James King

Audio Super-Resolution: A Deep Learning Approach

Computer Science Tripos – Part II

Pembroke College

May 14, 2021

Declaration

I, James King of Pembroke College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose. I am content for my dissertation to be made available to the students and staff of the University.

Signed: James Alexander King

Date: 13^{th} May 2021

Proforma

Candidate Number:	**2335E**
Project Title:	**Audio Super-Resolution: A Deep Learning Approach**
Examination:	**Computer Science Tripos – Part II, 2021**
Word Count:	**11,379**[1]
Final Line Count:	**2,348**[2]
Project Originator:	Alexander Campbell
Project Supervisors:	Ramon Viñas Torné with Professor Pietro Liò & Alexander Campbell

Original Aims of the Project

This project aims to analyse some of the current methods of performing audio super-resolution. I specifically wanted to introduce an approach based on the generative adversarial network design to see if it would improve the existing models. I also chose to implement models that both had a pre and post upsampling approach to the problem and analysed how they compared to each other.

Work Completed

Firstly, I implemented the AudioEDSR model. A model which I adapted from a popular post-upsampling image super-resolution architecture. Next, I implemented the AudioUNet. This was based on a paper. Finally, I implemented a new generative adversarial network over the top of the AudioUNet and called this new model AudioUNetGAN. The AudioEDSR proved to be surprisingly effective for low upsampling rates. I also investigated the effects artifacting has on the models.

[1] Calculated using `detex DIS.tex | tr -cd '0-9A-Za-z \n' | wc -w` (for the chapters between the introduction and conclusion inclusive)

[2] This was computed by `find . -name '*.py' | xargs wc -l`

Special Difficulties

None.

Contents

Chapter 1

Introduction

Audio Super-resolution is the process of artificially increasing the sample rate of a signal. It is a common approach to performing **bandwidth extension**, which is the process of expanding the frequency range of a signal. These terms are often used interchangeably in literature, and the Nyquist-Shannon Theorem [1] shows that sample rate and signal frequencies are proportional to each other. There are several applications of this type of super-resolution work. It can be used in speaker verification networks, improving the quality of telecommunications, audio style transfer networks, and any system that involves generating audio signals.

This project aims to apply several modern deep learning methods to the problem of audio super-resolution. It will also compare and contrast how these methods perform against each other and against several non-machine learning baselines.

An audio signal can be represented as a function of time $w(t) : [0, L] \to \mathbb{R}$ where L is the duration of the signal, and $w(t)$ is the amplitude of the signal at time t. In order to operate on this data computationally $w(t)$ needs to be discretised into $\tilde{w}(t; S) : \{\frac{1}{S}, \frac{2}{S}, \ldots, \frac{SL}{S}\} \to \mathbb{R}$ where S is the sampling rate or **resolution** of the discretised signal $\tilde{w}$. Suppose the same signal is discretised with two different resolutions $u = \tilde{w}(t; S)$ and $l = \tilde{w}(t; \hat{S})$ with $S > \hat{S}$. This project aims to learn a model over $\mathbb{P}(u|l)$ by assuming that $u = f_\theta(l) + \vec{\epsilon}$ for a model f_θ with parameters θ and $\vec{\epsilon} \sim \mathcal{N}(0, 1)$.

1.1 Motivation

With the World Health Organisation declaring a "Public Health Emergency of International Concern" in January 2020 [29], much of the world started rapidly moving to online communications. People started talking to friends and family online, conducting virtual business meetings and many schools moved to online learning. An influx on this scale requires a large amount of data to be transmitted globally, and while this project

has implications for machine learning-based audio compression, it is not the goal of this project. As many people have found with the transition to online communication, it is often difficult to understand what people are saying with the microphones built into many laptops and mobile phones generally being of a low quality. These microphones often have a naturally low-sample rate and bit depth, making them unsuitable and unpleasant for modern communication.

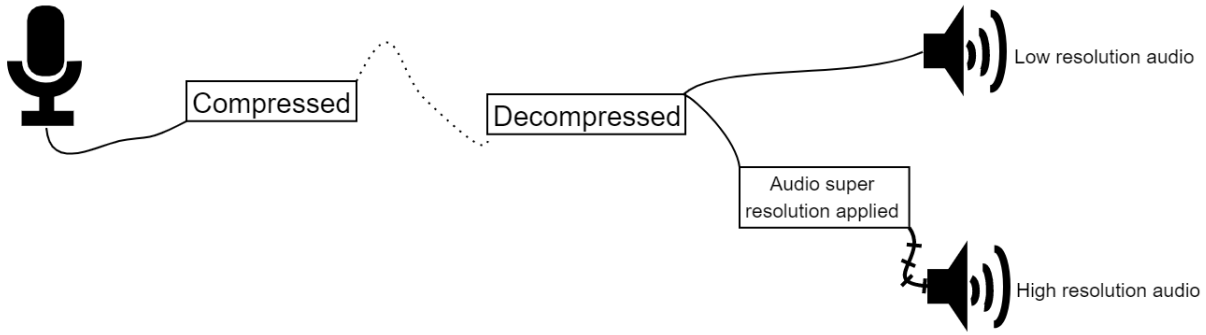

Figure 1.1: Example of a real-world application of the project. This also shows the distinction between super-resolution and compression since compression can only restore a file to the same resolution it was compressed from. It makes sense to apply super-resolution after any compression has occurred since it will inflate the file size.

This project aims to make communication cheaper and more enjoyable for people unable to pay a premium for high-quality microphones by applying state-of-the-art machine learning techniques to real-time generative audio modelling—something increasingly important during the ongoing pandemic.

1.2 Challenges of the Project

Direct modelling on raw audio signals can be computationally expensive. Particularly creating a real-time system can be resource-heavy since it needs to process thousands of samples per second. Generative audio models have also seen significant development in the last few years, and it is still an ongoing area of research. An especially understudied area is artifacts in audio, with them being notoriously complicated to deal with.

Another challenge was that I had no previous experience in creating models for deep learning. This lack of experience meant I had to quickly learn about the field and research different deep learning libraries. Additionally, this being in such an active area of research meant that the work for this project involved reading a lot of academic papers, the style of which I was not used to.

1.3 Previous Work

Previous works into audio super-resolution can be broadly split into two main areas: Interpolation pre-deep learning and methods that use deep learning.

1.3.1 Interpolation: Pre-Deep Learning

There are several straightforward ways of performing signal interpolation that do not require large complex networks; instead, they can be represented as functions based on simple heuristics. The methods discussed can be visualised in Figure 1.2. Perhaps the simplest of these is to assume the signal's amplitude is the same as at the latest recorded timepoint. This method is called flat or piece-wise interpolation as it appears to be a flattened version of what the signal might be.

Another standard method is to perform linear interpolation, which assumes the amplitude of the signal changes linearly from the previously recorded value to the next. While this generally produces a slightly better approximation to the desired signal, it requires additional information about the next point. However, the computation can still be performed in real-time for signals with a high frequency.

The primary way to perform interpolation is the spline method. This is an improved version of polynomial interpolation, which fits polynomials to the points but has an inaccuracy where the edges will oscillate from the desired value (Runge's phenomenon); the spline method fixes this problem. The spline method requires more computation than the previous two. However, with enough knowledge of the future signals, it can still interpolate a signal in real-time and typically leads to a more accurate prediction.

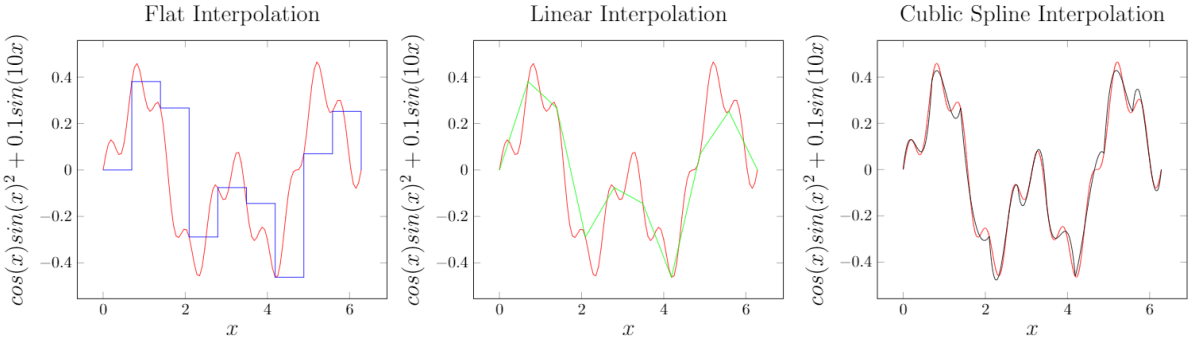

Figure 1.2: This figure shows a comparison of interpolation methods that do not use machine learning. Each method is given the values of the same 10 sample points. For application to audio signals, the x-axis would be time, and the y-axis would be amplitude, with the function much more complex.

1.3.2 Super Resolution: Deep Learning Era

There have been several successful deep learning models that perform audio super-resolution. Some of these have involved using preprocessed feature extraction methods

[7] which requires a lot of domain-specific signal processing knowledge to implement the filters. Convolutional Neural Networks (CNNs) are an example of how to improve upon this model. CNNs are used to perform feature learning, meaning the model learns the optimal filters on its own, requiring less input from the programmer. CNNs can produce better results without the need for as much domain-specific knowledge. This is demonstrated in a recent paper about the AudioUNet [15].

Generative adversarial networks (GANs) are also a relatively new idea in machine learning. There have been many attempts at using GANs to improve existing models, such as a recent paper building on top of the AudioUNet [22]. GANs generally consist of two networks, one that generates samples, and a new network whose goal is to discriminate between samples taken from the real and generated distribution. This new network encourages the generator to produce more realistic samples than when trained without. They are, however, notoriously difficult to train, particularly on one-dimensional data.

In 2016, researchers from DeepMind worked on an audio synthesis model called WaveNet [10]. WaveNet uses layered dilated convolutions to generate raw audio. This method is very quick to train. However, generating audio with this method will take a long time since generating each sample requires a whole pass through the network. For telephone quality audio, the model will need to generate around 10,000 samples for every second of audio. WaveNet produces a more realistic sounding output that its predecsors (parametric and concatenative methods), but the long synthesis times make it infeasible for any real-world application, such as telecommunications. One year after the creation of WaveNet, DeepMind researchers designed a new variant to address the slow synthesis time, and parallel WaveNet was created [17]. In a similar style to a GAN, this network introduces a new major sub-network to increase the quality of the outputs. However, instead of an 'adversary', the introduced sub-network is a teacher. The teacher scores the quality of the generator at each sample which is an improvement over the adversary because it also judges how close the generated sample was to a real one. While this fixes the issue of slow synthesis, it now takes a lot longer to train, mainly because the teacher needs to be as good as the upper bound of the generator.

There have also been many recent advances in other, significantly more widely researched, super-resolution fields, such as image super-resolution. An early example of the work done for using deep learning to perform image super-resolution was produced in 2014 and made use of CNNs [5]. This paper gives a good insight into the subject and involves techniques such as **patch extraction** to allow training over smaller datasets. Another example of recent innovation in this field is the MetaSRGan produced in a project last year [26] and adapted a commonly used deep learning model to work on arbitrary scales by learning how to generate the weights of an up-sampling network instead of just learning weights for a single scale.

Chapter 2

Preparation

This chapter introduces the core ideas behind the frameworks used in this project. It also discusses why artifacts can be produced when upsampling one-dimensional signals. A detailed requirements analysis and a discussion of the tools used for the implementation are also included.

2.1 Concepts

This section is designed to give the reader a brief introduction to neural networks before giving an overview of deep generative modelling in the context of audio synthesis.

2.1.1 A Short Introduction to Neural Networks

A neural network consists of a group of artificial neurons grouped into layers. These layers of artificial neurons are commonly referred to as single layer perceptrons. The input of an artificial neuron is the output from a subset of neurons from the previous layer $\vec{x}$ combined with a constant factor and a weight for each input defines each artificial neuron. The weight applied to the constant input is referred to as the bias and is denoted w_0. Each neuron also contains an activation function $\sigma : \mathbb{R} \to \mathbb{R}$, which alters the output and aids the training process. Figure 2.1 shows the structure of a neuron.

Single layer perceptrons are only able to learn linearly separable features. For learning features that are not linearly separable, MultiLayer Perceptrons (MLPs) can be used. MLPs are perceptrons that have a known input and output space with one or more hidden layers between them. These layers are fully connected, meaning the output of each artificial neuron is input to every artificial neuron in the subsequent layer. MLPs also have the desirable property that they are fully feedforward. Feedforward neural networks are those in which information only flows in one direction so that there are no cycles in the network, and each perceptron is visited at most once in a forward pass.

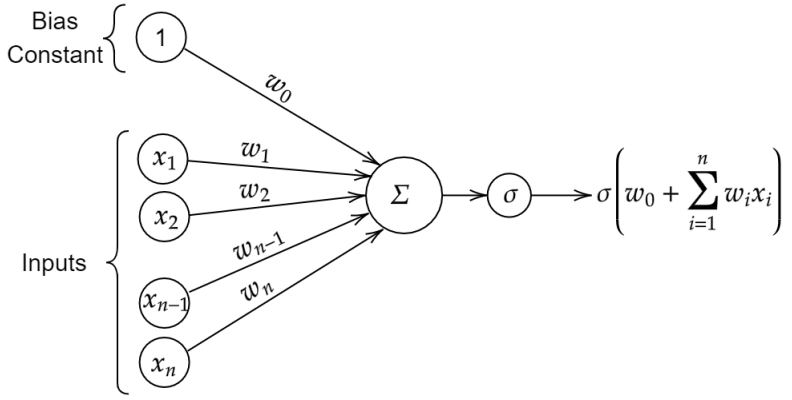

Figure 2.1: Structure of an artificial neuron

Networks containing any "feedback" connections are commonly referred to as Recurrent Neural Networks (RNNs). The feedback loops in RNNs can act as a form of memory, allowing the current pass to consider the previous passes. Feedback connections make RNNs useful for time series modelling. RNNs do have some drawbacks, such as being notoriously difficult to train with gradients used in backpropagation frequently exploding or vanishing on specific inputs, meaning unknown inputs may cause the model to produce excessive noise. However, some recent RNN variants largely address this issue (LSTMs, GRUs). RNNs also require enough of a signal to be passed through it before making use of its memory and introducing unwanted noise. Requiring past knowledge also restricts the ability of the network to run in real-time. Instead of RNNs, this project will focus on designing a fully feedforward approach to audio super-resolution.

2.1.2 Convolutional Neural Networks

In 1980, Kunihiko Fukushima developed a neural network based on a model of the visual nervous system [3]. This was highly successful and was the first to include convolutional layers. The modern Convolutional Neural Network (CNN) is heavily based on this model and is one of the most utilised architecture types in deep learning. Although the CNN was originally designed to work in a three-dimensional domain, the core ideas are not limited to this dimension and have seen great success in one-dimensional domains. The ideas are introduced in a one-dimensional context since it is the most helpful in understanding their applications in this works problem domain.

