that make it more or less suitable for specific applications.

In the digital domain, the RGB colour space is the most widely used. It represents every colour as a combination of red (R), green (G) and blue (B). It is an additive model, meaning that the addition of all colours makes white, as opposed to a subtractive one, such as the CMYK that printers use. Throughout

this work, unless indicated otherwise, the images can be assumed to be represented in the RGB colour space. This means that each pixel will have three associated values, one for each colour channel.

Another very widely used colour space in digital applications is YCbCr, which decomposes each colour into the luma component (Y) and the blue-difference (Cb) and red-difference (Cr) chroma components. What is particularly interesting about this model is that the luma component, corresponding to the greyscale value of the pixel, is perceptually much more important than the other ones, so the luma channel of the image carries much more information than the other two, and thus more than any of the individual RGB ones. This key concept is the basis for the architecture proposed in Section 4.7. Figure 2.2 shows the representation of a colour image in both the RGB and YCbCr colour spaces.

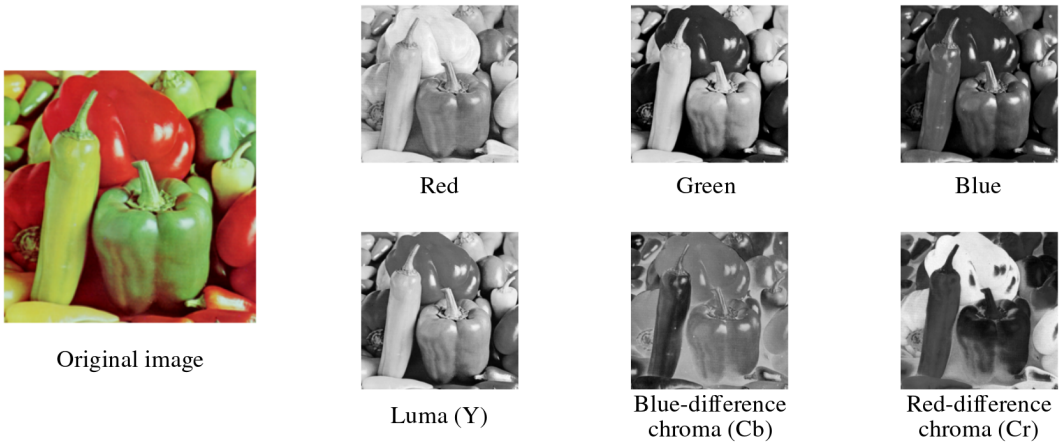

Figure 2.2: Example of the different colour channels of an image in RGB (top) and YCbCr (bottom) colour spaces. Note that every channel is 1-dimensional so it can be represented in a greyscale image, where lighter pixels indicate higher values and vice versa.

The remaining thing to note is that images represented in one colour space can be transformed to any other one. In our particular case, the conversion between RGB and YCbCr is done through a linear combination of the values, defined in the JPEG standard[1]:

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ Cb &= -0.1687R - 0.3313G + 0.5B + 128 \\ Cr &= 0.5R - 0.4187G - 0.0813B + 128 \end{aligned} \tag{2.1}$$

$$\begin{aligned} R &= Y + 1.402(Cr - 128) \\ G &= Y - 0.34414(Cb - 128) - 0.71414(Cr - 128) \\ B &= Y + 1.772(Cb - 128) \end{aligned}$$

The previous operations are applied individually to each pixel in order to convert the whole image from the RGB colour space to YCbCr and vice versa.

[1] https://www.w3.org/Graphics/JPEG/jfif3.pdf

Chapter 3 - Methodology

In this section we describe the details of PixInWav [13] 's original implementation. It is the model that will serve as a baseline for the work presented in Section 4.

The model has been coded in Python using the Pytorch framework[1]. The original code is publicly accessible at PixInWav's GitHub repository[2], but for this work it has been reimplemented from scratch with added features and fixes; the new code can also be found on GitHub[3]. Note that certain fixes from the original implementation make the results not comparable with the ones from [13].

3.1 Architecture

This section details how the PixInWav [13] model works. It takes an audio waveform and an image as input and outputs the container audio and revealed image. Figure 3.1 shows a more detailed diagram of the whole architecture.

The trainable part of the model consists of two neural networks: an encoder (*PrepHidingNet*), that preprocesses the original image and outputs the one that will be residually added onto the audio, and a decoder (*RevealNet*), that takes the audio container and outputs the revealed image.

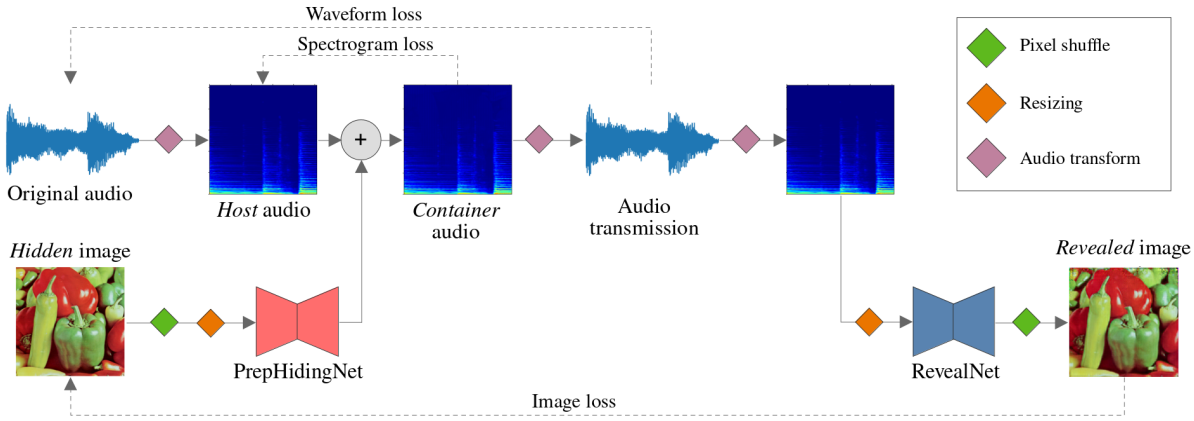

Figure 3.1: Detailed PixInWav architecture, showing the multiple operations that are applied to both the audio and the image, as well as the different losses used for training.

The next subsections will detail specific parts of the model as was presented in [13] and serves as a

[1] https://pytorch.org/
[2] https://github.com/margaritageleta/PixInWav
[3] https://github.com/migamic/PixInWav2

baseline for this work. The changes proposed for some of these components are detailed in Section 4.

3.1.1 Pixel shuffle

In order to flatten the RGB image into a single channel, a pixel shuffle [14] operation is applied: it arranges the 3 colour channels of each pixel side by side in a 2×2 grid, adding an empty element of value 0 as a fourth element.

This reversible procedure converts an image of shape $256 \times 256 \times 3$ into a flattened version of shape $512 \times 512 \times 1$. It has been shown to improve the quality of the colour of the recovered image.

3.1.2 Image resizing

The shape of the host spectrogram need not match the shape of the image. While images in the data set are of size 256×256 (512×512 after applying the pixel shuffle operation), the spectrograms are either of shape 1024×512 or 4096×1024, depending on the audio transform used, as is explained on Section 3.1.4.

This mismatch is overcome by stretching the image with linear interpolation, which is a reversible operation that allows to easily resize the image to any desired shape.

3.1.3 Encoder & decoder

Both the encoder and decoder are symmetrical U-Net [15, 4] convolutional neural networks. The input and output are a flattened RGB image of the same size as the container. Figure 3.2 shows an overview of the structure of the network and the shape of the tensors at every stage.

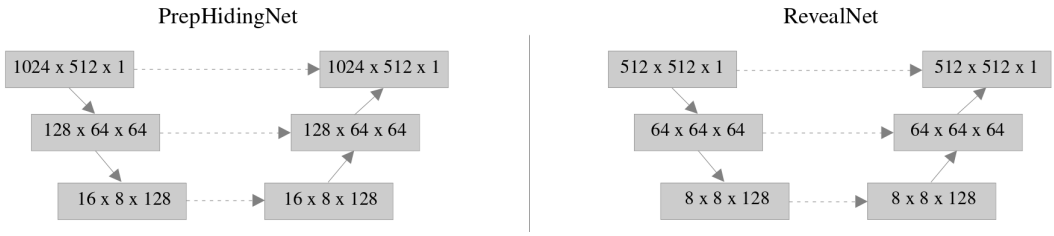

Figure 3.2: Schema of the encoder and decoder convolutional neural networks, showing the shape of the tensor at every stage. The first two numbers correspond to the spatial dimensions (height $\times$ width) and the third corresponds to the depth (number of channels).

The downsampling consists of two convolutional layers with a *max pool* operation. The upsampling is done through two transposed convolutional layers; after upsampling, the tensor is concatenated with the corresponding pair and two regular convolutional layers are used to halve the number of channels. After every convolutional layer there is a *Leaky ReLU* activation function.

The original architecture from PixInWav [13] also included batch normalization after every regular convolutional layer, but these have been removed for this work, since the performance has been shown to decrease when they are in use.

3.1.4 Audio representation

PixInWav [13] used the short-time discrete cosine transform (STDCT) to obtain the spectrogram of the host audio.

However, the short-time Fourier transform (STFT) is able to produce better results even when using a smaller container, as was shown in [16], a consequence of only using the magnitude not distorting the information in the phase.

In this work the audio is assumed to be of length 67522 (it is preprocessed to be so) and the STFT of filter length $2^{11} - 1 = 2047$ and hop length of $2^7 + 4 = 32$ is used to obtain a container of the required size of 1024×512.

3.1.5 Audio transmission

In this work we focus on the case of lossless audio transmission (*i.e.*, where the audio signal received is exactly the same as the one that was sent). This ideal scenario is not unrealistic, since most digital communication nowadays ensure that no data is lost or corrupted, such as the TCP protocol on Internet communication.