Convolutions

In mathematical analysis, a convolution is an operation that takes two functions f and g and returns a new function $f * g$. This convolved function represents how much these functions affect each other in shape and is defined as an integral over dummy-variable τ for time-point t:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(t - \tau) g(\tau) d\tau$$

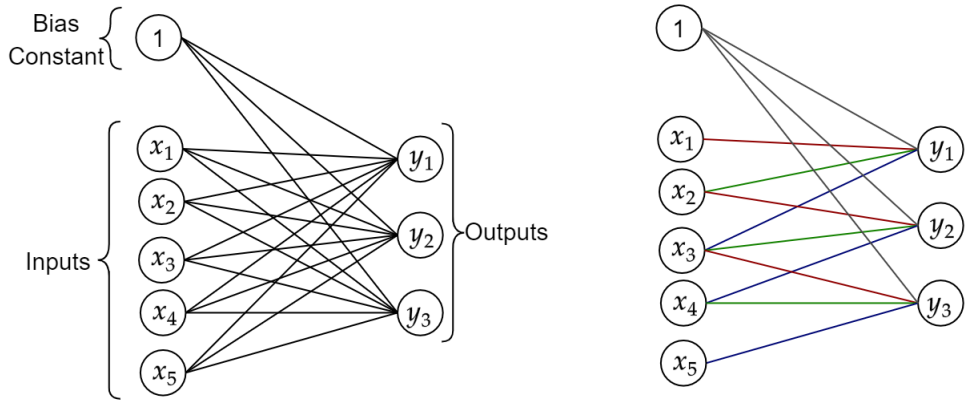

Figure 2.2: An example of a classic perceptron (dense layer) with 18 unique weights to learn (left). A convolutional layer with a kernel size of 3 so only has 4 unique weights to learn (right).

Since this project deals with a discrete set of numbers M, it will make use of the discrete approximation for a convolution defined for input n:

$$(f * g)[n] = \sum_{m \in M} f[m]g[n - m]$$

Convolutional Layer

Since this project uses functions defined over a discrete set of values, a convolutional layer can be defined as follows:

- Let g be a function corresponding to an input vector
- Let f be a function corresponding to a carefully chosen filter
- So $(g * f) = h$ gives us a vector representing the effect the filter has on each position.

So f can be chosen such that it has more of an effect on g when the section it is affecting has the desired features. h is commonly referred to as the feature map for this reason.

The Complete CNN

Backpropagation is typically used to calculate the weights in a CNN. A CNN consists of blocks of convolutional layers followed by an activation function and a pooling layer[1]. After these blocks, a fully connected layer or two are often used for classification.

There are several compelling reasons CNNs have seen a lot of success in the field of machine learning. One of these is the idea that CNNs have translational invariance. This means that CNNs learn a feature map that will produce the same results when applied to any sub-sequence of input data. This approach removes the need for highly specific feature extraction, and instead, features are extracted in this deep network. These features are

[1]Pooling layers are used to reduce the dimensionality of the output while using heuristic functions to give a summary of the input features

extracted in an end-to-end manner, meaning they are changed to optimise the output's quality.

Another benefit is that the repeated application of the same weight on the network means there are fewer weights to learn, making it faster to train. Figure 2.2 shows this parameter sharing aspect. The elements next to each other are also closely linked, meaning it is ideal for problem domains with this project's structure. Models in this project actually use a CNN without any dense layers. This type of network is called fully convolutional.

One of the key hyper-parameters of the convolutional layer is the stride. The stride represents how far apart the kernel takes its samples from. Figure 2.3 shows how the stride affects a convolution with kernel size three.

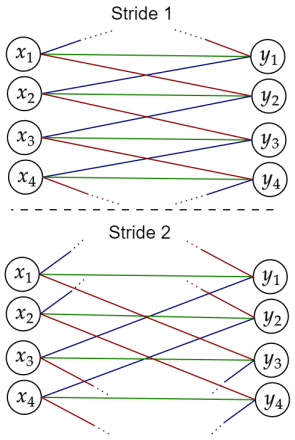

Figure 2.3: This figure shows the distinction between a convolution with stride 1 and stride 2. Both convolutions have kernel size 3

Another variation on the convolutional layer is a deconvolution[2]. A deconvolution is the inverse of a convolution. Given the same hyper-parameters, it effectively restores the signal to the correct dimensions.

2.1.3 Generative Adversarial Networks

It has been a long-established practise to use some kind of discriminator to determine whether or not the output of a network is realistic. Traditionally this role was played by the human who refines the model until the data looks about right. This process was soon automated. However, they still required a lot of very specialised, problem-specific input from the creator. This practice was generalised for deep learning in mid-2014 by Ian Goodfellow [4], near the start of the modern deep learning research era. Allowing this discriminator to be deep means the specialised knowledge required to perform tasks was reduced and allowed for more general models to be created. This simplistic approach has seen huge success as one of the main types of deep learning models, with many adaptations optimised for different tasks.

As proposed by Ian Goodfellow, the basic GAN structure consists of a generator and a discriminator network. The generator learns a distribution over a latent space of randomised audio samples, and the discriminator learns to distinguish real samples from those generated. This approach effectively produces two separate networks at the end, and the generator and discriminator can be separated. GANs are a strong implementation framework for deep generative models.

In the GAN approach, the generator $G_\theta(z)$ is a differentiable function that takes a vector of noisy random variables $z \sim p_z$ where p_z is a random distribution. The generator

[2]Also called a "transposed convolution" although the two are slightly different

also takes a set of input parameters that define the network weights θ. Similarly, the discriminator $D_\phi(x)$ is a differentiable function that takes as input a sample vector x and a set of parameters that define the network weights ϕ. $G_\theta : \mathbb{R}^n \to \mathbb{R}^m$ outputs a sample and $D_\phi : \mathbb{R}^m \to [0, 1]$ outputs a scalar representing the probability the sample came from G_θ. where n and m are model specific and need not even be constant due to both G_θ and D_ϕ being fully convolutional in nature. The goal of the network is to train D_ϕ to maximise the chance of it correctly calculating whether an input is real or generated. At the same time, G_θ is trained to maximise the chance it fools the discriminator. Traditionally this is done via an adversarial loss function, see Equation 2.1.

$$\min_{G_\theta} \max_{D_\phi} V(D_\phi, G_\theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D_\phi(x)] + \mathbb{E}_{z \sim p_z(x)}[\log(1 - D_\phi(G_\theta(z)))] \tag{2.1}$$

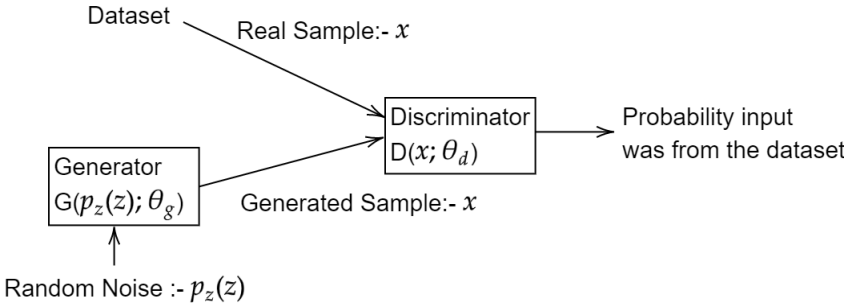

Figure 2.4: The network structure of the original Generative Adversarial Network with binary classification. The discriminator takes as input either the generated samples or samples from the dataset the generator is attempting to emulate.

Training GANs is a classic example of a min-max game, and the algorithm is as follows:

Algorithm 1 GAN training

for Number of training iterations **do**
  for K steps **do**
    Generate m random noise vectors
    Retrieve m samples from the real data
    Update the discriminators' parameters using gradient descent
  end for
  Generate m random noise vectors
  Update the generators' parameters using gradient descent
end for

Working with GANs is not only a very efficient way of solving many problems, but it also has the advantage that it ends up solving two problems at once. It produces both a network that can generate believable examples of something and a network that can determine something's validity.

2.1.4 Loss functions

Loss functions are used to optimise neural networks. They are vital in the training process since they indicate how close a neural network's performance is compared to its ideal. Choosing the correct loss function is vital in the correct execution and effective training of a network.

Mean Absolute Error

This loss function (also referred to as L1 loss) compares the difference in the input values and the values that would be expected in the desired output.

$$\text{L1Loss}(\mathbf{generated}, \mathbf{target}) = \frac{1}{n}\sum_{i=1}^{n} |\mathbf{generated}_i - \mathbf{target}_i| \tag{2.2}$$

Mean Square Error

This loss function (also referred to as L2 loss) is similar to mean absolute error. However, it squares the difference penalising mistakes in the prediction more.

$$\text{L2Loss}(\mathbf{generated}, \mathbf{target}) = \frac{1}{n}\sum_{i=1}^{n} |\mathbf{generated}_i - \mathbf{target}_i|^2 \tag{2.3}$$

2.2 Artifacts in Audio

For any kind of generative modelling, it is essential to look at the introduction of unwanted artifacts. An artifact is a property of the output. A well-known example of unwanted artifacts in image processing is the checkerboard pattern. It has been well studied that overlapping deconvolution layers cause this pattern. Essentially, some points in the output have influence from more of the previous points than others.

Logically, these issues should not be restricted to the two-dimensional domain, and a paper published earlier this year [28] explores artifacting in audio data. This paper explores two main types of unwanted audio artifacting, filtering artifacts and tonal artifacts.

Tonal artifacts

A tonal artifact is an unwanted additional sound with constant frequency (a tone). These can be introduced by using overlapping deconvolution layers for a similar reason to the 2D checkerboard example. The model should be able to learn how to correct these issues. However, because the model is fully convolutional and the signal may not have a length equal to multiple stride, there will likely not be a consistent degree of overlap at each stage. Even when there is constant overlap throughout the signal, there is not at the sides of the signal. This is called boundary artifacting.

To avoid these tonal artifacts being added due to overlapping, a one-dimensional version of the sub-pixel convolution network, as described by Wenzhe Shi in a recent paper [11], can

be used. This sub-pixel convolution uses a deep convolutional neural network to increase the number of channels the model is currently computing on by the upsample rate. Then the sub-pixel shuffle is used to reorder these into the feature space. This does mean that only a whole number upsample is possible but a recent work [26] shows an alternative dimension shuffle layer can be used to remove this condition. Figure 2.5 illustrates how sub-pixel convolutions remove overlapping and the need for padding.

Other ways tonal artifacts can be introduced include poor weight initialisation or loss functions. This would be a particular problem for a model that uses convolution filters since they are used repeatedly in series. Poor choices for initialised weights can introduce unwanted frequencies. Loss functions can cause high-frequency tonal artifacts in the gradients generated by backpropagation. It has been observed that it is an issue particularly in audio-based models, and it is thought to be caused by higher frequency components being present in audio. It is also a notable problem in adversarial learning, and paper [18] finds that these loss based tonal artifacts occur at predictable phases, making it easy for an adversary to discern fake samples. The paper explores a solution to this via altering the phase of the discriminator's kernels. This area has not seen much academic exploration, and it would be beyond the scope of this report to explore this phenomenon in any great detail.

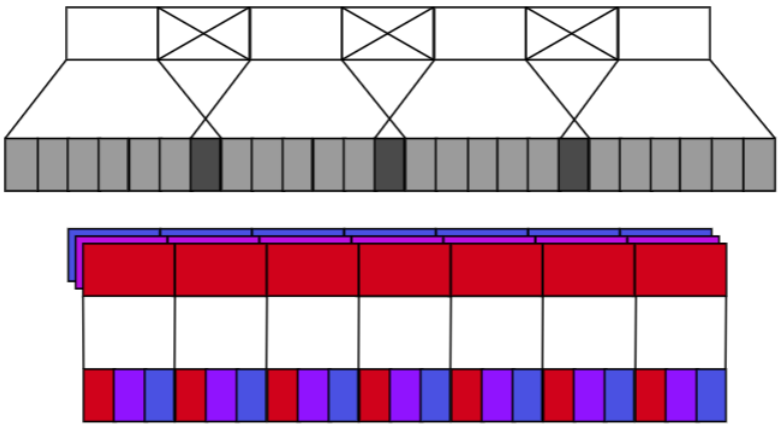

Figure 2.5: An illustration of how a deconvolution (top) and sub-pixel convolution (bottom) work for upsampling by factor 3. The deconvolution has stride 2 and upsamples each pixel by 6.

Filtering artifacts

In contrast to tonal artifacts, filtering artifacts can be defined as the unwanted absence of a sound with constant frequency. These can be introduced by using filters that are better at detecting certain frequencies than others or by a poor choice of the activation function. They can also be introduced by spectral replicas.

Spectral replicas

Since this project deals with signals over a discretised time interval, the signals it produces may contain spectral replicas. Spectral replicas are additional frequencies added that should not be present in the discretised version. To see why this happens when recording

audio, suppose a signal is sampled with a frequency of 2 Hz and a signal P with frequency 4 Hz is present. The effect of signal P will be observed every $\frac{1}{2}$ a second and will end up introducing a signal of frequency 2 Hz into the recording, which was not actually present. For the recording scenario, a low pass filter can be added to remove signal frequencies that are unable to be recorded correctly. However, when reconstructing a signal, care should be taken so as not to reintroduce frequencies that should not be present, particularly at factors of 2 higher than the original signal frequency (Nyquist frequency). These look like reflections of the lower signals on a spectrogram, which is where they get the name spectral replicas from. These may cause problems in this project and can also reflect other artifacts already introduced.

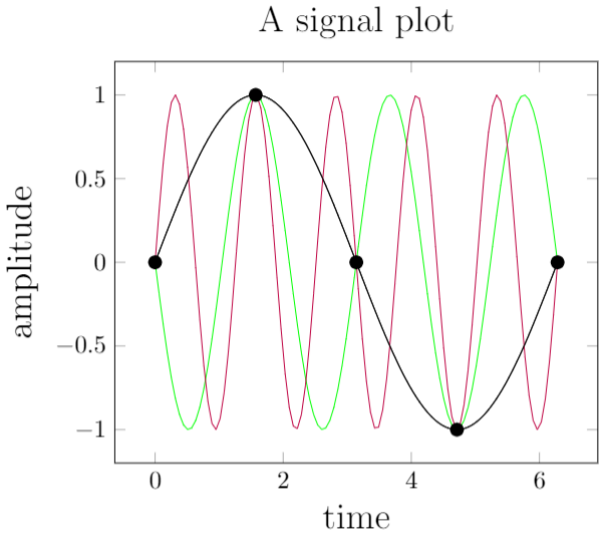

Figure 2.6: This graph demonstrates artifacting. If the three points in black were sampled, then if either the red or the green signal is present, they will incorrectly be interpreted as the black signal.

2.3 Dataset

This project uses CSTR VCTK corpus [30] as the primary dataset. This dataset contains speech from 110 different English speakers, each of whom read about 400 sentences from various newspapers. These sentences were chosen by an algorithm designed to maximise the phonetic coverage. All the samples in this dataset were recorded with the same microphones. The microphone has a sampling frequency of 96 kHz and a bit depth of 24 bits. All of the stored recordings were converted into having a 16-bit depth and were downsampled to a 48 kHz sampling frequency.

The VCTK corpus is a good database to use because this project aims to explore ways to improve sound signals, specifically concerning communications. This is because, in communications, data transmitted is often vocal. Making the data set specific to one type of audio will highlight the potential of this methodology since there will be more features to pick up on. This dataset is commonly used in other audio super-resolution works [22, 15] so it will be simple to benchmark the algorithm against existing baselines

for evaluation.

This dataset is licensed under a Creative Commons License: Attribution 4.0 International[3] and allows anyone to "produce, reproduce, and Share Adapted Material" provided the authors are attributed. This project produces adapted material, so it is permitted to use the dataset.

2.3.1 Metrics for evaluation

Signal-to-Noise Ratio

Perhaps the most common method of evaluating audio quality is to measure its signal-to-noise ratio. The signal-to-noise ratio function compares a generated signal to the objective truth signal. It then calculates the proportion of the generated signal that is useful information (in Decibels). In this project, the high-resolution audio will be considered the objective truth, and it will be compared to generated samples of the evaluated experiments.

$$\mathrm{SNR}(x_{\text{generated}}, x_{\text{actual}}) = 10 \log_{10} \frac{||x_{\text{actual}}||_2^2}{||x_{\text{generated}} - x_{\text{actual}}||_2^2} \tag{2.4}$$

A potential problem with using the original signal as the objective truth in SNR calculations is that these original signals will have at least some noise. This means that a better quality audio signal than the original may be produced, but it would produce a worse SNR than a slightly noisy signal.