However, [13] showed that over-the-air transmission is far from lossless and the amount of distortion that is introduced makes the hidden signal almost unrecognizable at the receiving side. Adding artificial noise at the audio signal during training did not have much effect in generalizing to a real-life case.

We hope that the improvements suggested in this work apply to the noisy setup as well, but reproducing such experiments is out of the scope of this project and we leave it as future work.

3.2 Loss function

We use a loss function that leverages the distortion in the container audio and the recovered image. The model should try to minimize both at the same time.

$$\mathcal{L}(s, s', C, C', c, c') = \beta\|s - s'\|_1 + (1 - \beta)\|C - C'\|_2 + \lambda\|c - c'\|_1 \tag{3.1}$$

where s and s' are the original and revealed images, C and C' are the audio host and container spectrograms and c and c' their corresponding waveform representations.

The $\beta \in [0, 1]$ hyperparameter controls the trade-off between image and audio quality, with higher values giving more importance to the image and vice versa. $\lambda \in \mathbb{R}_{\geq 0}$ adds an additional audio loss to account for the waveform distortion.

Chapter 4 - Experiments and results

This section explains the main developments of this project, that build upon the previously described model. These mainly consist of analysing the current system and trying to improve its performance through different changes in the architecture.

4.1 Data sets

There are two separate data sets being used. For images it is a subset of 10.000 images from the *ImageNet Large Scale Visual Recognition Challenge 2012* (ILSVRC2012) [17] while for audio we use the *FSDnoisy18k*[18] data set. The image/audio pairs are chosen randomly to produce more variety in the training set.

Figure 4.1: Random examples from the data sets: ILSVRC2012 images (top) and spectrograms from FSDnoisy18k (bottom). Note that they have already been preprocessed to be the required shape.

Note that both data sets have associated labels which are never used in this project, since the model is entirely trained in a self-supervised fashion.

4.2 Metrics

Multiple metrics are used to assess the performance of the models: the structural similarity index (SSIM) and the peak signal-to-noise ratio (PSNR) for image, and signal-to-noise ratio (SNR) and the L1 norm (referred simply as L1 in the report) for audio. These are used throughout this section in order to compare the quality of the different architectures and setups.

SSIM [19] is a perceptual metric for similarity between a pair of images (the original and revealed in our case). It is a value in the range $[0, 1]$, with larger values indicating more similarity.

PSNR [20] is an objective metric often used to quantify the quality of an image after applying lossy compression. The values are in a logarithmic scale, where high numbers indicate a better quality. It is defined relative to the peak dynamic range, which makes it more suitable to compare image data rather than SNR.

SNR [21] is strongly related to PSNR. It also uses a logarithmic scale with high values indicating less distortion between the original and target signal. It is more commonly used for audio rather than image data.

The L1 norm is the sum of absolute errors of two signals. We use it to compare the original and container waveform in order to measure the distortion in that domain. Note that, unlike the previous metrics, this one is also used in the loss function to optimize the model.

4.2.1 Comparing models

Although the previous metrics are used to determine how good a trained model is, some setups can be hard to compare, since sometimes they will obtain better images at the cost of audio quality or vice versa. In such a scenario, determining which model is best would depend on the particular usage.

Also worth noting is that the previous metrics do not always represent human perception — *e.g.* distortion in audio signals with SNR values higher than 40 is usually imperceptible, so achieving even higher SNR values could be irrelevant from a practical perspective.

Throughout this section, a setup will be considered better than another if the metrics are better overall, even though in some practical cases the 'worse' setup could be preferred.

4.3 Training details

The models in this work have been trained for two epochs and, unless specified otherwise, with a batch size of 1, since it has proven to deliver the best results. The optimizer used was Adam with a learning rate of 0.001.

The values for the β and λ hyperparameters of the loss function used are 0.75 and 1 respectively, which were found to obtain a good balance between image and audio quality. For the sake of comparability between different models, these values remained unchanged unless specified otherwise.

4.4 Look inside PixInWav

We performed a detailed analysis of the trained models in order to interpret the processes that were being used for embedding the image into the audio.

4.4.1 Encoding

Observing the output of PrepHidingNet, which can be visualized as an image since it is a tensor of size 1024×512, we can see a dimmed and stretched version of the original image, but the overall contours and structure appear intact.

In order to better compare this tensor with the original image, a *restoration* of the output was performed, where the following operations were applied in order: (1) rescale to return to the original 512×512 shape; (2) pixel unshuffle to recover the three RGB channels from the flattened image; (3) negate the tensor, in order to visualize the image with the original colours instead of a negated version (this depends on the random initialization of the training process, which might cause the image to be encoded in negative and negated again at the decoder, with no difference in computation whatsoever); and (4) rescaling to the 0-1 range for the individual pixel values, the same as the original image.

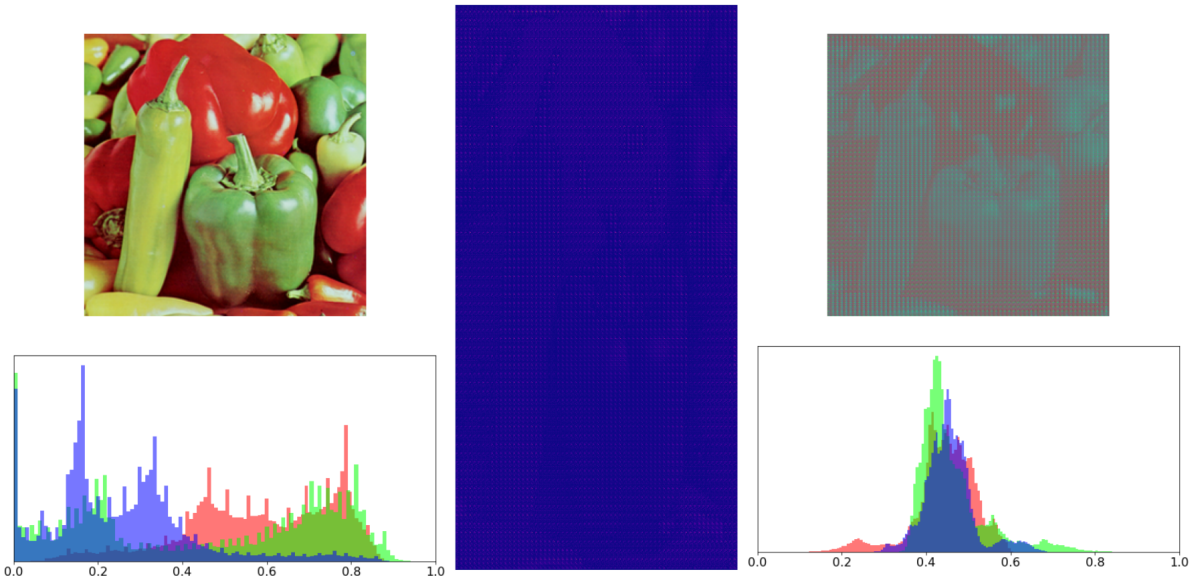

Figure 4.2: From left to right: original image, raw output of PrepHidingNet (notice the difference in dimensions) and restored image from the raw output. The first and last have a corresponding colour histogram underneath.

The image on the right of Figure 4.2 resembles a washed-out version of the original; while the colours have been distorted, the overall structure of the image remains. This shows that any preprocessing by PrepHidingNet is mostly done at a pixel level instead of treating the image as a whole.

The corresponding histograms show that colours tend to get compressed around the middle values, causing the image to lose contrast.

4.4.2 Embedding in the audio

For this step we looked at the distribution of values of the image and audio signals before and after being added.

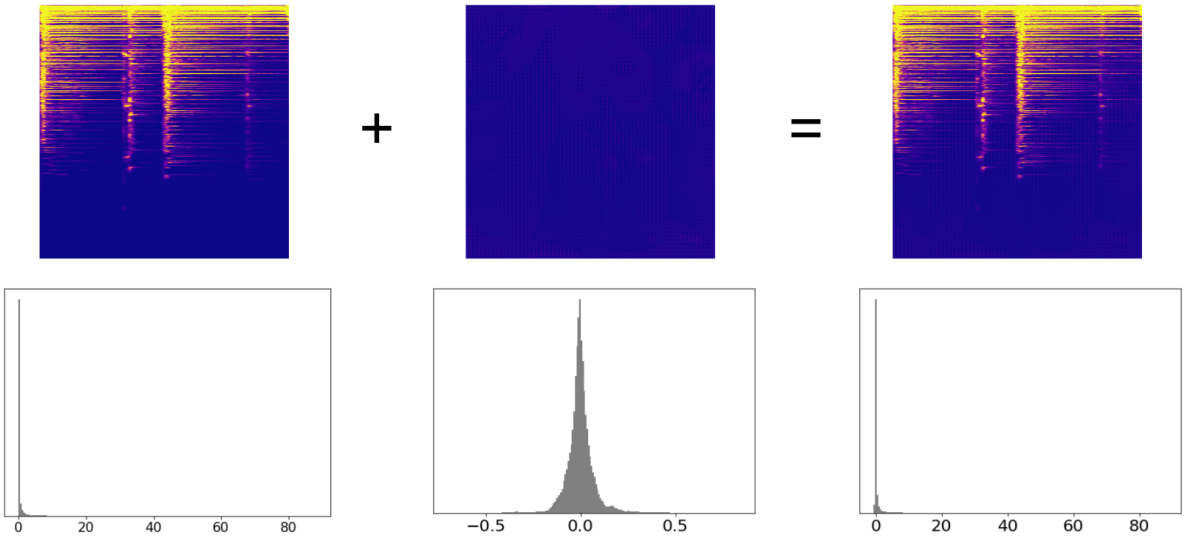

Figure 4.3: From left to right: original *host* spectrogram, preprocessed image (output from PrepHidingNet) and *container* spectrogram, corresponding to the addition of the two. The spectrograms have been rescaled to a square shape for visualization purposes. A histogram underneath each image shows its distribution of values (notice the change in scale for the horizontal axis).