Log-Spectral Distance

The Log-Spectral Distance (LSD) was a metric analysed in the 1970s by Gray and Markel [2] and was regarded as a way to give a better indication of the overall quality of a signal since this was not always reflected well in SNR. It instead computes how much distortion is present in the spectra of a signal. Again, this metric needs a "true" signal for comparison.

$$\mathrm{LSD}(x_{\text{generated}}, x_{\text{actual}}) = \frac{1}{W} \sum_{w=1}^{W} \sqrt{\frac{1}{K} \sum_{k=1}^{n} \Big(\log_{10} \frac{|x_{\text{generated}}(w,k)|^2}{|x_{\text{actual}}(w,k)|^2} \Big)^2} \tag{2.5}$$

Where $w \in W$ represents the time windows and $k \in K$ represent the frequency bins. $x_{\text{generated}}$ represents the generated signal and x_{actual} represents the expected output.[4]

As suggested in Gray and Markel's paper, the LSD metric is the best reference point for comparing speech processing methods. They argue its the most appropriate metric since it:

- "can be physically interpreted": As how close the frequencies are to the actual signal at a given time.

[3] `https://datashare.ed.ac.uk/bitstream/handle/10283/3443/license_text?sequence=3&isAllowed=y`

[4] This project uses a log of base 10 for its LSD measure similarly to some papers [27], although others use a natural log [15]

- is "analytically tractable": Can be computed with the analytic function cited in Equation 2.5.
- can be "easily and efficiently computed (using the cepstral measure)": Fourier methods can be used to calculate the measure. Specifically, a *cepstrum* can be used[5].
- is "relatable to several other widely used measures of distance": Explored in their paper.

MOS

The results will also be evaluated using a mean opinion score (MOS). In other words, a group of people will be asked to rate the overall quality of the sound produced by the algorithm. This is done because it is a more subjective metric, but it is also important that the signal produced sounds good, something this metric measures. In this project, people will be asked to rate the quality, and then the arithmetic mean of their answers will be taken as the overall score. The tests will be kept anonymous, and the ethics board has approved the methodology.

This metric is used because the signals produced should be of a good 'quality' and not just a signal that is close to the original. For example, there may be a model that scores well on the LSD and SNR metrics but produces audio that sounds clunky and robotic.

2.4 Requirements Analysis

After reading up on relevant concepts and architecture types in the field of generative modelling and machine learning, the criteria given in the project proposal (Appendix B) have been revised and evaluated both on how important they are to the project and how hard they will be to complete. This summary of requirements can be seen in Table 2.1 where high priority requirements are needed for a functional project, and medium priority requirements are needed for a successful project. The low priority requirements are not required for a successful project but rather potential extensions to the core.

After a criterion is met, in this document, it has been clearly marked as complete.

2.5 Tools Used

The primary aim of this project involves using deep learning. While implementing this from scratch would further the writers' understanding of machine learning's underlying concepts, it would take a long time and would not be as efficient or as easy to understand. Python is the language of choice for most machine learning projects, with its clear syntax making it easy to understand. Python has two main libraries used for deep learning, PyTorch and TensorFlow. This project will use PyTorch as its regarded as easier to learn using similar syntax since its written in Python, whereas TensorFlow is written in C++ and CUDA. It also gives more flexibility in network design. Other tools have been used

[5]The cepstrum is the inverse Fourier transform of the logarithm of the generated signal

Requirement	Priority	Difficulty
Process the dataset	High	Low
Output the upsampled signals into an appropriate format	High	Low
Implement the evaluation functions	High	Medium
Implement and evaluate the linear upsampling methods	Medium	Low
Adapt the EDSR model for use on audio data	Medium	Medium
Implement the AudioUNet model	High	Medium
Implement an altered version of the AudioUNet based on a GAN architecture	High	High
All models perform better on both SNR and LSD measurements than the Cubic Spline	Medium	Medium
All models achieves greater MOS than Cubic Spline	Medium	High
Adapt the EDSR network to incorporate Meta-learning for arbitrary scale upscaling	Low	High
Incorporate bit rate upsampling to the model	Low	Medium

Table 2.1: Summary of this projects high-level success criteria

for signal processing and recording the training process, and Table 2.2 summarises the tools used throughout this project along with their purpose.

2.6 Software engineering techniques

Throughout this project, an iterative development model was adopted. This meant that planning, design, implementation, testing and evaluation were done in series every cycle of development. This makes it easy to adapt the models after each iteration leading to a more polished final product, and is one of the leading Agile techniques used in industry.

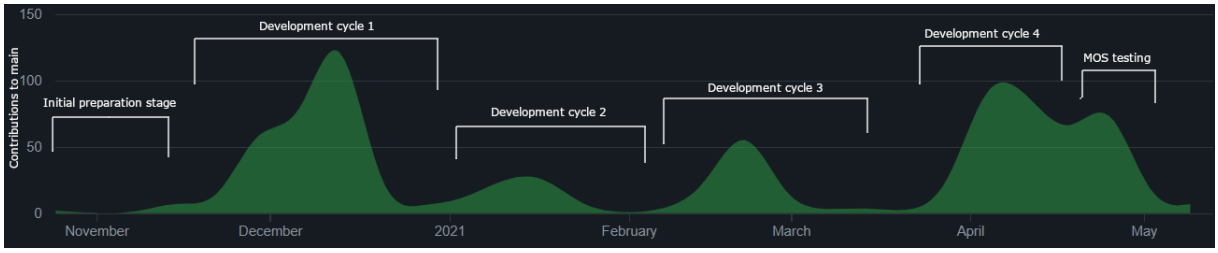

Figure 2.7: A summary of when commits were made to main. This shows the iterative nature that the project took. The development can bee seen by peaks in the commits and the testing/evaluation took place in the valleys.

2.6.1 Development Environment

Most of the development for this project was done in the PyCharm integrated development environment. This was mainly chosen because it has high support for Git version control and has excellent debugging tools. The GPUs used were in this project were provided by the Computational Biology Group, which contains both Nvidia Titan X GPUs and

Libraries & Programs	Primary Use
PyTorch	Deep learning library
Matplotlib	Plotting spectograms of data and graphs of results
NumPy	Efficient mathematical computation library
SciPy	Used for signal processing
Soundfile	Processing sound files to and from interpretable Python structures
tqdm	Viewing the progress of training
Weights and Biases [24]	Logging information such as losses and training progress remotely
Git	Version control
GitHub	Backup the project

Table 2.2: Libraries and Programs used for project development

Nvidia Titan Xp GPUs, for training the models. These were used to speed up training drastically.

2.6.2 Backup Strategy

Several measures were taken to ensure all elements of the project are kept safe and reduce the risk of losing any progress. Git commit was used for version control of the code, and it was pushed to GitHub for back up. A recent version of the repository was also pulled onto the Filer disc space on the Cambridge Computer Lab machines at any given point. These are resistant to failure. This report was backed up using Overleaf, which has version control; a recent version was also regularly saved to the writers hard disk.

2.7 Personal Starting Point

I have studied machine learning and AI as part of my degree. I otherwise had no experience in this area before I began this project. I had also never studied any signal processing beyond the high school level. I have also attended the lectures for “Machine Learning and Bayesian Inference” and “Computer Vision”.

To learn more, I worked through “Generative Adversarial Networks with Python” by Jason Brownlee [21] and the tutorial produced by Chuan Tan for the MetaSRGan [31], the latter of which was beneficial for learning how PyTorch deals with prepossessing. I have also read through lots of papers, a fraction of which are referenced throughout this document. Additionally, I have learnt PyTorch and TensorFlow, which are popular machine learning packages in Python.

Chapter 3

Implementation

This chapter takes you through the development of the project to its current form. It details how I implemented three different deep learning models to perform audio super-resolution, AudioEDSR, AudioUNet, and AudioUNetGAN. It also gives an overview of the data-processing required. The next chapter will compare these models to each other.

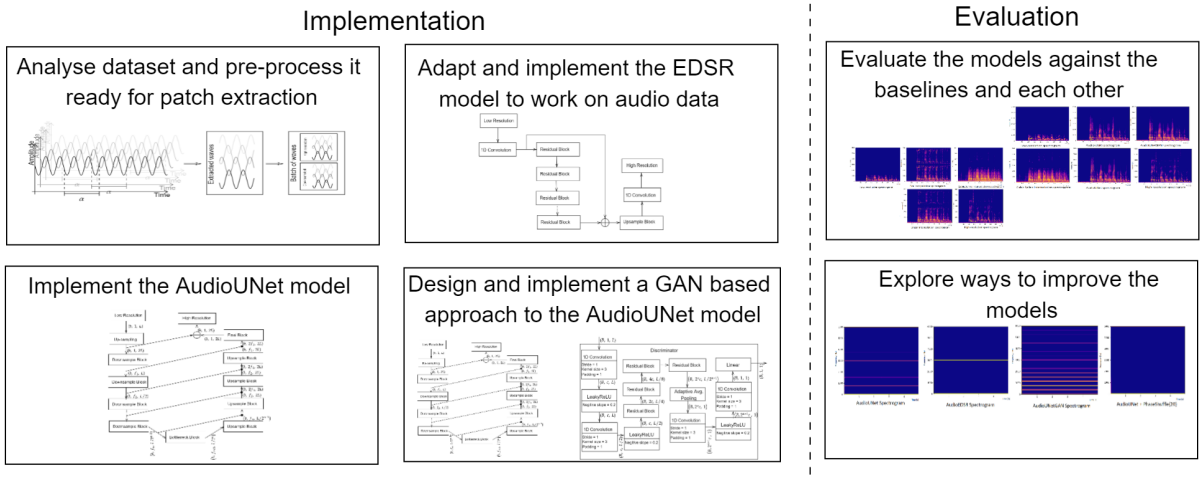

Figure 3.1: A summary of the work carried out in this project. Three different deep learning models were implemented, the dataset was processed and results were analysed and the models compared.

3.1 AudioEDSR

As a popular type of Deep Convolutional Neural Network, the Enhanced Deep Super-Resolution (EDSR) network was developed in mid-2017 [16] and was designed for image super-resolution. While many modern models tend to outperform the EDSR model,

it stood as a major breakthrough in the residual super-resolution field. Its structures effective use of deep residual blocks for extracting features is not restricted to the image domain, and I have adapted the design of the EDSR and implemented a version of this model for application to audio samples. The adapted network is referred to as AudioEDSR throughout this report, and Figure 3.2 shows its structure.

The AudioEDSR network is a form of *post-upsampling* network. This means that all feature detection is done on the input signal, and then the signal is upsampled. Performing feature detection on the smaller input size is beneficial as it reduces the size of the model and, as a consequence, decreases the time it takes to train and the space it takes up in memory. It is also less prone to noise since wide architectures can introduce this. A crucial downside discovered through the development of this model is that it suffers from distinct tonal and bordering artifacting.

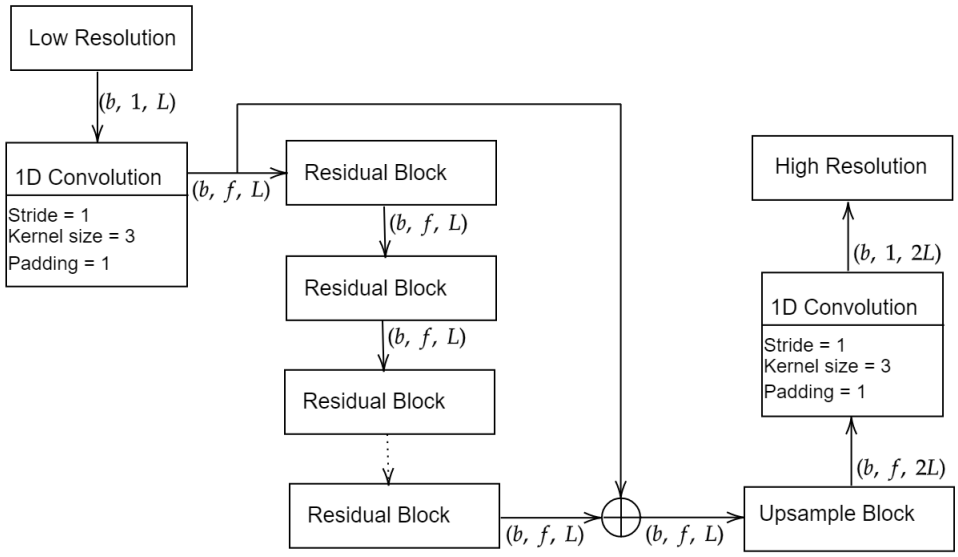

Figure 3.2: The structure of the AudioEDSR network. The arrows show how the network layers are connected and are each labelled with the shape of the tensor they carry. The residual blocks all contain residuals in them, and the number n of these blocks is a hyperparameter. The circle with a plus in it represents element-wise addition. b is the batch size of an iteration, L is the length of the signal, and f is another hyperparameter representing the number of filters extracted. This model is for a 2x upsampling problem. For the 2^q upsampling problem, the upsampling block is replaced by q upsampling blocks in sequence.

The general idea behind the AudioEDSR model's architecture is that a sequence of residual blocks extracts the features, and then the signal is upsampled based on these features.

3.1.1 Residual Block

Residual blocks extract the features of the signal so that they can feed the most prominent features into their upsampling block. A crucial feature of these blocks are the 'skip' connections. Skip connections help enhance the learning of each block since they can take

into account information about blocks in layers not immediately preceding them, so they can learn things such as the rate of change of features to make the model better. These connections also have the desirable property that, in deep neural networks, they help to mitigate the vanishing gradient problem [19].

In order to perform this feature extraction, the input is passed through a convolutional layer which, as discussed, is a fast and efficient feature extractor. Then a ReLU activation is applied on the blocks to assist learning. Next, the signal is passed through another convolutional layer and multiplied by a small residual scale factor. Skip connections are discussed in the paper and have been known to stabilise training. Finally, each residual block has a *skip* connection. This means that the features extracted from the last block are added to the features extracted in the current one. This idea was first shown to be effective for deep learning in the ResNet paper [6], and the results are that the features extracted by each block are less likely to be forgotten in the forward propagation process.

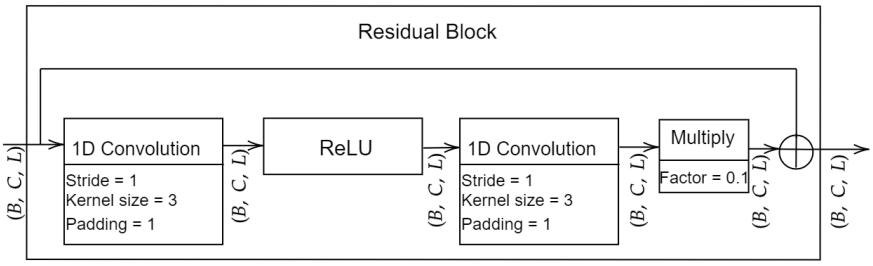

Figure 3.3: The stucture of the residual block used in the AudioEDSR framework. The signal is then multiplied by a hyperparameter in order to improve training. (B, C, L) represents the abstract shape of a tensor input to the block.

3.1.2 Upsample Block

The upsample block doubles the input length. To achieve this, first the signal is passed through a convolutional layer which is used to expand the channel space to the desired width and then a PixelShuffle layer is used to upsample its features. This layer is based on the efficient sub-pixel convolution, which utilises the additional channels to upsample the signal length. After upsampling, use a LeakyReLU layer to assist the training process.

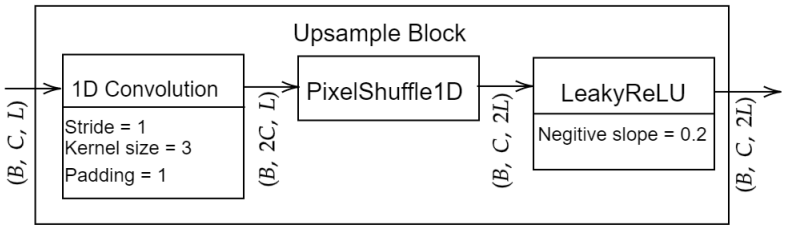

Figure 3.4: The structure of the upsampling block used in the AudioEDSR framework. The Convolution block expands the channels, and the PixelShuffle uses these to upsample the signal. (B, C, L) represents the abstract shape of a tensor input to the block.

3.1.3 Training

This model is trained to minimise L2 loss (Equation 2.3). See section 4.3.1 for an explanation of why this metric was chosen.

Algorithm 2 AudioEDSR training

Use hyper-parameters $\alpha = 0.0001, \beta_1 = 0.9, \beta_2 = 0.999$
for K steps **do**
  Sample high resolution data $x = \tilde{w}(t; S)$ from current batch
  Generate $l = \tilde{w}(t; \hat{S})$ from $x = \tilde{w}(t; S)$
  $\tilde{x} \leftarrow G_\theta(l)$
  $L \leftarrow \frac{1}{n} \sum_{i=1}^{n} |\tilde{x} - x|^2$
  $\theta \leftarrow \text{Adam}(\nabla_\phi \frac{1}{m} \sum_{i=1}^{m} L, \theta, \alpha, \beta_1, \beta_2)$
end for

Criterion met: *Adapt the EDSR model for use on audio data*

3.2 AudioUNet

Another type of deep residual network was developed in 2015 and was named U-NET after the model's shape. The U-NET was originally developed for use in biomedical image segmentation [8]. However, in 2017 Volodymyr Kuleshov *et al* [15] adapted this model's structure for use in an audio super-resolution task. This is probably the most state-of-the-art model to look at since it is still the one to beat, and there have been few improvements to its structure in the past four years. Although the authors have worked on subsequent models, they are not fully feedforward, so they lose some training benefits. I have implemented the model introduced by Kuleshov[1], and refer to it as AudioUNet in this report.

The AudioUNet structure works on blocks that downsample the input condensing the information further and further. At the lowest levels, the information stored in each block represents some abstract feature of the model. The signal is then upsampled using a 1D subpixel upsample layer[2] and append the output of the result from the corresponding downsample block[3]. This is achieved via skip connections. Skip connections have been shown to drastically improve training when the input distribution is highly correlated to the output [19]. This is because stacking gives the upsample block more context of the detected features and should be present at these levels.

In contrast to the AudioEDSR, this architecture is an example of a *pre-upsampling* network. While this does mean the model is larger in terms of storage, it only has to

[1] Correcting mistake in the papers explanation of the bottleneck structure. When contacted, the authors confirmed the mistake

[2] adapted from [12]

[3] cropped to be the same size as the upsample layer if needed

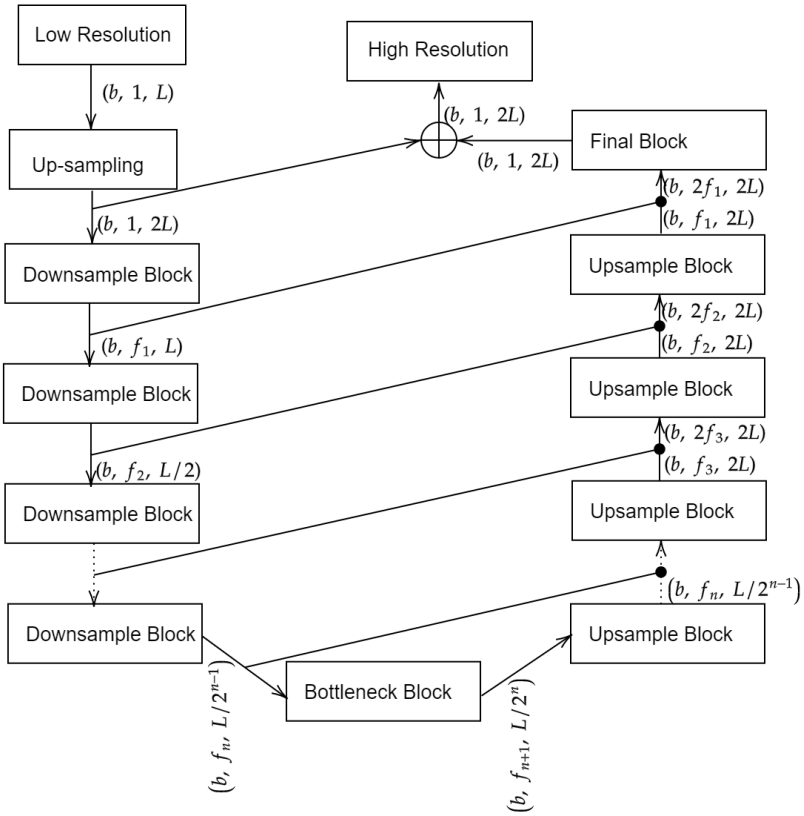

Figure 3.5: The structure of the AudioUNet. Features f_i are hyper-parameters, and the dark circles are stack connections that combine the features. b is the batch size of an iteration, L is the length of the signal, n is the number of residual blocks.

learn how to improve upon the splined input, and the network does not need to learn how to upsample from scratch. This helps reduce the impact of many of the upsampling artifacts discussed in section 2.2 and will potentially produce a more realistic sounding output.

3.2.1 Downsample Block

The downsample block decreases the length of the input signal by passing it through a convolutional block. Then the signal is passed through a Dropout block with a chance of ignoring each weight of $\frac{1}{2}$. This helps reduce the chance of overfitting the data.

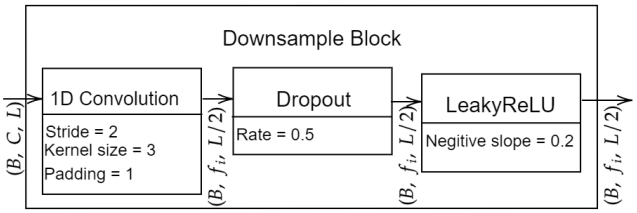

Figure 3.6: The structure of the downsample block used in the AudioUNet model. Typically this block doubles the input channel dimension and halves the input signal length. (B, C, L) represents the abstract shape of a tensor input to the block.

The output of a downsample block is also cropped (allowing for arbitrary scale upsampling) and fed into the corresponding upsample block, giving them a greater idea of the features extracted at this level. This is the main benefit of the AudioUNet structure.

3.2.2 Upsample Block

The upsample block increases the number of channels of the input by passing it through a convolutional block. Similarly to the downsample block, the signal is passed through a Dropout block with a chance of ignoring each weight of $\frac{1}{2}$. This helps reduce the chance of overfitting the data. At the end of the block, a subpixel upsampling layer is used. This reduces overlap based tonal artifacting, although tonal artifacts can still be introduced.

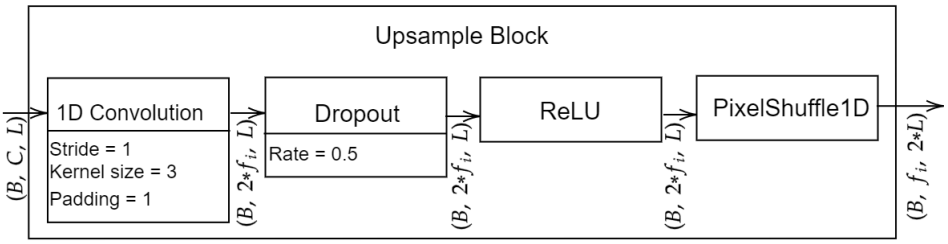

Figure 3.7: The structure of the upsample block used in the AudioUNet model. Typically this halves the input channel dimension and doubles the input signal length. (B, C, L) represents the abstract shape of a tensor input to the block.

3.2.3 Training

This model is trained to minimise L1 loss. See section 4.3.1 for an explanation of why this metric was chosen. (Equation 2.2)

Algorithm 3 AudioUNet training

Use hyper-parameters $\alpha = 0.0001, \beta_1 = 0.9, \beta_2 = 0.999$
for K steps **do**
  Sample high resolution data $x = w_D(t; S)$ from current batch
  Generate $l = w_D(t; \hat{S})$ from $x = w_D(t; S)$
  $\tilde{x} \leftarrow G_\theta(l)$
  $L \leftarrow \frac{1}{n} \sum_{i=1}^{n} |\tilde{x} - x|$
  $\theta \leftarrow \text{Adam}(\nabla_\phi \frac{1}{m} \sum_{i=1}^{m} L, \theta, \alpha, \beta_1, \beta_2)$
end for

Criterion met: *Implement the AudioUNet model*

3.3 AudioUNetGAN

Section 2.1.3 introduces the basic adversarial GAN structure. It consists of a generative model that learns a distribution of outputs and a discriminator model that works out if

its example is believable. The SRGAN [9] was the first to incorporate the idea of using a GAN in the super-resolution field and used a slightly different approach. This project aims to develop a generator that is conditional on the low-resolution input data. For this, it is no longer necessary to learn a distribution of outputs if only one deterministic output for each input is required. This means an input involving a random latent space is no longer needed. The model could be changed to learn a distribution of mappings for each input. It could be changed to add a small latent space, but this is not necessary for a functional model and would take longer to train for arguably limited benefit. This will be left as a potential route to explore in future development.

I have designed and implemented a GAN inspired structure for performing audio super-resolution. The GAN structure was chosen because there have been many examples of its use to improve various generative models over the past few years. In using a GAN I hope to show that it is possible to improve the quality of the signals generated by using a discriminator to judge their quality. This discriminator would effectively be ranking how realistic the signals produced are. The mean opinion score will evaluate this realism aspect. This structure will be built using the AudioUNet at the generative model and will be called the AudioUNetGAN in this report.

The AudioUNetGAN will be a combination of the SRGAN and an improved GAN model called a Wasserstein-GAN (WGAN) [13]. The WGAN was designed in early 2017 and was a new approach to helping assisted generative modelling. This new model was designed to fix two main problems with the traditional GAN:

- Updating two models simultaneously does not guarantee convergence and a vanishing gradient of the loss function is common. Finding the right hyper-parameters to train a model into training equilibrium is time-consuming.
- Traditional GANs are prone to mode collapse. This is when the generator learns how to trick the discriminator with a small range of generations and only outputs those regardless of input.

Fixing these two problems is ideal for any model, so the AudioUNetGAN will make use of it.

The type of WGAN the AudioUNetGAN will use is called WGAN - gradient penalty (WGAN-GP) [14]. The other main way of implementing a WGAN uses weight clipping which the WGAN-GP argues can be difficult hyper-parameters to get right, leaving the clipping method prone to some of the gradient problems present in the original GAN structure. WGAN-GP involves a slight modification to both the training method and the discriminator model, and it uses a Wasserstein-1 distance measure. Using a distance measure has the advantage of scoring how believable the generated signals are and not only if they are believable, which the authors argue leads to a more stable learning process. The Wasserstein-1 distance measure, in particular, is used because it is thought to be continuous and differentiable at most points. This means the discriminator will need to have a linear output activation function.

3.3.1 Generator

Currently, the AudioUNet model is being used as the generator (Figure 3.5). This model is being used as the generative model over the AudioEDSR because the AudioUNet structure is identical to an existing model that has proven to work. Whereas the AudioEDSR is an adaptation never tried on the domain before.

3.3.2 Discriminator

The discriminator used is heavily inspired by the SRGAN model and is displayed in Figure 3.8. It consists of n residual blocks, for hyperparameter n. Each residual block consists of 1D convolutions so that the network can be fully convolutional and learn effectively. Throughout the discriminator, the SRGAN paper states that LeakyReLU works best, and this makes sense since they reduce the chance of vanishing gradients over normal ReLU.

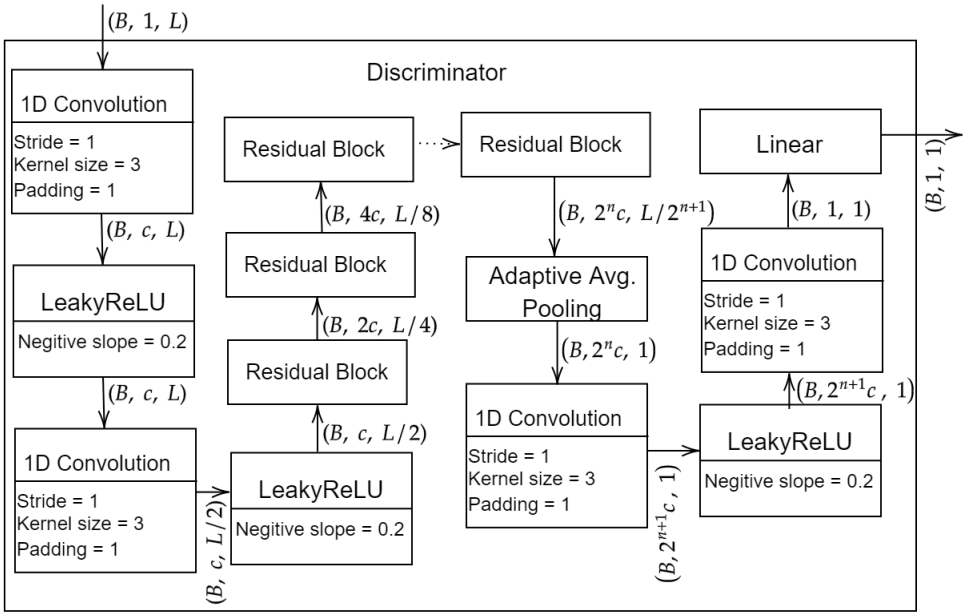

Figure 3.8: The architecture of the discriminator used in the AudioUNetGAN. It takes in a signal and outputs a score for each signal. $(B, 1, L)$ represents the abstract shape of a tensor input to the block.

Since the discriminator network no longer classifies the generated samples but instead scores how close they are, the literature on Wasserstein based GANs often rename the discriminator to the 'critic'. However, it serves mostly the same purpose in the network, and this report shall continue referring to this sub-network as the discriminator.

3.3.3 Training

The calculation of loss for this model is quite complicated. The primary loss is the adversarial loss (Equation 3.1), which D_ϕ tries to maximise and G_θ tries to minimise.

$$\mathbb{E}_{x \sim p_{\text{data}}(x)}[D_\phi(G_\theta(x_{\text{downsampled}})) - D_\phi(x)] \tag{3.1}$$

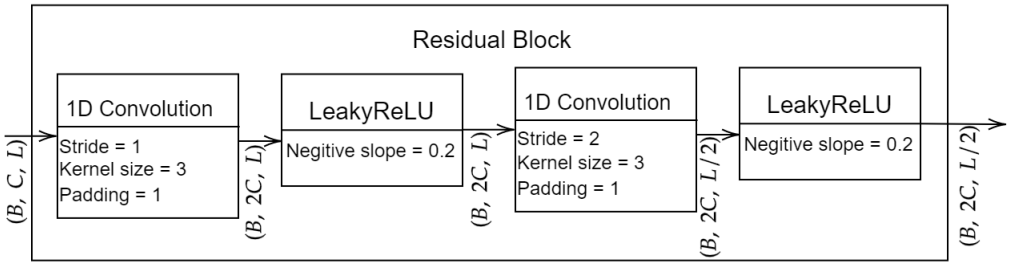

Figure 3.9: The architecture of the residual block used in the AudioUNetGAN's discriminator. It doubles the input channels and halves the input length. (B, C, L) represents the abstract shape of a tensor input to the block.