What the histograms from Figure 4.3 show is that the range of values that the container and the preprocessed image use is very different in magnitude: while the spectrogram, although being mostly values close to zero, has very high values (around 80), the preprocessed image is in the range $[-0.5, 0.5]$. This behaviour resembles the least significant bit (LSB) approach. Notice that the pattern of the preprocessed image can still be observed in the empty regions of the container spectrogram but, being so small in magnitude, the distortion is very low.

4.4.3 Decoded image

The final step of the pipeline is the decoding of the image from the container spectrogram. To analyse the behaviour of the system as a whole, the revealed image is compared to the original one.

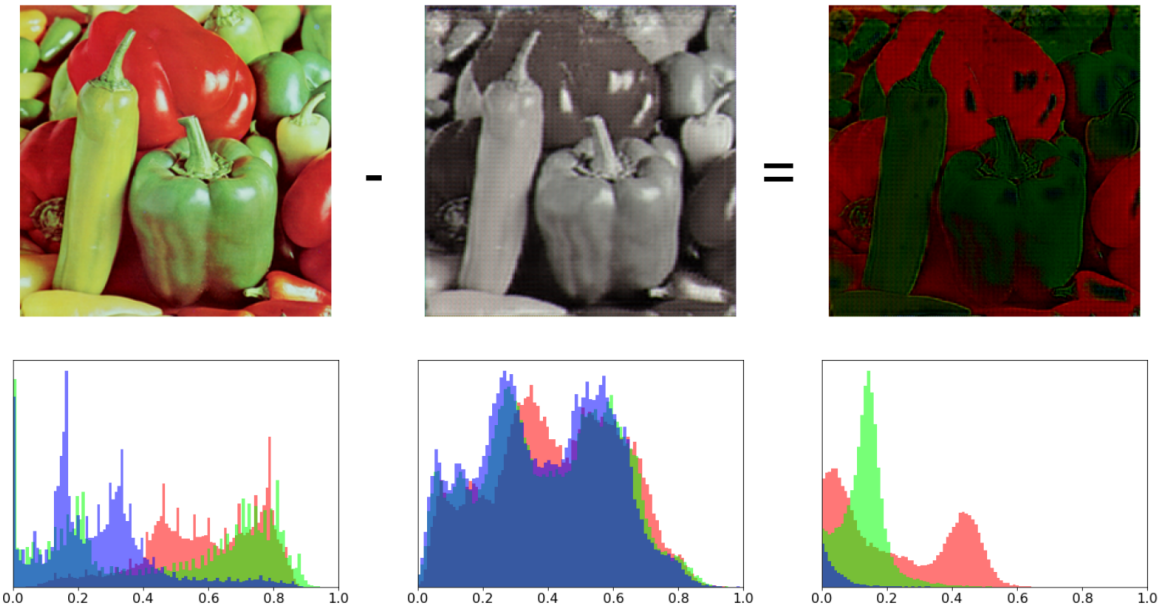

Figure 4.4: From left to right: original image, revealed image and difference of the two. A histogram underneath each image shows its colour distribution.

The revealed image lacks most of the colour and the top part is slightly distorted following the shape of the spectrogram in which it was embedded. The model is not powerful enough to be able to transmit colour images. Note that some hyperparameters, such as β from the loss function, could be tweaked to improve the quality of the revealed image at the cost of poorer audio quality.

4.5 Using STFT magnitude and phase

The short-time Fourier transform is a complex transform that produces both a magnitude and a phase from a single waveform. The existing architecture simply ignored the phase, as shown in Figure 4.5; here we examine its possibilities, first as a standalone container and then in combination with the magnitude.

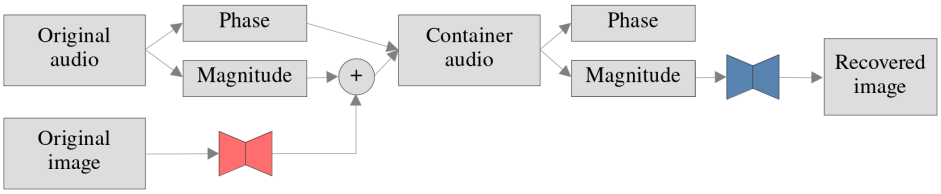

Figure 4.5: Original PixInWav architecture, that uses only the STFT magnitude as a container while the phase is left untouched.

4.5.1 STFT phase as a single container

The existing model can be easily adapted to use the STFT phase instead of the magnitude, as depicted in Figure 4.6, since they are both the same size. However, results show that the phase is less suitable for this task and the results are significantly worse.

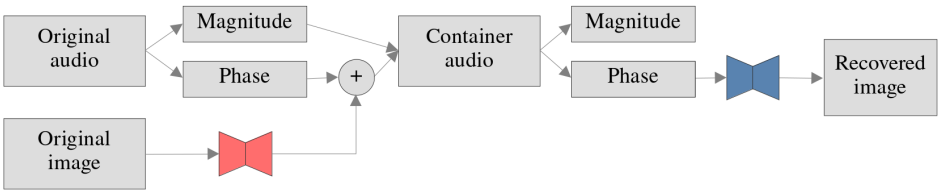

Figure 4.6: Modified architecture, using only the phase as a container. The adaptation is very straightforward.

Table 4.1: Comparison of metrics between using the magnitude or phase as a container.

Audio container	**SSIM ↑**	**PSNR ↑**	**SNR ↑**	**L1 ↓**
Magnitude	0.82	24.01	41.12	2E-4
Phase	0.30	12.67	23.46	1E-5

The results from Table 4.1 show that using the phase as the sole container is clearly inferior to using the magnitude. There is a very significant drop in both image and audio quality.

Our reasoning for this results are twofold: firstly, the phase is much more noisier signal in nature, as is shown in Figure 2.1, which makes the task of hiding information more difficult; secondly, small alterations in the phase produce a very noticeable distortion in the resulting audio, again making it very hard to add secret information.

4.5.2 STFT magnitude + phase

A more advanced setup was developed in which we used both signals as containers at the same time. This requires for the architecture to be adapted to handle multiple containers.

The two containers should be treated separately due to their very different structure. The architecture proposed, shown in Figure 4.7 uses different encoders and decoders for each. The two revealed images are fed into a third network that processes them to obtain a single image as output; out of the multiple solutions tried, a simple trained weighted average worked the best.

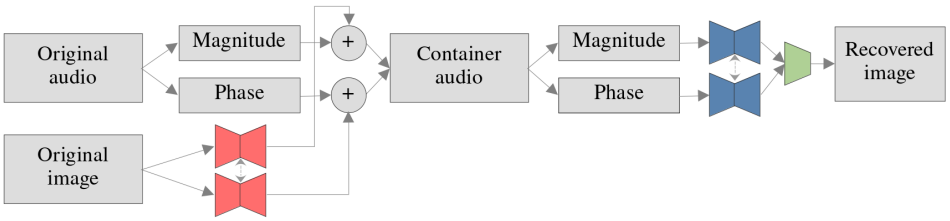

Figure 4.7: Proposed architecture, where both magnitude and phase are used as container signals.

The loss function also needs to be tweaked to accommodate for the possibility of using multiple containers. The proposed formula is:

$$\mathcal{L}(s, s', C_m, C'_m, C_p, C'_p, c, c') = \beta\|s - s'\|_1 + (1-\beta)((1-\theta)\|C_m - C'_m\|_2 + \theta\|C_p - C'_p\|_2) + \lambda\|c - c'\|_1, \quad (4.1)$$

where C_m and C_p now denote the magnitude and phase signals respectively, which are weighted by a new hyperparameter θ, that controls the trade-off between magnitude and phase distortion. Notice that the waveform c is still unique.

Since this loss function already explicitly accounts for the phase distortion, the waveform loss can be deemed redundant. We also explored the possibility of not using it (setting $\lambda = 0$).

Table 4.2: Comparison of performance of magnitude + phase models depending on the loss function hyperparameters.

Loss setup	**SSIM ↑**	**PSNR ↑**	**SNR ↑**	**L1 ↓**
Waveform loss ($\theta = 0, \lambda = 1$)	0.77	21.3	34.95	2E-4
Container loss ($\theta = 0.5, \lambda = 0$)	0.82	22.39	35.52	2E-4
Both losses ($\theta = 0.5, \lambda = 1$)	0.82	23.23	31.95	4E-4

The results from Table 4.2 show that the best setup consists of not including the waveform in the loss function, relying only in the MSE of the phase.

Finally, a comparison was performed between using the magnitude and phase as containers simultaneously or using only the magnitude. The results from Table 4.3 show that, while using both containers does increase the image quality slightly, there is a significant drop in audio quality, possibly a consequence of distorting the phase additionally.

Table 4.3: Comparison of metrics between using the magnitude alone or both the magnitude and phase as containers.

Audio container	**SSIM ↑**	**PSNR ↑**	**SNR ↑**	**L1 ↓**
Magnitude	0.84	24.84	43.51	2E-4
Magnitude + Phase	0.85	25.05	35.65	4E-3

Our conclusion is that it is not worth using the phase as a container, since it does not improve the metrics obtained with the baseline model that only uses the magnitude and thus there is no justification for the added overhead in the model.

4.6 New architectures

As is mentioned in Section 3.1.2, the original PixInWav architecture made use of linear interpolation for upsampling and downsampling the encoded and decoded image in order to make them the exact same shape as the container and the original shape, respectively. This strategy, named *Stretch* and depicted in Figure 4.8, although simple, does not make full use of the encoder and decoder capabilities, since the same amount of information is being carried in a container twice the necessary size (the encoded image after pixel shuffle is 512×512, whereas the container spectrogram is 1024×512).