This adversarial loss is all that was used in the original WGAN. However, training a model this way requires an additional hyperparameter to be bound to how much the gradients can change after each update[4]. The WGAN-GP version of training, which the AudioUNetGAN is based on, replaces the need for clipping the gradients by altering the discriminator's loss function. The AudioUNetGAN's discriminator loss function is defined by Equation 3.2.

$$\mathbb{E}_{x\sim p_{\text{data}}(x)}[D_\phi(G_\theta(x_{\text{downsampled}})) - D_\phi(x)] + \lambda\mathbb{E}_{\hat{x}\sim p_{\text{data}}(\hat{x})}[(||\nabla_{\hat{x}} D_w(\hat{x})||_2 - 1)^2] \tag{3.2}$$

If the AudioUNetGAN were to follow the WGAN-GP model strictly, it would train the discriminator to maximise this altered function and the generator to minimise the original WGAN adversarial loss function. However, if L1 loss is added onto the original WGAN loss and adds a small coefficient as a hyperparameter to the adversarial loss so that they are on the same scale, training will be improved. This is because producing a good sounding output does not necessarily mean the output is close to the desired original. Adding the L1 loss will allow the loss to learn to produce both a believable sample of audio but also one that is close to the original signal.

$$\mathbb{E}_{x\sim p_{\text{data}}(x)}[\lambda D_\phi(G_\theta(x_{\text{downsampled}})) + \text{L1Loss}(G_\theta(x_{\text{downsampled}}), x] \tag{3.3}$$

So in AudioUNetGAN the generator will learn to minimise Equation 3.3, and the discriminator will learn to maximise Equation 3.2.

The training algorithm for the AudioUNetGAN is as follows:

[4]known as clipping the gradient

Algorithm 4 AudioUNetGAN training

Use hyper-parameters $\lambda = 10, n_{\text{critic}} = 5, \alpha = 0.0001, \alpha = 0.0001, \beta_1 = 0.9, \beta_2 = 0.999$
while θ not converged **do**
  for $t = 1, \ldots, n_{\text{critic}}$ **do**
    for $i = 1, \ldots, m$ **do**
      Sample high resolution data $x = \tilde{w}(t; S)$ from current batch
      Generate $l = \tilde{w}(t; \hat{S})$ from $x = \tilde{w}(t; S)$
      $\tilde{x} \leftarrow G_\theta(l)$
      $\hat{x} \leftarrow \epsilon x + (1 - \epsilon)\tilde{x}$
      $L^{(i)} \leftarrow D_\phi(\tilde{x}) - D_\phi + \lambda(||\nabla_{\hat{x}} D_\phi(\hat{x})||_2 - 1)$
    end for
    $\phi \leftarrow \text{Adam}(\nabla_\phi \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, \phi, \alpha, \beta_1, \beta_2)$
  end for
  Generate a batch of m low resolution variables $l = \tilde{w}(t; \hat{S})$ from $x = w_D(t; S)$
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} [-D_\phi(G_\theta(l))], \theta, \alpha, \beta_1, \beta_2)$
end while

Criterion met: *Implement an altered version of the AudioUNet based on a GAN architecture*

3.4 Pre-processing the Dataset

The VCTK dataset stores its audio samples in Free Lossless Audio Codec (FLAC) files. The *soundfile* Python package can be used to efficiently convert these files into an array of amplitudes in the range $[-1, 1]$ at any given time point. The models implemented throughout this project all make use of the ReLU activation function[5], which has been shown to be one of the most efficient for training. To make the most out of this function, the signals will be shifted to be in the range $[0, 2]$ before inputting it to each model and then simply shifted back to the appropriate range $[-1, 1]$ in post-processing. This avoids the model needing to learn any bias for the input and ensures the output is in an appropriate range.

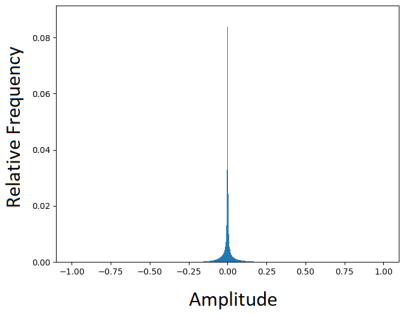

Figure 3.10: A graph showing the distribution of amplitudes from a sample the VCTK dataset

In the VCTK dataset there are $2,637,815,104$ individual measurement points. The mean amplitude is -1.53×10^{-6}, and the variance is 2.49×10^{-3}. This data, along with Figure 3.10 suggests that the data can be fit to a steep normal distribution which means there is no need for any data normalisation.

[5]ReLU forces negative values to zero

Another thing to note is that when downsampling a signal, artifacts may be introduced due to higher frequencies (discussed in section 2.2). These artifacts can actually act as a way of encoding information about which higher frequencies are present. So any model produced would be able to use this to its advantage to improve the accuracy of its predictions. However, downsampling in this way does raise an interesting question about the problem statement. When a traditional condenser microphone records audio, it passes the signal through a band-pass filter. This removes detected frequencies that are not in the range their digitised form can represent without artifacts. In order to stay true to the problem statement, the models in this project will upsample audio as if it has been recorded by a microphone and digitised to a lower sample rate. So when a signal is downsampled, it will be passed through an order eight low-pass Butterworth filter to remove the frequencies that the downsampled rate could not represent. This encoding of data via artifacting is something to consider for audio compression.

3.4.1 Patching

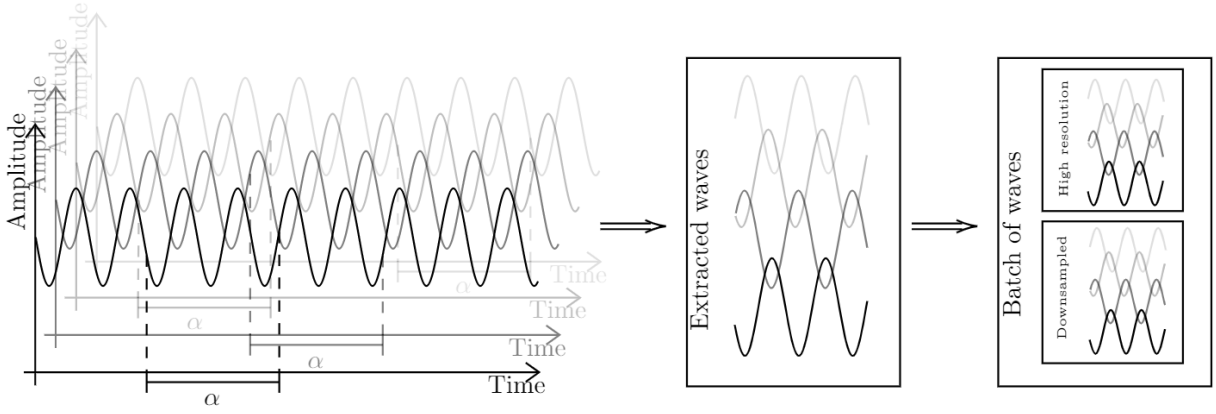

Figure 3.11: This figure demonstrates the process of extracting a batch with patch length α and batch size 4.

In order to extend the dataset, each signal can be split into patches of a fixed length of $6,000$ samples. Then a tensor can be formed containing 64 of these samples (this is called the batch size). The patches are selected randomly from within each signal, and each batch contains a selection of patches from a single file.

Typically extracting data in this way has two advantages; it reduces the amount of data required to be loaded into memory at any given time and, it can be used to increase the size of the dataset artificially. For this project, the foremost of these points is the main motivation for processing data in this way. The files in the VCTK dataset have an average number of samples of around $160,000$. This would mean loading an average file in its entirety would take up a large amount of memory. The latter is also good here, though, as it reduces the risk of overfitting.

The VCTK dataset consists of files with signals $\tilde{w}(t; 480,000)$ of variable lengths. For the processing of a signal $\tilde{w}(t; 480,000) : \{\frac{1}{480,000}, \frac{2}{480,000}, \ldots, \frac{480,000L}{480,000}\} \to \mathbb{R}$ of length

L, n patch's $\tilde{w}(t; 480,000) : \{\frac{x+1}{480,000}, \frac{x+2}{480,000}, \ldots, \frac{x+480,000\alpha}{480,000}\} \rightarrow \mathbb{R}$ are extracted for x's chosen uniformly from $U[0, L - \alpha]$. Next, the downsampled signal is computed for each of the patches e.g. for learning of upscale factor 2: $\tilde{w}(t; 240,000) : \{\frac{x+1}{240,000}, \frac{x+3}{240,000}, \ldots, \frac{x+240,000\alpha}{240,000}\} \rightarrow \mathbb{R}$. Finally a batch is formed from the n downsampled values signals along with the n signals sampled at the original quality. Figure 3.11 demonstrates this process.

Criterion met: *Process the dataset*

3.5 Navigating the Repository

As mentioned in section 2.7, I used the MetaSR tutorial [31] to learn about PyTorch and several image super-resolution techniques. The dynamic batch processing has been adapted and built on top of this tutorial. PyTorch also lacks an implementation of subpixel convolution, but there is a commonly used implementation of it found online [32].

There are many different cogs to make this project work. The code has been split into several directories based on their use. For example, the code for the architectures of the models are in the "Models" directory, and the code for training the models can be found in the 'Training_Functions' directory. On the top level of the directory, there are the files that are designed to be run from the command line and a detailed structure of the program directory can be seen below.

```
SoundSuperResolution/
├─Data/ ..... Pre-trained models, evaluation results
│  ├─Results/ ..... Example outputs, used in MOS testing
│  ├─TEST/ ..... Files in the test dataset
│  ├─TRAIN/ ..... Files in the train dataset
│  └─VAL/ ..... Files in the validation dataset
├─Models/ ..... Code for the deep-learning models
├─Processing/ ..... Code for processing the dataset
├─Training_Functions/ ..... Code used for during the training process
├─Utils/ ..... Code for dataset analysis and separation
├─Evaluate.py ..... Run-able file used for evaluating trained models
├─Predict.py ..... Run-able file used for evaluating non-ml functions
├─requirements.txt ..... List of project dependencies
├─TestDiscrim.py ..... Run-able file used for training the discriminator
└─Train.py ..... Run-able file used for training models
```

Chapter 4

Evaluation

This chapter gives a detailed look at the different deep-learning models final results along with the non-machine learning models. It also contains a detailed look at how artifacting effects the output of the models and explores some potential improvements.

4.1 Measuring the Criteria

As discussed in section 2.3.1 three different metrics of evaluation will be used: log spectral distortion (LSD), signal-to-noise ratio (SNR) and mean opinion score (MOS). In order to perform testing, training and evaluation fairly, the dataset is split into three parts, with testing containing 88% of the dataset, the training containing 6% and the validation set containing 6%. When evaluating the models for the LSD and SNR objectives, the metric will be run over all files in the evaluation set, and the mean and variance of the metric will be logged.

The subjective MOS metric will be evaluated slightly differently. Each file in the evaluation set will be split into the low-resolution file, the high-resolution file and the predicted file. They will then be anonymised, and a selection of people will be asked to score each file on how nice it sounds on a scale of one to five. The high and the low-resolution samples are there for control and will give context to the scores for the different models. Similarly to the LSD and SNR metrics, each file's mean and variance will also be recorded.

Criterion met: *Implement the evaluation functions*

The VCTK dataset contains data in the free lossless audio codec (.flac) format. This is an open-source audio compression format that uses a form of lossless compression. There is full read and write support for these files in the *soundfile* library in Python, so it is the logical format to process the data in and out of for the models in this project.

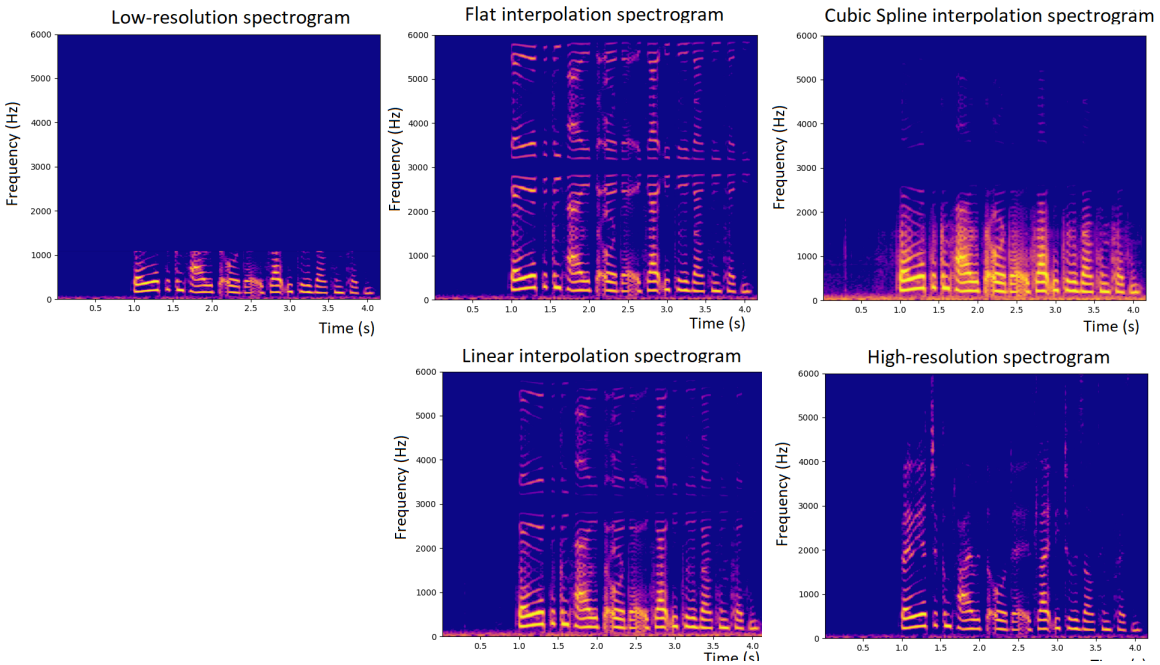

Figure 4.1: Spectrograms of some basic interpolation methods on the 4x upsampling problem.

Criterion met: *Output the upsampled signals into an appropriate format*

4.2 The Linear Methods and Visualising Results

The effects of the resolution enhancing methods that do not use any form of machine learning were seen in Figure 1.2. It is clear from this diagram that both the flat and linear methods are not ideal and comes nowhere close to re-constructing the signal visually. However, using this representation of a signal, it is not clear why the spline approach is also not that good. Spectrograms can be used for a more accurate visual analysis of what is happening in a signal.

Spectrograms have the time points of a signal along the x-axis and frequencies along the y-axis. Spectrograms also contain altering colours to show the amplitude of frequencies at any given time point. In this project, a scale from dark blue to yellow is used for spectrograms. Dark blue representing frequencies with no amplitude (i.e. not present), and yellow is used to represent those frequencies that are present and have high amplitude.

Figure 4.1 shows the spectrograms of the pre-machine learning interpolation methods. It is clear from inspecting these spectrograms that are missing frequencies in the spectral bands around $3,000$ Hz that should be present and suffering from filtering artifacts. What is also apparent on closer inspection is that these methods suffer from the creation of spectral replicas. These spectral replicas appear to be considerably more severe in both

the flat and linear methods than in the spline approach. However, the spline approach adds considerably more noise around the existing frequencies. In order to decide upon the best metric to use, the evaluation metrics will have to be consulted.

	Method		
	Flat	Linear	Spline
LSD scores			
2x Upsampling	$\mathbf{0.785 \pm 0.00271}$	1.01 ± 0.00225	1.89 ± 0.00303
3x Upsampling	$\mathbf{1.07 \pm 0.00565}$	1.50 ± 0.00650	2.95 ± 0.00775
4x Upsampling	$\mathbf{1.20 \pm 0.00684}$	1.78 ± 0.0112	3.64 ± 0.0109
SNR scores			
2x Upsampling	1.72 ± 2.32	$\mathbf{3.14 \pm 2.29}$	2.90 ± 2.38
3x Upsampling	-1.39 ± 1.53	$\mathbf{0.00422 \pm 1.78}$	-0.365 ± 1.99
4x Upsampling	-2.65 ± 0.806	$\mathbf{-1.43 \pm 1.05}$	-1.94 ± 1.24
MOS scores			
2x Upsampling	2.63 ± 0.368	3.00 ± 0.625	$\mathbf{3.80 \pm 1.07}$
3x Upsampling	1.76 ± 0.566	1.71 ± 0.596	$\mathbf{2.82 \pm 0.404}$
4x Upsampling	1.24 ± 0.191	1.29 ± 0.221	$\mathbf{1.94 \pm 0.434}$

Table 4.1: A table showing the LSD, SNR and MOS scores of the pre-interpolation machine learning approaches. Values for LSD and SNR are in dB, and the MOS score was out of five. The standard deviation is used as the bounds of error. The best performing metric for each scale is in bold.

Table 4.1 shows the results of the evaluation metrics. This table shows that the LSD from the flat interpolation is superior to both of the other methods. This makes logical sense since the flat interpolation method effectively stretches out existing time points so frequencies will be replicated. This even goes some way to explaining why spectral replicas occur.

The table of results also shows that linear interpolation produces amplitudes closer to the high resolution at each time point than the others. However, the spline approach is close and within the bounds of error. These results make it clear why multiple metrics are used since they can give conflicting results. When deciding upon the best model to use as input the pre-interpolation models, the most important metric, mean opinion score, was used.

The table shows that the spline method is the nicest to listen to, so it seems like the ideal method to create a machine learning model over to improve. Another reason to choose this model is that the spectral resultants cause less severe artifacts. These artifacts would be replicated in any upsampling layer, making it harder for any model to learn.

Criterion met: *Implement and evaluate the linear upsampling methods*

4.3 How the models were trained

4.3.1 The selection of key hyper-parameters

L1 vs L2 loss

Both the AudioEDSR and the AudioUNet network can be implemented as stand-alone networks. The two most popular loss functions for training super-resolution model are L1 and L2 loss (introduced in section 2.1.4). The AudioUNet paper [15] uses L2 loss in training. This section investigates whether or not that was the optimal choice.

It has been shown that L1 can be superior to L2 for image processing when looking at the commonly chosen evaluation functions [20]. The paper shows that because L2 loss has its property penalises outliers more, it can learn to train faster over data with minimal noise.

In order to find out which is loss function to use for the AudioEDSR and AudioUNet, respectively, an investigation was carried out over the 4x upsampling problem. The LSD and SNR results were compared for both models, and the results are shown in Table 4.2, and both of these models were trained over 32,100 batches which took approximately 7 hours for each model. The validation set was then used to evaluate the models, and the results show us that for AudioUNet, L1 loss looks like the best loss function to use since it scores more highly on both SNR and LSD. However, in AudioEDSR, the results show that SNR is better for L1 loss, but the LSD is better for L2 loss. The AudioEDSR will be trained with L2 loss since, as previously discussed, LSD is generally a better metric, and the SNR is within the bounds of error.

L1 vs L2 loss on AudioUNet		
	Loss function	
	L1	L2
SNR	$\mathbf{18.0 \pm 1.81}$	17.2 ± 1.66
LSD	$\mathbf{2.20 \pm 0.02}$	2.35 ± 0.02

L1 vs L2 loss on AudioEDSR		
	Loss function	
	L1	L2
SNR	$\mathbf{17.8 \pm 1.96}$	16.4 ± 1.66
LSD	1.76 ± 0.02	$\mathbf{1.60 \pm 0.02}$

Table 4.2: A comparison of L1 loss vs L2 loss on the 4x upsampling problem. The best results for each are in bold.

Complex generator loss

Another key hyperparameter is for the complex loss function for the generator in AudioUNetGAN. Training with them equal means the adversarial loss component overwhelmingly determines the loss since its losses start an order of magnitude higher. While L1 loss quickly becomes of order 10^{-3}. To compensate for this, a hyperparameter scale factor λ is applied to the adversarial loss. The optimal value for this hyperparameter will be chosen based on which value leads to the best LSD convergence. Figure 4.2 shows the results of these experiments, and it is clear from them then $\lambda = 0.001$ is optimal.

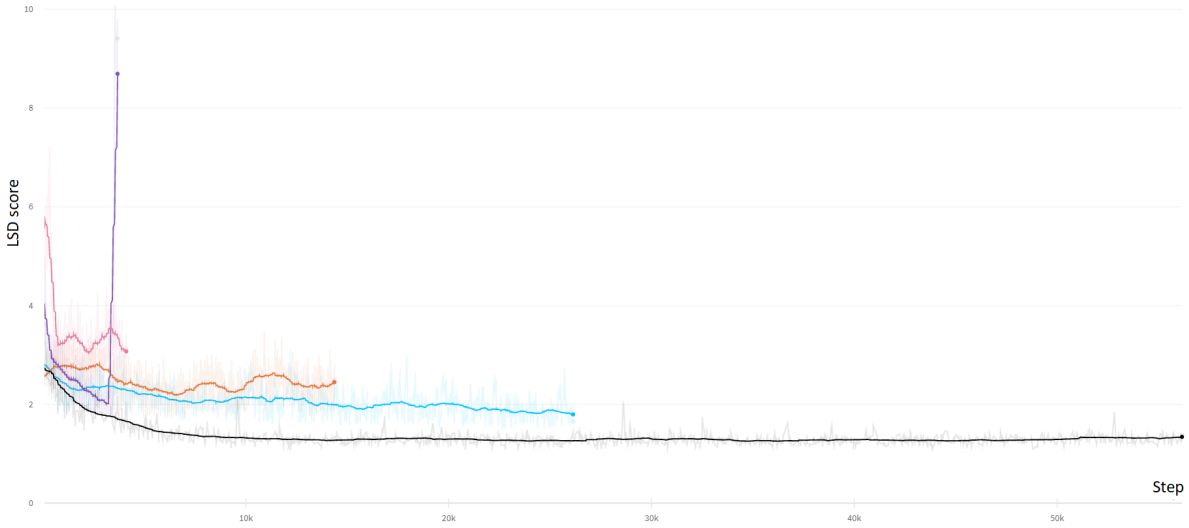

Figure 4.2: A comparison of LSD convergence for different λ's. Weights and Biases generated this graph. The pink line has $\lambda = 1$, the orange line has $\lambda = 0.1$, the pink line has $\lambda = 0.01$ and the black line has $\lambda = 0.001$. The purple line shows a model that failed when trained with L1 loss decreased by 0.001 instead of adversarial loss.

4.3.2 The Training Strategy

The signal generated by the models are of the type 'float32' and in the range of 0 to 2, meaning the signal has a bit-depth of 64. So bit-depth upsampling can trivially be performed by not cropping the output in the final signal.

Criterion met: *Incorporate bit rate upsampling to the model*

Training of AudioEDSR and AudioUNet

Apart from section 4.3.1 showing a different loss function is best suited to training the AudioEDSR and AudioUNet model, the training process was the same. They were both trained using the Adam optimiser with learning rate $\alpha = 10^{-4}$. Each implementation of the AudioUNet was trained with 4 to downsample blocks, and 4 upsample blocks, while the AudioEDSR implementations each had 32 residual blocks. They were trained over the VCTK dataset, and it took 400 epochs to complete the running with batches of size 64 and 321 batches per epoch. Titan X GPUs were used to train them, and it took approximately two days to complete training.

The learning rate was reduced by α by 20% after every 20 epochs during training. This kind of learning rate decay has been shown to improve many modern machine learning models since it reduces the chance that gradient descent will get trapped in a cycle and assists convergence to a minima [23]. After every epoch, the LSD and SNR scores were logged against whole files in the validation set to give an independent indication of training progress. After the conclusion of training, the model was run on the whole files of the test dataset, and the mean and standard deviation of their SNR and LSD scores were

	Method			
	Spline	AudioEDSR	AudioUNet	AudioUNetGAN
LSD scores				
2x Upsampling	1.89 ± 0.003	$\mathbf{1.31 \pm 0.009}$	1.58 ± 0.009	1.52 ± 0.005
3x Upsampling	2.95 ± 0.008	N/A	$\mathbf{2.00 \pm 0.013}$	2.04 ± 0.001
4x Upsampling	3.64 ± 0.011	$\mathbf{1.79 \pm 0.023}$	2.22 ± 0.019	2.11 ± 0.014
SNR scores				
2x Upsampling	2.90 ± 2.38	$\mathbf{23.9 \pm 6.74}$	22.1 ± 2.10	20.8 ± 1.53
3x Upsampling	-0.37 ± 1.99	N/A	$\mathbf{19.1 \pm 1.44}$	17.8 ± 1.44
4x Upsampling	-1.94 ± 1.24	17.6 ± 1.78	$\mathbf{18.3 \pm 1.93}$	16.9 ± 1.37
MOS scores				
2x Upsampling	3.80 ± 1.067	$\mathbf{4.06 \pm 0.809}$	3.82 ± 0.529	3.94 ± 0.559
3x Upsampling	2.82 ± 0.404	N/A	$\mathbf{3.47 \pm 0.390}$	3.18 ± 0.779
4x Upsampling	1.94 ± 0.434	2.24 ± 0.191	$\mathbf{2.76 \pm 0.691}$	2.06 ± 0.559

Table 4.3: A table showing the LSD, SNR and MOS scores of the Deep learning-based approaches. Values for LSD and SNR are in dB, and the MOS score was out of 5. The standard deviation is used as the bounds of error. The best performing metric for each scale is in bold.

recorded.

Training AudioUNetGAN

The AudioUNetGAN network was trained over an adversarial loss function combined with gradient penalty function for the discriminator and combined with L1 loss for the generator. For the generators loss function, the hyperparameter of $\lambda = 0.001$ was chosen, meaning both losses impacted training to an approximately equivalent degree. Again, the Adam optimiser was used to train both the discriminator and the generator. Both of these sub-networks were trained with learning rate $\alpha = 10^{-4}$. The discriminator was trained 5 times more often than the generator to make sure it could score the generator well. The AudioUNet based generator consisted of 4 downsample blocks, and 4 upsample blocks, while the discriminator consisted of 8 blocks. The AudioUNetGAN was trained over the VCTK dataset, and it took 400 epochs to complete the running with batches of size 64 and 321 batches per epoch. Titan X GPUs were used to train the dataset, and it took 4 days to train.

Similarly to the Non-GAN based models, after every epoch, the LSD and SNR scores were logged against whole files in the validation set to give an independent indication of training progress. After the conclusion of training, the model was run on the whole files of the test dataset, and the mean and standard deviation of their SNR and LSD scores were computed.

4.4 Analysis of results

This section discusses the results of the models and contains reference to the table of results (Table 4.3). A visualisation of the models performances is also displayed on Figure 4.3, but the results are visually similar.

The results show that all of AudioEDSR, AudioUNet and AudioUNetGAN outperform the spline method at all upsampling factors for both LSD and SNR by a clear margin. It also shows that the models outperform the spline on the MOS too. Although it is worth mentioning the spline signal does not make the signal sound too unpleasant at the 2x upsampling factor, the models still managed to perform bette quality audio not just a better reconstruction. This proves that deep learning models can be highly effective at this form of signal reconstruction. Surpassing this baseline is evidence of the success of this project.

Criterion met: *All models perform better on both SNR and LSD measurements than the Cubic Spline*

Criterion met: *All models achieves greater MOS than Cubic Spline*

Some of the results are surprising, though, especially when comparing the results from the three deep learning models. For example, the post-upsampling AudioEDSR model is the best performing model on all metrics in the 2x upsampling case. This is unexpected since all recent models that performing audio super-resolution have been based on the pre-upsampling framework. A benefit to using the AudioEDSR network is that it uses significantly fewer parameters to train, meaning it is more lightweight, so it takes up less space in memory and is faster to train. This difference can be seen in Table 4.4 where it is clear that the AudioEDSR is approximately 83% smaller than AudioUNet.

	Method	
	AudioEDSR	AudioUNet
2x	9,594,113	56,411,394
4x	9,692,673	56,411,394

Table 4.4: A table showing the number of parameters the AudioEDSR and AudioUNet models.

Despite its performance in the 2x upsampling case, the AudioEDSR falls short of the AudioUNet's MOS in the 4x upsampling problem. This is likely due to the tonal artifacting added by the subpixel shuffle layer, making it nearly harder for this model to train and perhaps even audible to some listeners. Although the AudioUNet also sufferers from this type of artifacting, its structure of skip connections makes it easier for the model to learn to lessen the effect. Notably, though, the AudioEDSR still outperforms the AudioUNet on the LSD metric and is not that far behind on the SNR front, meaning the lightweight nature may still make it a better option for real-world deployment.

Another surprise is that the GAN does not provide much improvement over the original

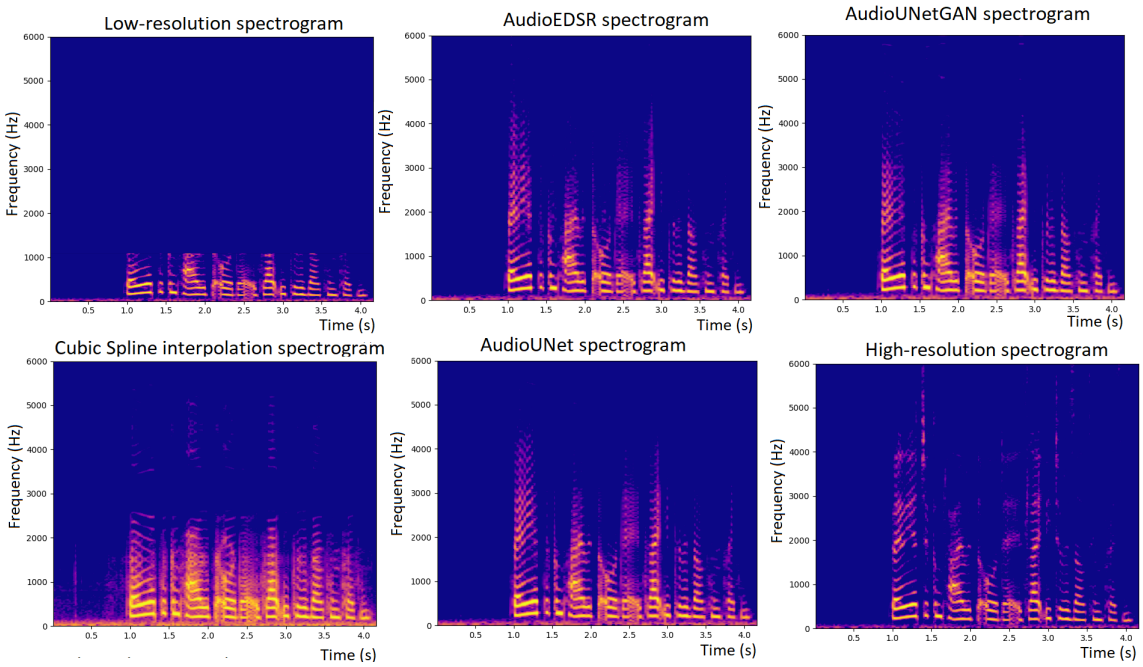

Figure 4.3: A visualisation the performance of the AudioEDSR, AudioUNet and AudioUNetGAN. compared to low-resolution input and the spline baseline.

purely L1 based training of the AudioUNet, performing slightly worse in almost all metrics. The one metric that beats the AudioUNet in is the MOS on the 2x upsampling problem. The main theory for this lack of improvement is that the systematic tonal artifacts produced by the AudioUNet, are predominately used in the discriminators scoring process. This is backed up by almost all of the AudioUNetGANs performance metrics getting further from the AudioUNets because the upsampling problem increases since the intensity of these tonal artifacts becomes larger. The next section explores this artifacting and explains why it is a problem when training a GAN.