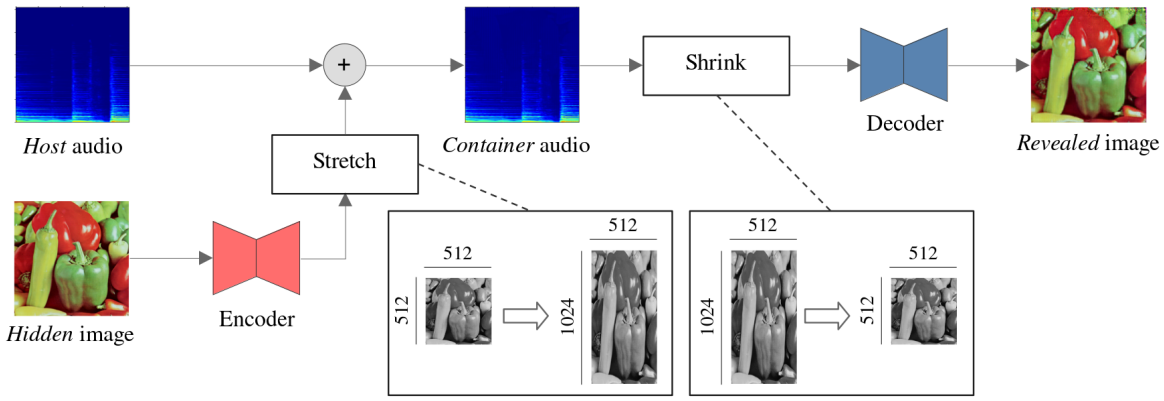

Figure 4.8: Overview of the default architecture (*Stretch*) used for embedding the audio onto the spectrogram.

In this section we propose alternative architectures to address this problem and make full use of the container size.

4.6.1 Replicate

This approach takes advantage of the encoded image being smaller than the spectrogram by adding multiple copies of it (instead of stretching a single one). In the default STFT case, the container is twice the size of the encoded image, so the system in able to fit two copies, thus carrying twice the information.

At the decoder side both concatenated replicas are revealed at once, split and averaged to produce the final outputted image. This is shown in Figure 4.9.

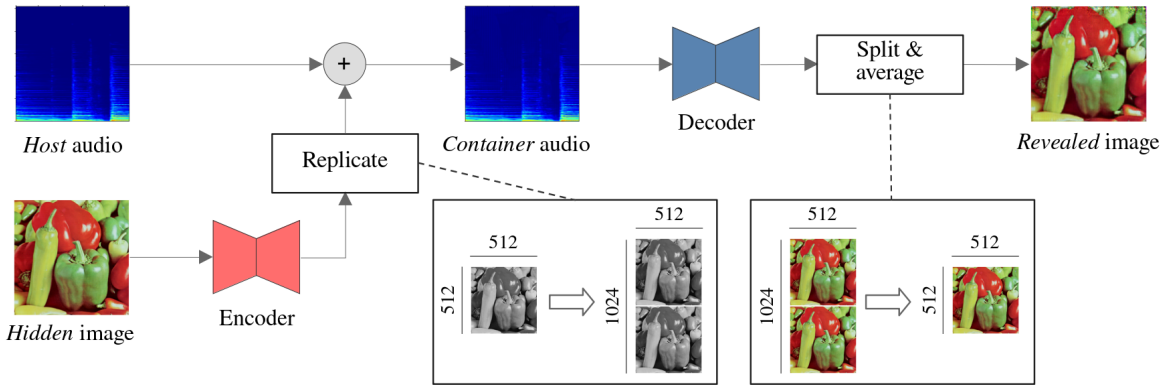

Figure 4.9: Overview of the *Replicate* method to embed the audio onto the spectrogram.

4.6.2 Weighted Replicate

As an improvement on the previous idea, *Weighted Replicate* (W-Replicate) scales each of the replicas before adding them onto the container spectrogram and also when merging them into a single one (essentially, a trained weighted average). Four trainable weights are used in total for the default STFT case where two replicas are used. This is shown in Figure 4.10.

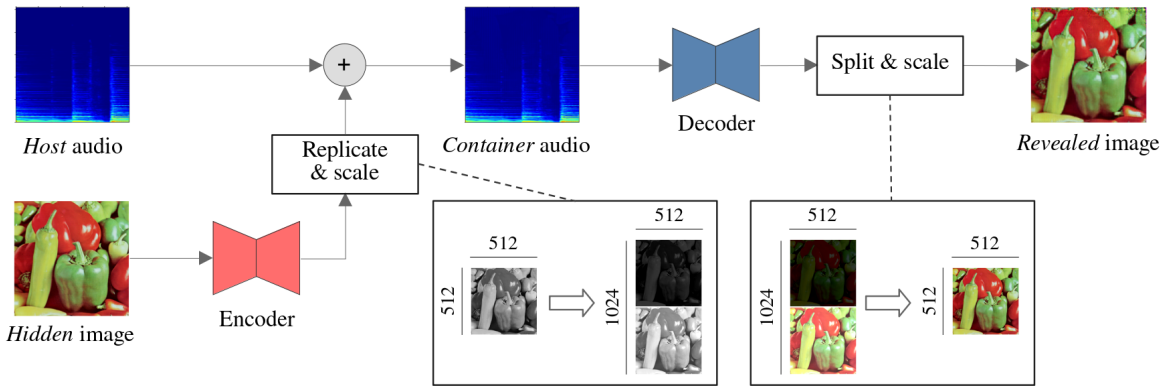

Figure 4.10: Overview of the *Weighted Replicate* method to embed the audio onto the spectrogram.

4.6.3 Weighted & Split Replicate

Building on top of the previous approach, *Weighted & Split Replicate* (WS-Replicate) splits the replicas before forwarding through the decoder, which takes N (N = 2 in the case of the default STFT container) input channels instead of 1. This allows processing the multiple replicas in a more intuitive and scalable manner. This is shown in Figure 4.11.

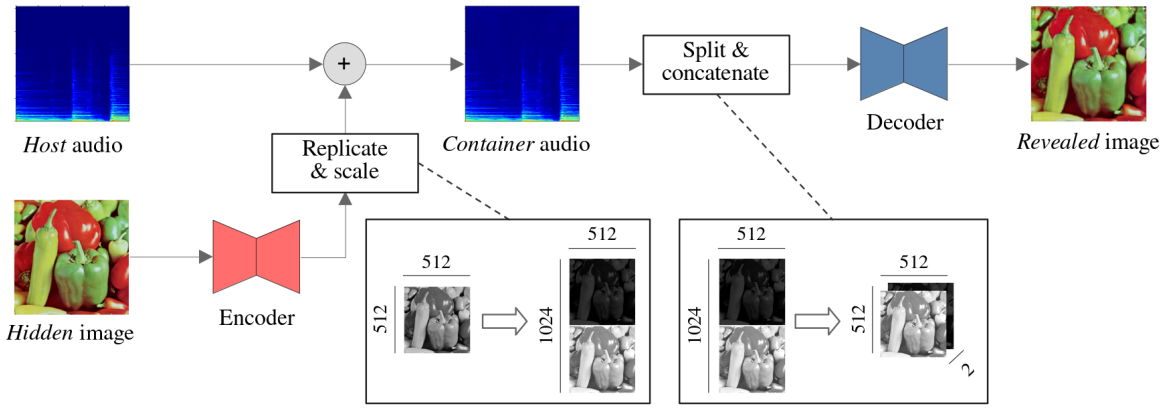

Figure 4.11: Overview of the *Weighted & Split Replicate* method to embed the audio onto the spectrogram.

4.6.4 Multichannel

Expanding on the previous idea, *Multichannel* has the encoder output multiple channels that will be directly used as the different copies of the image (instead of duplicating the exact same output).

While previously PrepHidingNet outputted 1 channel (corresponding to the flattened 3 colour channels), in this setup it outputs 8 channels, which will be used as 8 different images. This is possible because the possibility of outputting multiple channels makes the previous pixel shuffle operation no longer necessary, thus the encoded image is of size 256×256 instead of the previous 512×512.

As before, the replicas will be split before being fed to RevealNet, which will have as many input channels as needed (8 in this case). This is shown in Figure 4.12.

In this scenario, the model needs to learn how to represent the 3 colour values using the eight

different replicas. While this is initially more challenging, since it has no sense of colour, it also offers more flexibility in the way they can be encoded.

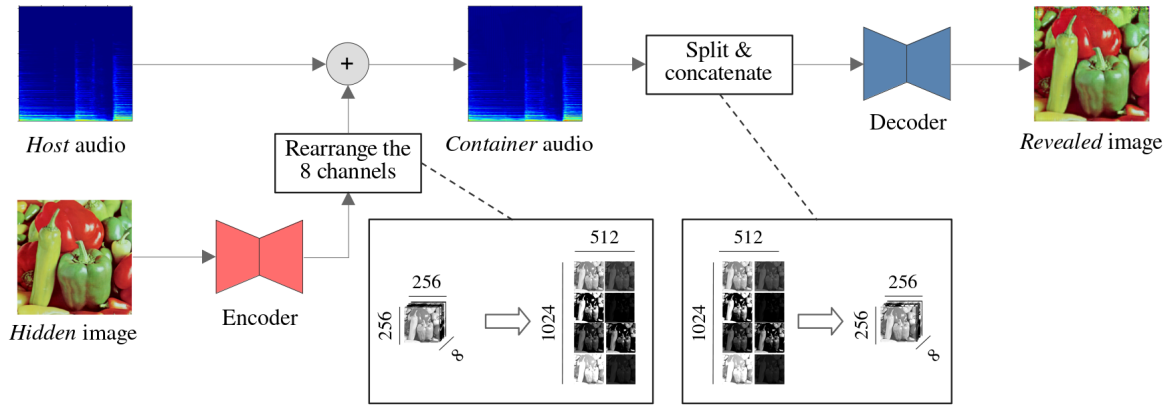

Figure 4.12: Overview of the *Multichannel* method to embed the audio onto the spectrogram.

4.6.5 Comparison of embedding methods

Table 4.4: Quantitative results obtained using the different models proposed above.