4.4.1 Analysis of Artifacts

Minimising the number of artifacts the models produce is ideal since it will help maximise both the overall quality of the signal produced and how close it is to the original. Figure 4.4 shows that, on an empty input, there are several minor artifacts produced by the AudioUNet and one major artifact produced by the AudioEDSR for the 4x upsampling problem. A signal with no frequencies is used to analyse artifacts here since the spectrogram should be blank, so any signal generated by the models must have been erroneously interpolated.

Intuitively the artifacts are produced by the subpixel convolutions both because they are prone to introducing them [28] but also, regardless of scale trained on the AudioUNet produces $2^{\text{number subpixelshuffle blocks}}$ artifacts at fixed phases. The major EDSR artifacting is likely what is causing the low MOS score and if this artifacting can be reduced, it may even lead to a shift in research from these pre upsampling approaches to post-upsampling.

It is interesting to note that Figure 4.4 shows that the AudioUNetGAN contains the same

artifacts as the AudioUNet model. This furthers the assumption that the AudioUNetGAN was making slow progress due to its discriminator network scoring the generator whether these artifacts were present or not. The WGAN framework did little to help avoid this problem. The rest of this section will explore potential fixes and mitigations to the artifacting problem.

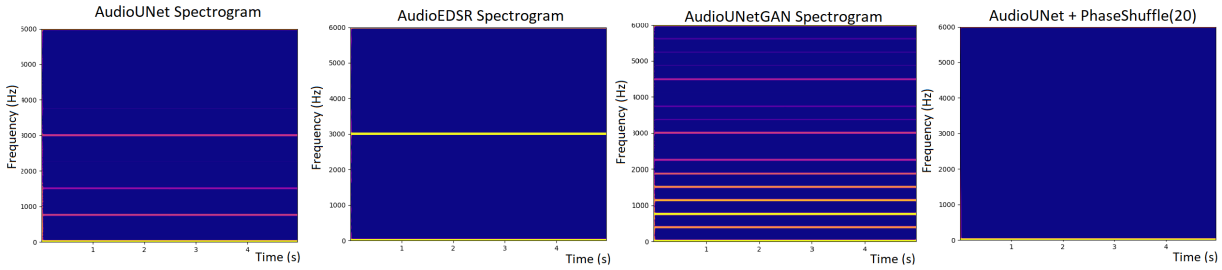

Figure 4.4: Spectrograms showing the tonal artifacts produced by the 4x upsampling layer of the three Deep learning models when the input is an empty signal. The image also shows how modifying the network with phase shuffles can reduce these artifacts.

4.4.2 Improving the results

There are many ways the results of this project could be improved in future development. The input could be learnt over a latent space, and outputs could be selected from this distribution, so artifacts are less likely. The effect of the tonal artifacts could be mitigated by ignoring them in the discriminator of the AudioUNetGAN. One way of doing this is by decomposing the signal into its frequency components and feeding these components into the discriminator as channels. The process of breaking down a signal in this way can be done with empirical mode decomposition and makes it much harder for the discriminator to learn.

Another way the effects of artifacting could be mitigated is by adding feature loss to the generators loss function. There are a few ways of performing feature loss. Traditionally, it involved comparing the feature maps of both the generated and the target signal when halfway through the discriminator. However, more recent approaches involve using a pre-trained autoencoder network to extract the signals key features. This approach has seen a lot of success when applied to GAN based super-resolution models, and several recent papers have used it to improve existing models [9, 22].

While it would be interesting to investigate the effects these changes would have on the AudioUNet and AudioUNetGAN, This report will instead explore how effective phase shuffle layers are at removing and mitigating tonal artifacts.

Phase Shuffle Layers

One potential way of reducing the effects of the tonal artifacts produced by the AudioUNet and AudioUNetGAN, is by using carefully placed phase shuffle blocks [18]. A phase shuffle block effectively shifts the feature map forwards or backwards by x where $x \sim \text{Uniform}[-n, n]$ for hyperparameter n. The shuffle does not change the dimensions of

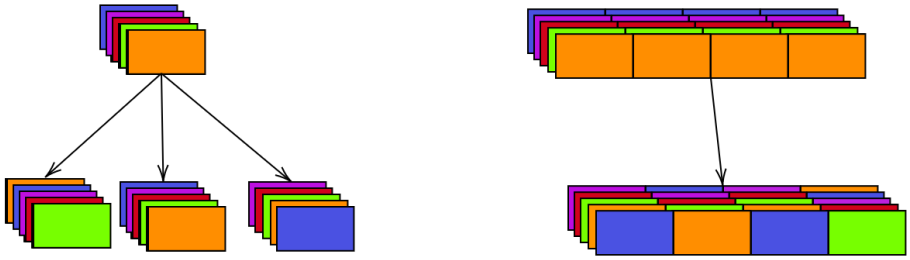

Figure 4.5: A figure showing the effect a phase shuffle block, with hyperparameter one, has on a signal. The feature map for each point in the signal is randomly assigned to one of the shuffled outputs (left). The shuffle factor is re-sampled for each point, and the figure also shows a possible outcome for the phase shuffle with a channel depth of five and signal length of four (right).

the feature map. It cycles any overflow features back to the start. Figure 4.5 shows the process in more detail.

The reason phase shuffle blocks can help to reduce these artifacts is that, when upsampling the channel space before a sub-pixel convolution, the neighbouring channels still have a lot of their feature detection properties in common with each other. This is what causes the artifacting. While shifting the problem to the spectral domain is not as problematic for image super-resolution, we have seen it is for audio super-resolution. The phase shuffle attempts to remove this reliance on neighbouring channels in the feature space, to force the models to learn a more general upsampling method.

Phase shuffle blocks were first used in the WaveNet paper [10] and we have shown to reduce the chance that we have phase-based tonal artifacts significantly. In the WaveNet paper, they use these phase-based shuffle blocks on the discriminator since its generator learns over a latent space, so it can already alter the phase of generations to make its job harder. However, we could easily add these blocks to the generator. Figure 4.4 demonstrates that this approach is successful at removing artifacts in the signal, which is a good sign for future development. However, Table 4.5 shows that this altered approach falls behind on our evaluation metrics despite overcoming this artifacting problem.

	No difference	+ Phase Shuffle(2)	+ Phase Shuffle(20)
SNR	$\mathbf{18.3 \pm 1.926}$	7.26 ± 5.222	-2.19 ± 1.020
LSD	$\mathbf{2.22 \pm 0.018}$	2.34 ± 0.018	2.87 ± 0.013
MOS	$\mathbf{2.73 \pm 0.638}$	1.53 ± 0.410	2.27 ± 0.210

Table 4.5: A table showing the effects of augmenting the AudioUNet structure with different degrees of phase shuffle. The best results for each row is shown in bold.

Chapter 5

Conclusions

This chapter summarises the project with a discussion of what went well, what lessons I have learnt and a few thoughts on applications of this project and potential ways to improve upon it.

Working on this project was an enjoyable experience and gave me the opportunity to learn about generative modelling and audio processing. I particularly enjoyed reading all of the recent papers on the topic of audio synthesis. This being an ongoing area of research, numerous papers even came out during the development of the project [28, 27].

5.1 Project Achievements

As set out in previous chapters, this project has been a success in that it has met all of the high and medium priority criteria. It has also met the extension criteria for bit-rate upsampling. I implemented three deep learning models that can perform audio super-resolution better than the current best non-machine learning method.

My first step was to attempt to develop a post-upsampling model in the form of the AudioEDSR. This proved a useful investigation since the AudioEDSR is a more lightweight alternative to the pre-upsampling approaches that current literature focuses on [15, 22, 27]. The AudioEDSR produced better results than the existing AudioUNet model for the two times upsampling case. This was one of the major contributions of this project since there has not been any previous work into applying pre-upsampling deep learning models to audio super-resolution.

I also implemented the AudioUNet model. This was an already established pre-upsampling model [15], which showed very promising results. I then designed and implemented the AudioUNetGAN in order to explore the effectiveness of the adversarial framework on audio data. This was another major contribution to the field of audio super-resolution research since, to my knowledge, there has been no WGAN based approach taken previously.

5.2 Lessons Learnt

I also enjoyed adapting models commonly used for image upsampling and applying them to the under-explored domain of audio. From an educational standpoint, this project has proven invaluable for both my understanding and enthusiasm for cutting edge research. Another important lesson I learned was how to program using PyTorch, which made it a lot easier than it would otherwise have been to implement all of my models. I will no doubt be using this again in future endeavours.

There were also a few oversights made in the development of the project. In particular, a long time was spent investigating why the post upsampling models had tonal artifacts. It turns out that I am not the first person to discover these artifacts, and in fact, a paper was released halfway through the project development discussing artifacts in audio [28]. When encountering this issue, I assumed there was a flaw in my implementation of the model and re-programmed the whole model in TensorFlow; this was a bit extreme. In future, I will attempt to diagnose the problems more thoroughly before such drastic measures.

5.3 Future Development

Despite this project being a success and meeting all of the original success criteria, there are still some ways I can see that the model can be improved in the future. Although considering the fast pace of improvement in the field, some major improvement will likely be discovered within the next few years. I have summarised a few envisioned improvements below:

- Once an appropriate post-upsampling method for audio interpolation becomes more viable, the first extension mentioned in the project proposal would be possible. This would involve adapting the model's GAN structure to allow upsampling of arbitrary scales.
- The current project learns a deterministic mapping from the low-resolution signal to the super-resolved signal. The model could be changed to learn a distribution by adding some noise into the input. Alternatively stochastic differential equations (SDEs) could be used to train the AudioUNetGAN. SDEs are a recent development to the field [25] and are seeing a lot of recent research [33].
- Seeing how the model performs on different domains would also be of interest to me and a potential investigation for the future. Some examples of other domains you could apply this model to are: Recordings of different animals, recordings of the sea and music (e.g. on a phone call, hearing music on a phone call often sounds bad due to frequencies higher than human speech).

It would also be good to see these models used practically in the field of telephony.

Bibliography

[1] C.E. Shannon. "Communication in the Presence of Noise". In: *Proceedings of the IRE* 37.1 (1949), pp. 10–21. DOI: 10.1109/JRPROC.1949.232969.

[2] A. Gray and J. Markel. "Distance measures for speech processing". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 24 (1976), pp. 380–391.

[3] Kunihiko Fukushima. "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". In: *Biological Cybernetics* 36 (1980), pp. 193–202.

[4] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[5] Chao Dong et al. "Image Super-Resolution Using Deep Convolutional Networks". In: *CoRR* abs/1501.00092 (2015). arXiv: 1501.00092. URL: http://arxiv.org/abs/1501.00092.

[6] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[7] K. Li and C. Lee. "A deep neural network approach to speech bandwidth expansion". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, pp. 4395–4399. DOI: 10.1109/ICASSP.2015.7178801.

[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[9] Christian Ledig et al. "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network". In: *CoRR* abs/1609.04802 (2016). arXiv: 1609.04802. URL: http://arxiv.org/abs/1609.04802.

[10] Aäron van den Oord et al. "WaveNet: A Generative Model for Raw Audio". In: *CoRR* abs/1609.03499 (2016). arXiv: 1609.03499. URL: http://arxiv.org/abs/1609.03499.

[11] Wenzhe Shi et al. "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network". In: *CoRR* abs/1609.05158 (2016). arXiv: 1609.05158. URL: http://arxiv.org/abs/1609.05158.

[12] Wenzhe Shi et al. *Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network*. 2016. arXiv: 1609.05158 [cs.CV].

[13] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[14] Ishaan Gulrajani et al. "Improved Training of Wasserstein GANs". In: *CoRR* abs/1704.00028 (2017). arXiv: 1704.00028. URL: http://arxiv.org/abs/1704.00028.

[15] Volodymyr Kuleshov, S. Zayd Enam, and Stefano Ermon. *Audio Super Resolution using Neural Networks*. 2017. arXiv: 1708.00853 [cs.SD].

[16] Bee Lim et al. *Enhanced Deep Residual Networks for Single Image Super-Resolution*. 2017. arXiv: 1707.02921 [cs.CV].

[17] Aäron van den Oord et al. "Parallel WaveNet: Fast High-Fidelity Speech Synthesis". In: *CoRR* abs/1711.10433 (2017). arXiv: 1711.10433. URL: http://arxiv.org/abs/1711.10433.

[18] Chris Donahue, Julian J. McAuley, and Miller S. Puckette. "Synthesizing Audio with Generative Adversarial Networks". In: *CoRR* abs/1802.04208 (2018). arXiv: 1802.04208. URL: http://arxiv.org/abs/1802.04208.

[19] A. Emin Orhan and Xaq Pitkow. *Skip Connections Eliminate Singularities*. 2018. arXiv: 1701.09175 [cs.NE].

[20] Hang Zhao et al. *Loss Functions for Neural Networks for Image Processing*. 2018. arXiv: 1511.08861 [cs.CV].

[21] Jason Brownlee. *Generative Adversarial Networks with Python. Deep Learning Generative Models for Image Synthesis and Image Translation*. 2019. URL: https://machinelearningmastery.com/generative_adversarial_networks/.

[22] Sung Kim and Visvesh Sathe. "Bandwidth Extension on Raw Audio via Generative Adversarial Networks". In: *CoRR* abs/1903.09027 (2019). arXiv: 1903.09027. URL: http://arxiv.org/abs/1903.09027.

[23] Kaichao You et al. "Learning Stages: Phenomenon, Root Cause, Mechanism Hypothesis, and Implications". In: *CoRR* abs/1908.01878 (2019). arXiv: 1908.01878. URL: http://arxiv.org/abs/1908.01878.

[24] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[25] Haoyang Cao and Xin Guo. "Approximation and convergence of GANs training: an SDE approach". In: *CoRR* abs/2006.02047 (2020). arXiv: 2006.02047. URL: https://arxiv.org/abs/2006.02047.

[26] Chuan Tan, Jin Zhu, and Pietro Lio'. *Arbitrary Scale Super-Resolution for Brain MRI Images*. 2020. arXiv: 2004.02086 [eess.IV].

[27] Junhyeok Lee and Seungu Han. *NU-Wave: A Diffusion Probabilistic Model for Neural Audio Upsampling*. 2021. arXiv: 2104.02321 [eess.AS].

[28] Jordi Pons et al. *Upsampling artifacts in neural audio synthesis*. 2021. arXiv: 2010.14356 [cs.SD].

[29] URL: https://www.who.int/news/item/30-01-2020-statement-on-the-second-meeting-of-the-international-health-regulations-(2005)-emergency-committee-regarding-the-outbreak-of-novel-coronavirus-(2019-ncov).

[30] Yamagishi, Junichi; Veaux, Christophe; MacDonald, Kirsten. (2019). CSTR VCTK Corpus: English Multi-speaker Corpus for CSTR Voice Cloning Toolkit (version 0.92), [sound]. University of Edinburgh. The Centre for Speech Technology Research (CSTR). https://doi.org/10.7488/ds/2645.