Embedding method	**SSIM** ↑	**PSNR** ↑	**SNR** ↑	**L1** ↓
Stretch	0.73	21.82	40.60	2E-4
Replicate	0.77	22.07	45.41	5E-5
W-Replicate	0.77	21.62	41.50	1E-4
WS-Replicate	0.76	24.40	37.54	2E-4
Multichannel	0.92	27.35	14.73	3E-4

Table 4.4 shows that *Multichannel* has a large increase in image quality, but at the cost of a drop on audio metrics, making the model less suitable than the others for most practical applications. On the other hand, all the replicate-based embedding methods outperform the baseline *Stretch*, with *Replicate* being slightly the best of the three, although the difference is very small.

However, the qualitative results shown in Figure 4.13 show that *WS-Replicate* is able to obtain a much better reconstruction of the original image, since it is the only one able to preserve the original colour.

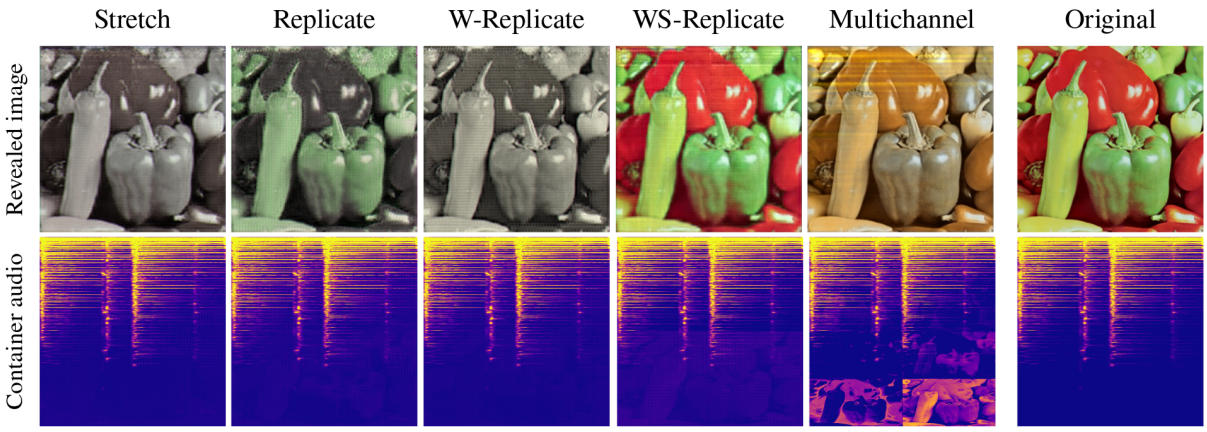

Figure 4.13: Qualitative results from the different models proposed above: the revealed image (top) and the audio container spectrogram (bottom).

The results from Figure 4.13 show that the model is learning to encode the image data in the high audio frequencies, since it is where the distortion is less noticeable. This can be easily observed in the *Multichannel* case, where the replicas on the high frequency area are very easy to see compared to the other ones, which are very dimmed (the fact that they are so clearly visible concurs with the low audio quality of this model).

WS-Replicate also shows this behaviour, where only the bottom replica is visible in the spectrogram; in the case of this model, this can be further confirmed by observing the trained scaling weights: 0.0013 and 0.2368 for the top (low frequencies) and bottom (high frequencies) replicas respectively, meaning that the model has learnt to only use the second replica, since the first one would distort too much the audio and be corrupted by the audio spectrogram in return.

4.7 Luma

This other approach aims to increase the efficiency of the models that use the pixel shuffle operation (i.e. all the previous ones but *Multichannel*). Instead of appending a 0 as the fourth component, it adds the luma of the pixel to be used as additional information.

When reversing the pixel shuffle at the decoding side, the luma is used as extra information to obtain a better image.

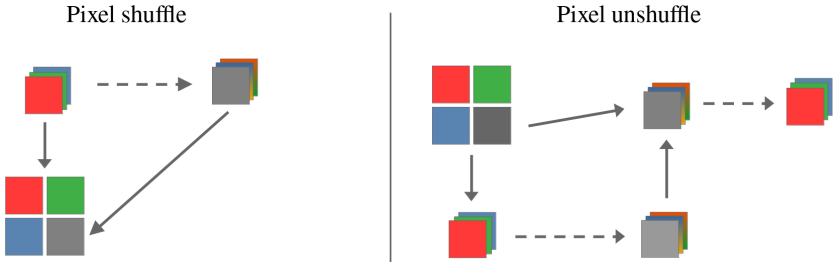

Figure 4.14: Schema of the pixel shuffle operation using the luma as the fourth component; dashed lines indicate a colour space conversion of the pixel.

Figure 4.14 depicts an overview of the process. The pixel shuffle operation, shown on the left, uses the luma of the pixel as the fourth value, instead of appending a zero. The pixel unshuffle, shown on the right, uses the three RGB channels to compute the YCbCr components of the pixel, averages the computed luma value with the fourth value to obtain a more reliable luma value and converts the pixel back to its RGB representation.

Table 4.5: Comparison of metrics between the baseline and the model using the proposed *Luma* architecture.

Model	**SSIM** ↑	**PSNR** ↑	**SNR** ↑	**L1** ↓
Baseline	0.73	21.82	40.60	2E-4
Luma	0.78	24.67	40.63	2E-4

Table 4.5 shows that this addition does improve the quality of the revealed images while maintaining the same audio quality as the baseline model. The improvement in image quality is better appreciated through the qualitative results shown in Figure 4.15, were it can be seen that the model is now able to reproduce colour, albeit a bit washed out.

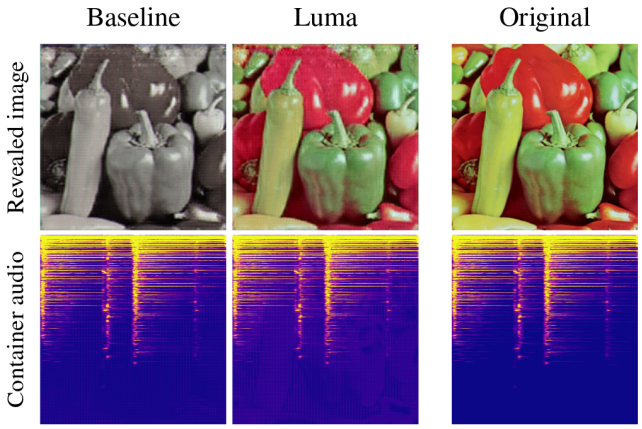

Figure 4.15: Qualitative results comparing the baseline model and the one with the luma modification.

4.8 Permutation

[16] considered the possibility of randomly permuting the pixels of the preprocessed hidden image before being added to the spectrogram and unpermuting them before feeding into RevealNet, as depicted in Figure 4.16. The expected benefits are both in audio and image quality:

- Any spatial pattern is broken before the image is added into the audio. This means that the distortion will be uniform among all frequencies. Additionally, from a secrecy perspective, the image is not visible anymore in the spectrogram.
- Likewise, the spectrogram structure is not visible on the revealed image anymore, which is a common problem when not using this technique. Instead, any distortion on the image to be revealed will resemble random Gaussian noise, which is less perceptually relevant.

The permutation can be fixed or changed at every iteration. In this work we only consider the fixed case, but there should be little difference since the convolutional model is unlikely to overfit a 1024×512 random permutation.

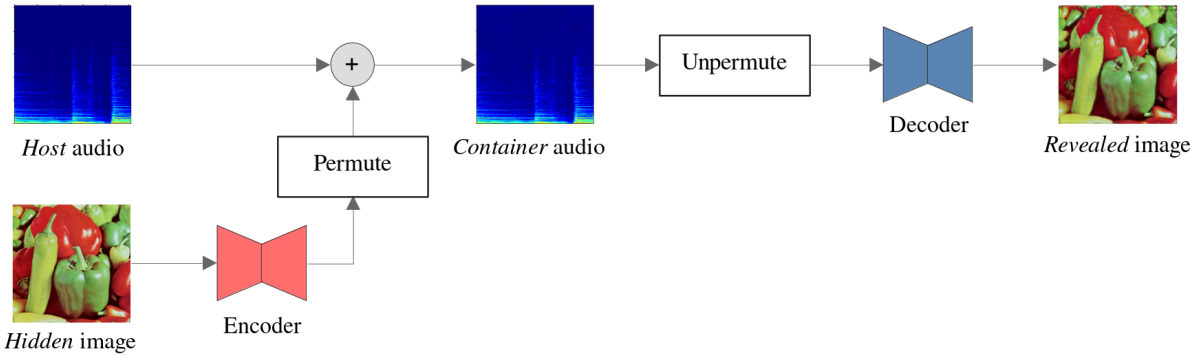

Figure 4.16: Simplified schema of the architecture, showing where the signal is permutted and unpermutted.

Table 4.6: Comparison of metrics for different embedding methods when using the permutation or not.

Embedding	**Permutation**	**SSIM** ↑	**PSNR** ↑	**SNR** ↑	**L1** ↓
Stretch	No	0.73	21.82	40.60	2E-4
	Yes	0.30	12.15	76.38	< 1E-5
Replicate	No	0.77	22.07	45.41	5E-5
	Yes	0.34	12.59	79.79	< 1E-5
W-Replicate	No	0.77	21.62	41.50	1E-4
	Yes	0.55	18.15	31.21	7E-4
WS-Replicate	No	0.76	24.40	37.54	2E-4
	Yes	0.59	20.44	30.95	7E-4
Multichannel	No	0.92	27.35	14.73	3E-4
	Yes	0.51	19.58	26.52	7E-4

[16] reported that the inclusion of this operation significantly decreased the performance of the models when used in another data set with the default architecture; our results, shown in Table 4.6, concur with those findings. While audio quality is increased very substantially for the *Stretch* and *Replicate* embedding methods, the drop in image quality causes the revealed image to be unrecognizable in the output, as can be seen in figure 4.17.

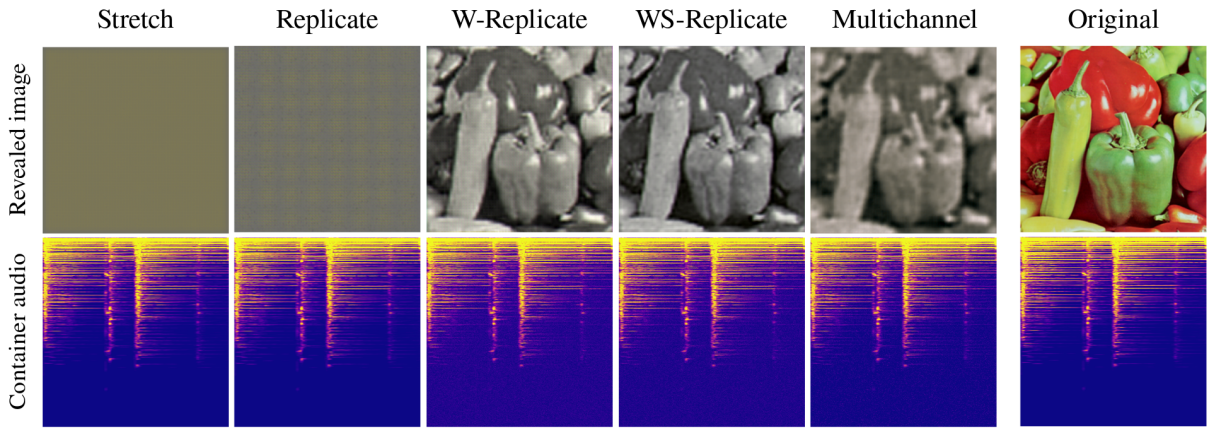

Figure 4.17: Qualitative results comparing the different model when using the permutation. Notice the decrease in image quality with respect to the results shown in Figure 4.13. Also notice that all the distortion in both the image and spectrogram is now in the form of unstructured noise (*i.e.*, the image can no longer be seen in the spectrogram and vice versa).

One of the supposed advantages of this operation is that the trainable part of the model is completely unaware of it (other than the distortion in the unpermutted container lacking the structure of a spectrogram), since the permutation happens after the image has been encoded and it is unpermutted before being decoded. This not only makes it easy to apply to any type of proposed architecture, but should, in theory, allow models trained without permutation to perform well when using it and vice versa. However, as is shown in Figure 4.18, this is far from being the case.

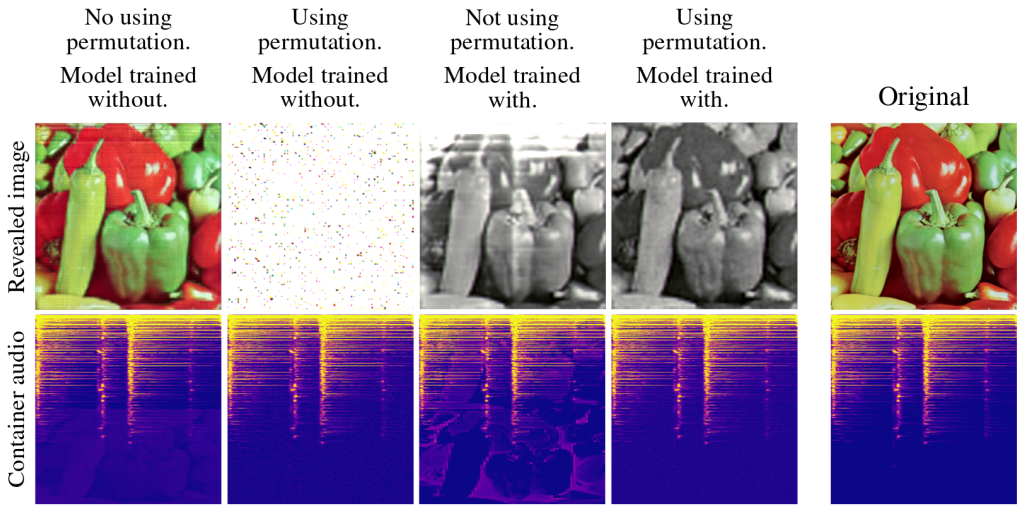

Figure 4.18: Qualitative results of models trained with and without permutations when being used in inference time with and without permutations. As can be seen the model does not generalize to the other case. It is using the *WS-Replicate* embedding, but the same conclusions apply to all the others.

4.9 Audio preprocessing

The PixInWav architecture assumes an audio of fixed length, 67522 samples, corresponding to around 1.5 seconds at the frequency of 44.1 kHz. However, the audio signals of the data set used are of variable length and are unlikely to be that exact same size. Originally this is dealt differently depending on

whether it is shorter or longer than the threshold:

- If the audio signal is shorter than the limit, it is padded on the right size with random Gaussian noise of low magnitude until the needed threshold.
- If the audio signal is longer than the limit, a chunk of the desired size is randomly selected from anywhere in the audio.

Note that all the random components of the audio preprocessing can be disabled during test time in order to ensure that the exact same audio signal is being used every time.

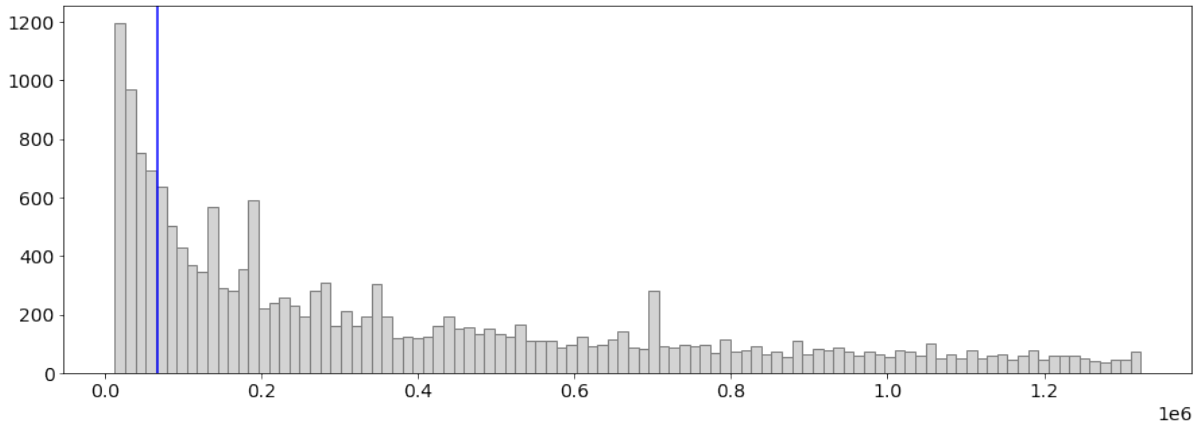

Figure 4.19: Histogram of the length of the audios in the training partition of the data set. The blue line corresponds to the audio threshold that is being used: shorter audios will be padded and longer ones will be cut. The number of audios left of the line correspond to 21.06% of the total.

4.9.1 Padding bias

Padding only the right side of the spectrogram caused the data to be slightly biased, since it was more likely to have empty (or, rather, noise-filled) regions at the right side.

In order to correct this while causing minimal changes to the original data, we added a random amount of padding at both the left and right sizes.

4.9.2 Padding noise

We removed the Gaussian noise in the padding area, preferring an all-zero signal which represents complete silence and is more likely to be found in a real-life scenario. Moreover, adding noise hardens the task of the model, which needs to encode the image into a less structured signal (as is shown in the results of encoding the image into the STFT phase from Section 4.5.1); thus, at inference time, it would be reasonable to pad a short audio with zeros.

4.10 Larger container

All the previous work has been done with a container of size 1024×512, which is determined by the STFT applied on the input audio waveform. However, these values are arbitrary and could be changed

through the STFT hyperparameters; in this section we explore the possibilities offered by the use of a larger container.

Increasing the size of the Fourier transform results in a larger size in the frequency dimension, while reducing the hop size of overlapping windows causes the container to increase in the time dimension. We applied both modifications to obtain a container of size 2048×1024.

The previous embedding methods explained in section 4.6 need to be adapted to accommodate for the larger container size.

- *Stretch* needs to interpolate to a larger target size, the same as the container.
- Replicate-based methods use 8 replicas instead of 2, which are arranged in the same manner as *Multichannel* used to. The number of trainable weights also increases by a factor of four.
- *Multichannel* can be expanded to use more channels. The architecture remains the same, but the number of channels (i.e. replicas) increases from 8 to 32.

Table 4.7: Validation metrics obtained for the different embedding types when using a large STFT container. Note that the *Multichannel* used a batch size of 2, since the training was very unstable otherwise.

Embedding	**STFT container**	**SSIM ↑**	**PSNR ↑**	**SNR ↑**	**L1 ↓**
Stretch	Small	0.73	21.82	40.60	2E-4
	Large	0.80	22.00	57.65	3E-5
Replicate	Small	0.77	22.07	45.41	5E-5
	Large	0.83	26.16	43.28	7E-5
W-Replicate	Small	0.77	21.62	41.50	1E-4
	Large	0.84	26.73	46.80	9E-5
WS-Replicate	Small	0.76	24.40	37.54	2E-4
	Large	0.85	26.71	34.31	4E-4
Multichannel	Small	0.92	27.35	14.73	3E-4
	Large	0.94	29.49	18.93	2E-3

The values from Table 4.7 show a very significant improvement when using a larger container, both in image and audio quality, as is to be expected from having more capacity for carrying information. From the qualitative results from Figure 4.20, it can be seen that the models show similar behaviour than when using a smaller container.

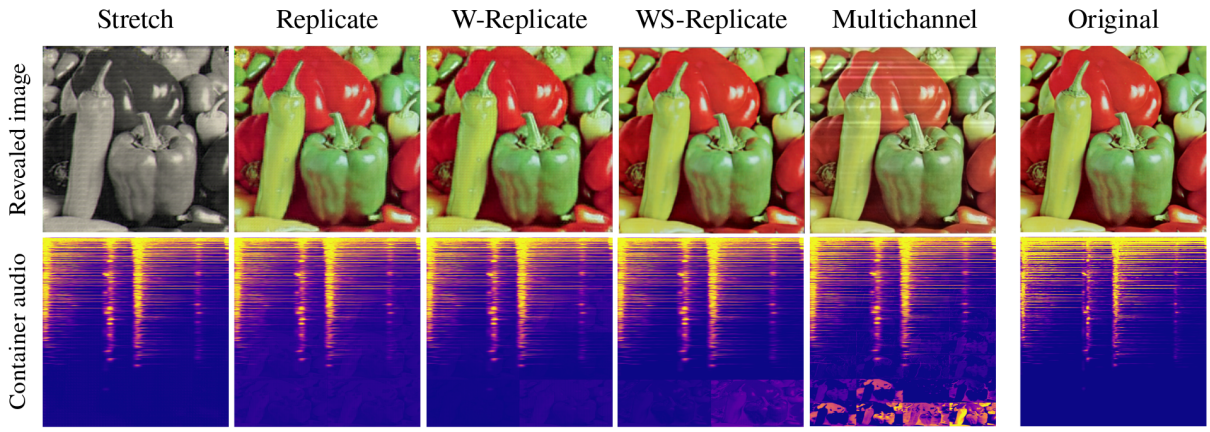

Figure 4.20: Qualitative results comparing the different models when using a large container. Notice the big improvement with respect to the results shown in Figure 4.13 from page 20.

This increase in performance comes at the cost of increased memory usage (twice as much) and longer training times (three times as much). Note that for training purposes the audio transform can be precomputed for every audio; however, the inverse transform will still be needed to compute the waveform loss so there will still be an increase in training time.

Chapter 5 - Ethics

This section is devoted to give a brief overview of the ethics and impact of this work.

5.1 Ethics of steganography

The core of this project is improving a steganography model. While it can be used for multiple different purposes, such as hiding metadata inside an audio file, the main purpose has always been to carry secret information in an inconspicuous medium.

This secrecy application is a double-edged sword that put in the right hands can be used for ethical reasons, such as watermarking songs to avoid piracy, but can also be used by malicious individuals to avoid detection by unsuspecting surveillance systems.

While we do not endorse or take responsibility for the usage of this tool for unethical reasons, we are committed to making it open to everyone in an effort to ensure total transparency, believing this is the way towards better and fairer research.

5.2 Environmental impact

At the core of PixInWav is a deep learning model, composed by two neural networks that need to be trained for hours for every individual experiment. While the number of parameters is relatively low for today's standards ($<$ 1 million) the amount of energy needed for training should be taken into account.

Several improvements have been made from the original PixInWav work, most notably using the faster STFT transform, training during less epochs and performing less validation steps, driving down the training time to a fraction of the original (more than 50h to only 2h).

The amount of electrical power needed to train an experiment is very variable and depends on a large number of factors, but can be roughly approximated to 300W, which during the average of 10 hours for each experiment to run leaves the energy consumption at around 3kWh.

A total of 44 experiments have been used for the development of work; however, taking into account erroneous attempts that had to be redone, this number easily surpasses one hundred tries, leaving the total energy consumption at more than 300kWh. While this is not a high number compared to today's state-of-the-art projects, it is still something to keep in mind for future works.

This should also be considered when putting this system into practice. For example, it can be a deciding factor when choosing the size of the container if peak performance is not the main priority.

Chapter 6 - Conclusions

In this work we presented in detail the existing PixInWav [13] architecture, analysed the behaviour of the system from the inside and proposed several modifications in an attempt to improve its performance.

The analysis of the model showed that the encoder and decoder are mostly imitating a least significant bit approach, where the image is hardly modified before being added into the audio using values of much smaller magnitude.

From the different proposed modifications to the architecture, some proved to work better than others:

- Using the STFT phase produced very bad results overall, while using it in combination with the magnitude did not increase the quality in any significant way.
- Some of the new proposed architectures for embedding the image onto the spectrogram did improve the metrics obtained with the baseline *Stretch* method. The replicate-based methods were shown to be the best, with little difference between them.
- Including the luma as the fourth value of the pixel shuffle operation allowed the model to make full use of the container space and was shown to produce a slight improvement in performance.
- Adding the random permutation of pixels did not work as expected and caused a significant decrease in image quality to all models tried, albeit for some the audio quality did improve.
- Using a larger container, resulting from changing the STFT hyperparameters, caused the metrics to improve significantly in all models tried. The increase in capacity of the container allows for more information to be transmitted, at the cost of computational resources.

6.1 Future work

This last section is devoted to outline directions of research for the future, that were considered during the development of the thesis but were out of the scope of this project.

6.1.1 Luma for Replicate-based embedding methods

Explicitly transmitting the luma channel has been shown to improve the image quality of the baseline model and so has the use of Replicate-based embedding methods. Since the latter use the same pixel

shuffle operation as the baseline *Stretch* embedding, the luma model could also be applied to these embedding methods to hopefully achieve even better performance.

6.1.2 Flipped replicas

One of the main advantages of the architectures above that are based on replicating the encoded image throughout the container space is that the same information is being transmitted more than once, making it easier to recover the original image from the noisy audio.

However, due to the strongly non-random nature of most spectrograms, it is sometimes the case that the same part of the image is polluted with noise in all the replicas, cancelling the benefit of replicating in the first place.

Our suggested fix for this scenario consists in randomly rotating and flipping the multiple copies of the image in a way that can be reversed at the receiving side. This way it is more unlikely that the exact same part of the image will be corrupted.

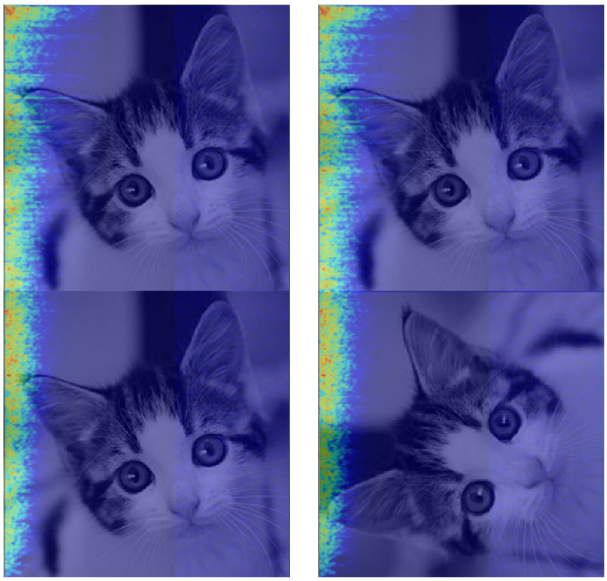

Figure 6.1: Example of a scenario where flipping the replicas could improve the quality of the recovered image. The two copies have been added one on top of the other, as would be the case for any of the *Replicate* embedding methods. On the left, none of the copies have been altered and the left side of the recovered image will most likely be distorted; on the right, the bottom replica has been flipped and rotated, ensuring that there is little overlap between the polluted areas in the two copies.

6.1.3 Attention for multichannel

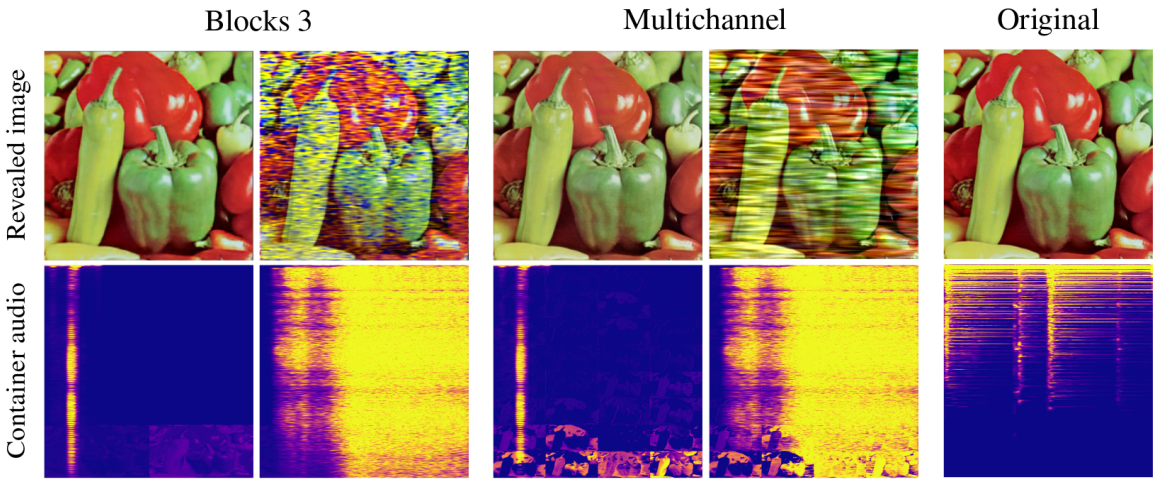

Figure 6.2: Behaviour of repilca-based models when using very different audio spectrograms. The image quality is substantially degraded when very high frequencies are present.

It has been observed, as shown in Figure 6.2 that the models that use many replicas, such as *Multichannel* or *Replicate*, learn to encode most of the information in the high frequencies, since that area of the spectrogram is usually empty. However, when that is not the case, the resulting image quality is significantly worsened. A model that is not aware of the audio when encoding has no way of countering this effect.

The original PixInWav [13] architecture was shown not to benefit from conditioning the image preprocessing on the audio, making a residual setup preferable due to its simplicity. However, for some of the proposed new architectures this could no longer be the case, since the redundancy of the multiple replicas allows for a very natural attention setup [22] where they can be scaled depending on the particular shape of the spectrogram (*i.e.*, placing the image information where it less disturbs the audio). A schema of such an architecture is shown in Figure 6.3.

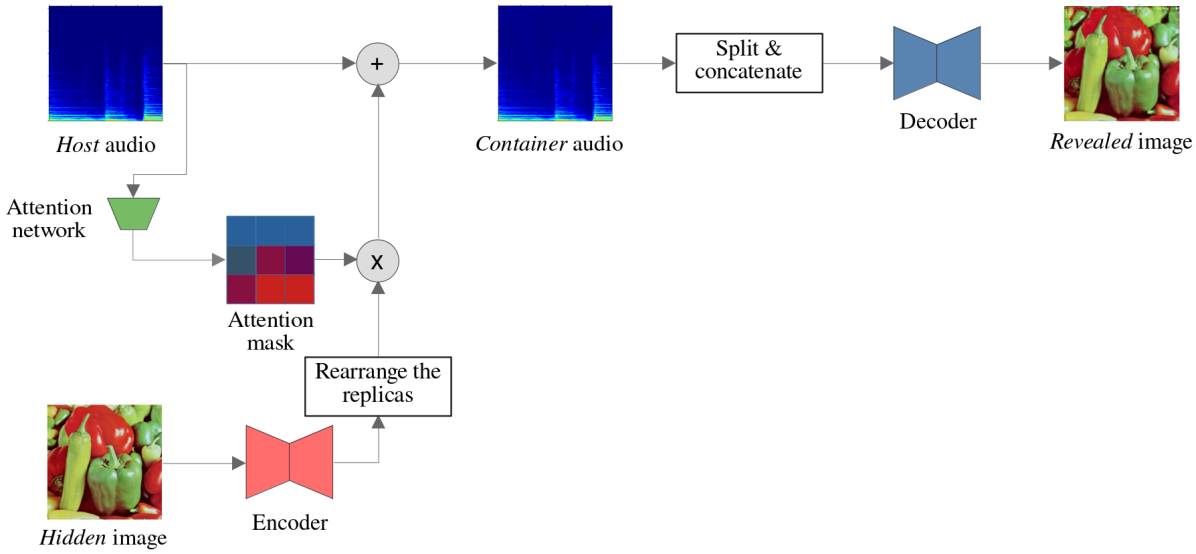

Figure 6.3: Proposed architecture that uses an attention network to scale the multiple replicas. In such a setup the image preprocessing is no longer independent from the audio.

A more advanced setup could condition PrepHidingNet on the attention weights. If the multiple encoded images are not exact copies of each other, as is the case for the *Multichannel* embedding, the type of information contained in each could vary depending on the expected distortion from that particular audio.

6.1.4 Generalize for lossy transmission

As is mentioned in Section 3.1.5, the current setup only considers lossless audio transmission. If this were to be used over a noisy channel such as over-the-air transmission the conclusions obtained in this work could not apply since, for example, the best architectures proposed need not be the most resilient to noise.

Also worth investigating in the case of real-life digital implementations is the problem of lossy compression, which is strongly related to noise. Again, the obtained conclusions should be reexamined because, for instance, the STFT cannot be made indefinetely large to obtain more container capacity, since only low frequencies will be preserved.

A thorough exploration should be carried out to corroborate that the previous conclusions apply in these new scenarios.

6.1.5 Perceptual loss

One of the problems that current architectures face is that of a distortion that is perceptually very noticeable to a human, such as the shape of the spectrogram appearing in the revealed image. The current loss function, build from L1 and L2 losses for individual components, fails short for this kind of task.

We suggest trying other losses that take into account human perception, such as the ones proposed in [23] , in order to minimize the perceived amount of distortion in the resulting signals.

Alternatively, a multi-modal adversarial loss could be considered, where a separate discriminator network is trained to discern between the original and modified signals (the audio container and the revealed image), such as the one presented in [8]. The encoder and decoder networks thus need to learn how to fool the discriminator by producing less polluted signals.

6.1.6 Allow for audios and images of variable size

The current architecture assumes predefined audio and image sizes. This is the case for the image data set, and the audio signals are cut or padded to make them the right size. However, this constraint limits the usability of the whole system in the real world.

To overcome this limitation, the STFT transform's hyperparameters could be changed depending on the image resolution in order to produce a host spectrogram of the required shape.

Alternatively, the process of embedding of the image into the audio could be made dependant on the respective original and target sizes. For example, the *stretch* operation can be easily modified to obtain any desired size while the methods that are based on replicating the image can use a varying number of copies.

6.1.7 Embedding in the waveform

PixInWav [13] has always used the spectrogram of the audio as a container for the image; this is very convenient because both signals are 2-dimensional and thus can easily be added. However, another possibility is adding the hidden image into the waveform directly, avoiding having to compute frequency transforms altogether.

Such a system would need to find a way to flatten the image into a 1-dimensional signal. A naïve flattening of the image with the original resolution (possibly having been previously preprocessed by PrepHidingNet) would produce a 1D vector of length $256 \times 256 \times 3 = 196608$, so an audio of this length at least would be required (at the frequency of 44.1 kHz this would roughly correspond to 4.5 seconds), which is more than most of the audio samples in the data set used.

More advanced setups could rely on convolutional layers to reduce the size of the image before or after flattening and upscaling it afterwards, making the system able to work for shorter audio signals.

Bibliography

[1] Tayana Morkel, Jan Eloff, and Martin Olivier. "An overview of image steganography". In: Jan. 2005, pp. 1–11.

[2] Shumeet Baluja. "Hiding Images in Plain Sight: Deep Steganography". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 2066–2076. ISBN: 9781510860964.

[3] Felix Kreuk et al. "Hide and Speak: Towards Deep Neural Networks for Speech Steganography". In: *Proc. Interspeech 2020*. 2020, pp. 4656–4660. DOI: 10.21437/Interspeech.2020-2380.

[4] Xintao Duan et al. "Reversible Image Steganography Scheme Based on a U-Net Structure". In: *IEEE Access* 7 (2019), pp. 9314–9323. DOI: 10.1109/ACCESS.2019.2891247.

[5] Matthew Tancik, Ben Mildenhall, and Ren Ng. "StegaStamp: Invisible Hyperlinks in Physical Photographs". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[6] Jessica Fridrich and Miroslav Goljan. "Practical steganalysis of digital images: state of the art". In: *Security and Watermarking of Multimedia Contents IV*. Ed. by Edward J. Delp III and Ping Wah Wong. Vol. 4675. International Society for Optics and Photonics. SPIE, 2002, pp. 1–13. DOI: 10.1117/12.465263. URL: https://doi.org/10.1117/12.465263.

[7] Konstantinos Karampidis, Ergina Kavallieratou, and Giorgos Papadourakis. "A review of image steganalysis techniques for digital forensics". In: *Journal of Information Security and Applications* 40 (2018), pp. 217–235. ISSN: 2214-2126. DOI: https://doi.org/10.1016/j.jisa.2018.04.005. URL: https://www.sciencedirect.com/science/article/pii/S2214212617300777.

[8] Kevin Alex Zhang et al. "SteganoGAN: High Capacity Image Steganography with GANs". In: *ArXiv* abs/1901.03892 (2019).

[9] Shaofeng Li et al. "Invisible Backdoor Attacks on Deep Neural Networks Via Steganography and Regularization". In: *IEEE Transactions on Dependable and Secure Computing* 18.5 (2021), pp. 2088–2105. DOI: 10.1109/TDSC.2020.3021407.

[10] Daewon Lee, Tae-Woo Oh, and Kibom Kim. "Deep Audio Steganalysis in Time Domain". In: *Proceedings of the 2020 ACM Workshop on Information Hiding and Multimedia Security*. IH&MMSec '20. Denver, CO, USA: Association for Computing Machinery, 2020, pp. 11–21. ISBN: 9781450370509. DOI: 10.1145/3369412.3395064. URL: https://doi.org/10.1145/3369412.3395064.

[11] R.A. Santosa and P. Bao. "Audio-to-image wavelet transform based audio steganography". In: *47th International Symposium ELMAR, 2005.* 2005, pp. 209–212. DOI: 10.1109/ELMAR.2005.193679.

[12] Dalal Hmood, Khamael Abbas, and Mohammed Altaei. "A New Steganographic Method for Embedded Image In Audio File". In: *International Journal of Computer Science and Security* 6 (Apr. 2012), p. 135.

[13] Margarita Geleta et al. "Pixinwav: Residual Steganography for Hiding Pixels in Audio". In: *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2022, pp. 2485–2489. DOI: 10.1109/ICASSP43922.2022.9746191.

[14] Wenzhe Shi et al. "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2016, pp. 1874–1883. DOI: 10.1109/CVPR.2016.207.

[15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015.* Ed. by Nassir Navab et al. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4.

[16] Teresa Domènech Abelló. "Hiding images in their spoken narratives". PhD thesis. UPC, Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona, Departament de Teoria del Senyal i Comunicacions, Feb. 2022. URL: http://hdl.handle.net/2117/365843.

[17] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[18] Eduardo Fonseca et al. "Learning Sound Event Classifiers from Web Audio with Noisy Labels". In: *CoRR* abs/1901.01189 (2019). arXiv: 1901.01189. URL: http://arxiv.org/abs/1901.01189.

[19] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[20] Shaofeng Li et al. "Invisible Backdoor Attacks on Deep Neural Networks Via Steganography and Regularization". In: *IEEE Transactions on Dependable and Secure Computing* 18.5 (2021), pp. 2088–2105. DOI: 10.1109/TDSC.2020.3021407.

[21] Phillip Isola et al. "Image-to-Image Translation with Conditional Adversarial Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2017, pp. 5967–5976. DOI: 10.1109/CVPR.2017.632.

[22] Chong Yu. "Attention Based Data Hiding with Generative Adversarial Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.01 (Apr. 2020), pp. 1120–1128. DOI: 10.1609/aaai.v34i01.5463. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5463.

[23] Hang Zhao et al. "Loss Functions for Image Restoration With Neural Networks". In: *IEEE Transactions on Computational Imaging* 3.1 (2017), pp. 47–57. DOI: 10.1109/TCI.2016.2644865.

Appendix

The code for the original PixInWav implementation, along with the instructions on setting up the environment and running it, can be found on GitHub: `https://github.com/margaritageleta/PixInWav`.

As is mentioned before, the code used in this project is a complete reimplementation of the original, with more features and fixes. It is also hosted on GitHub: `https://github.com/migamic/PixInWav2`.