[31] URL: https://github.com/pancakewaffles/metasrgan-tutorial/blob/master/Tutorial_SuperResolution.ipynb.

[32] URL: https://gist.github.com/davidaknowles/6e95a643adaf3960d1648a6b369e9d0b.

[33] URL: https://openreview.net/forum?id=padYzanQNbg.

Appendix A

Training Diagrams

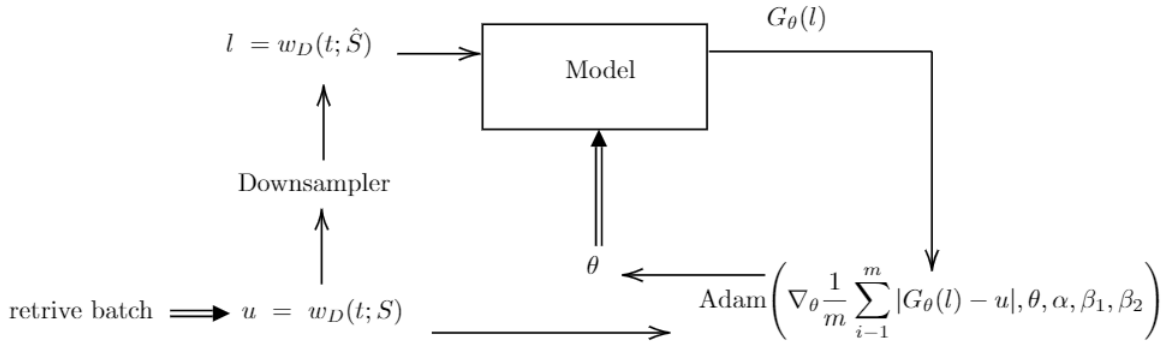

Figure A.1: The training process for the AudioEDSR. The double lined arrows mark the start of a new cycle

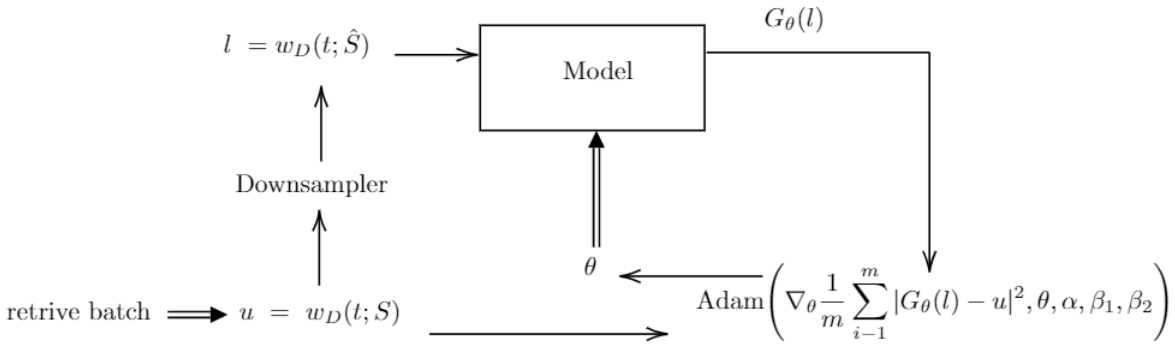

Figure A.2: The training process for the AudioUNet

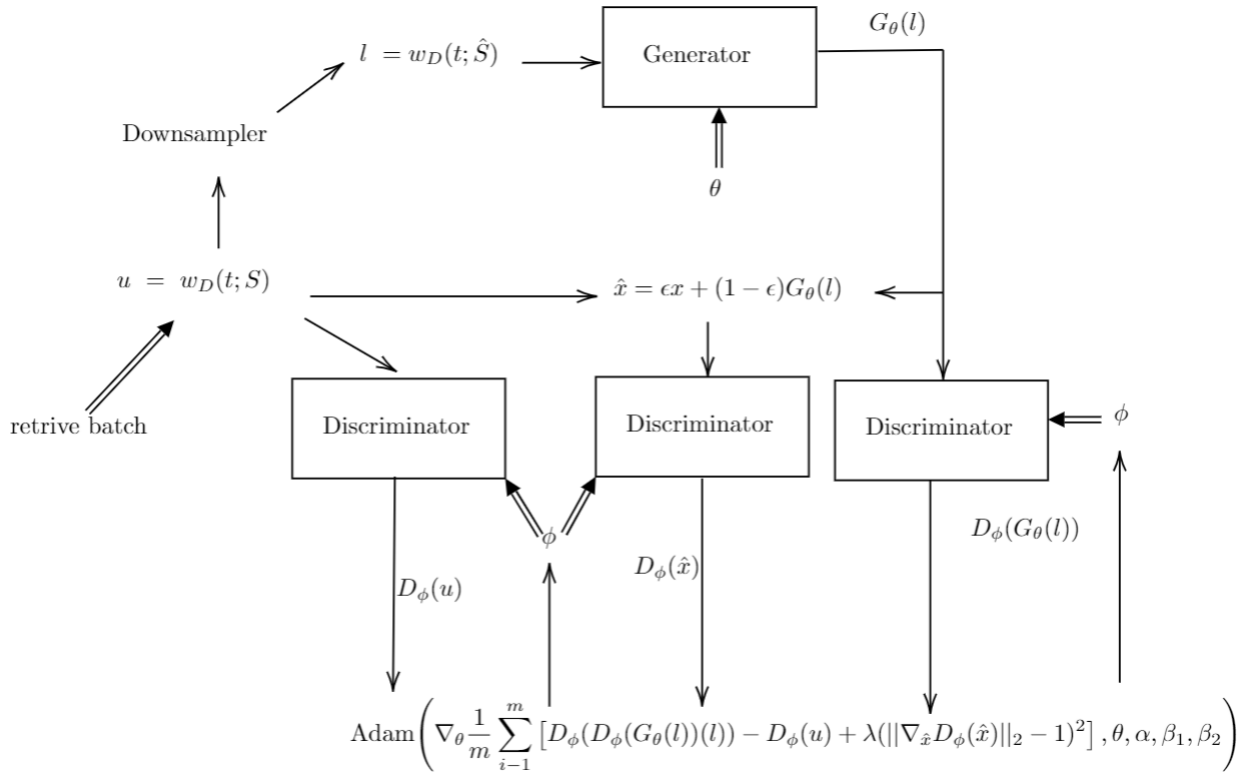

Figure A.3: The training process for the AudioUNetGAN discriminator

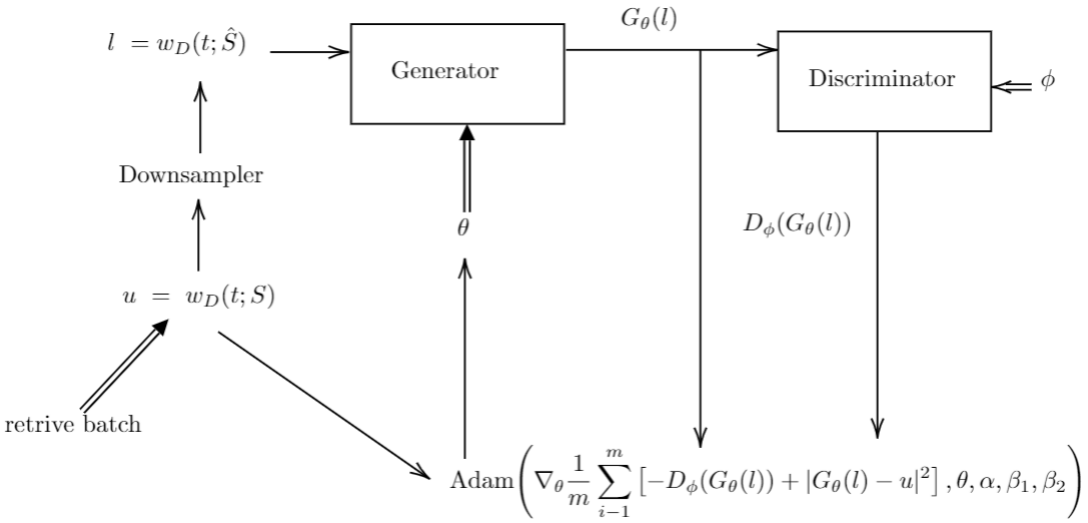

Figure A.4: The training process for the AudioUNetGAN generator

Appendix B

Project Proposal

Project Proposal
Improved Audio Super-Resolution

Candidate 2335E

October 23, 2020

Main Project Supervisor:	Ramon Viñas Torné
Project Co-Supervisors:	Professor Pietro Liò, Jin Zhu, Alexander Campbell
Director of Studies:	Dr Amanda Prorok
Project Overseers:	Professor John Daugman, Dr Nicolas Lane
Project Originators:	Alexander Campbell

Introduction

Audio super-resolution is the process of taking in a low-resolution (LR) sound file with sample rate L and transforming it into a high-resolution (HR) version with sample rate H where $L < H$. The upsampling ratio is defined as $r = H/L$, and this project aims to develop a model for performing audio super-resolution on fixed integer values of r. Solving this problem will have the advantage of allowing audio files to require less space in memory without compromising the quality (or even improving upon it).

Over the past few years, there have been many impressive techniques used to achieve fantastic results on super-resolution applied to images. While there are some papers on performing super-resolution on audio, there are distinctly less than for images. This means that techniques that have been developed for images have the potential to improve audio super-resolution. In this project, we will look at repurposing such techniques and exploring their effects on improving the sample rate of audio.

In this project, I aim to develop a framework based on a generative adversarial network (GAN) to perform super-resolution on audio files. For the training set, I plan to downsample a selection of audio files from the chosen database. When training, GANs are a two-part system. The first part is a generator; this will take in an LR sound file and try and upscale it to an HR sound file. The second is a discriminator; this will be given samples of both actual HR files and files produced by the generator and tries to work out if it is real or generated. Based on whether the generator manages to deceive the discriminator or not, the system will change the way both either generate or discriminate. Eventually, both of these parts will become very good at their generation and discrimination ability, respectively. This means we can ignore the discriminator, and now we have a generator that produces HR audio for LR samples.

This project has obvious implications in both telecommunications and compression of sound files. One of the potential benefits is that it will allow audio transmission to require less data. Another benefit to the project is that it will improve the quality of audio files sent across a network. A specific example of its use is a VoIP service using this algorithm to reduce the cost of transmission without compromising much quality. Using similar ideas to reconstruct files of reduced size has also seen much success recently with video [1]. Altogether, it is clear that this project has the potential to make a difference in communications.

Evaluation

To measure the performance of the model, I plan to use two quantitative metrics: signal to noise ratio (SNR) and log-spectral distance (LSD) [2]. The SNR is measured in decibels and is defined as:

$$\text{SNR}(x, x_{ref}) = 10 \log_{10} \frac{||x_{ref}||_2^2}{||x - x_{ref}||_2^2} \tag{1}$$

where x is the actual signal and x_{ref} is our approximation. The LSD is defined as:

$$\text{LSD}(x, x_{ref}) = \frac{1}{W} \sum_{w=1}^{W} \sqrt{\frac{1}{K} \sum_{k=1}^{n} (\log_{10} \frac{|x(w,k)|^2}{|x_{ref}(w,k)|^2})^2} \tag{2}$$

where w represents the window bins and k represents the frequency bins.

SNR is a good metric because it quantifiers what proportion of the signal is useful, which we want to maximise. Whereas LSD is a good metric to use because it tells us how close the generated signal was to the original so helps determine how accurate the model is.

I will also evaluate the results by computing a mean opinion score (MOS). In other words, I will ask a small number of people to rate the overall quality of the sound produced the algorithm. This is a more subjective metric, but it is also important that the signal produced sounds good, something this metric measures. In this project, I will ask people to rate the quality and take the arithmetic mean as the overall score.

Success Criteria

In order to measure the success of the project, I will judge it against the following criteria:

1. **Audio Processing:** The algorithm will be able to receive (processed) input data and output to a well-defined format.
2. **Model Checking:** The algorithm will have a higher SNR and LSD score than a cubic B-spline implementation.
3. **Quality Checking:** The algorithm will produce sound files with a MOS higher than a cubic B-spline.

Potential Extensions

Once the model has convincingly passed all of the success criteria, there are multiple extensions I have in mind to add functionality and improve performance:

1. At the moment we can only perform super-resolution with fixed integer scale factors. However, we can apply a meta-learning model [3] which will add another section to the model that predicts the weights' models of arbitrary scales. This means we only need to train one neural network that can generate the weights of a model that upscales by any amount. This helps reduce learning times since we learn how to generate models. This has already seen great success when applied to images [4]. I could adapt the meta-super resolution GAN (Meta-SRGAN) framework to audio.
2. We currently only looked at improving the sample rate of the audio. However, we could also look at up-scaling the bit rate. This would allow for even less data to be collected from audio samples amplifying the effectiveness of the base model. We can do this by altering the way that data is processed in the input and altering the training model to take this into account too.
3. Augment the training data with noise. Doing this will make the model more robust to invariances in the data so that it will be more robust for general usage. For example, If someone is talking on the phone in a busy park, we do not want the algorithm to amplify background noise.

Dataset

I intend to use the VCTK corpus [5] as the primary dataset. This dataset contains speech from 110 different English speakers, each of whom read about 400 sentences from various newspapers. These sentences were chosen by an algorithm designed to maximise the phonetic coverage [6]. All the data in this dataset was recorded with the same microphone. The microphone has a sampling frequency of 96 kHz and a quantisation depth of 24 bits. All of the stored recordings were converted into having a 16-bit quantisation depth and were downsampled to a 48 kHz sampling frequency.

I think that it is a great database to use because the project aims to explore ways to improve sound signals specifically with respect to communications. This is because, in communications, data transmitted is often vocal. Making the data set specific to one type of audio will help to highlight the potential of this methodology since there will be more features to pick up on. This dataset is commonly used in other audio super-resolution works [2, 7, 8] so it will be simple to benchmark the algorithm against existing baselines for evaluation.

Starting point

I have studied machine learning and AI as part of my degree. I otherwise have no experience in this area. In order to learn more, I will read "Generative Adversarial Networks with Python" by Jason Brownlee. I will also attend the lectures for "Machine Learning and Bayesian Inference" and "Information Theory". Additionally, I will read up on any relevant deep learning packages for Python, for example, PyTorch. I will also take tutorials where necessary.

Work Plan

I plan to start working on this project on the 26/10/2020. This will allow me time to start reading up on GANs and learn about pipeline structures for super-resolution tasks before beginning work on implementation.

Michaelmas

- **Michaelmas weeks 3–4** (26th October - 8th November)
  Research GANs, super-resolution and audio processing techniques and design a preliminary pipeline for the program.
- **Michaelmas weeks 5–6** (9th November - 22nd November)
  Finalise language selections for implementation and read up on reliant packages. Also, write a program for processing the dataset into a suitable input format for the algorithm and decide on the output format.
  milestone *Have a system for converting a dataset into an input for the program*
- **Michaelmas weeks 7–8** (23rd November - 6th December)
  Implement a simple system for interpolating sound data for super-resolution such as a cubic B-spline or an EDSR (Enhanced deep super-resolution network) for the audio data. Here I will also finalise the pipeline for the core program.
- **Christmas Vacation** (7th December - 17th January)
  Part one: implement and train the decided upon pipeline for the neural network

  milestone *Have a working implementation of the program*

  Part two: complete the testing and evaluation of the model.

  Extension work can be completed here or later, time allowing

Lent

- **Lent weeks 1-2** (18th January - 31st January)
  Write up a progress report and prepare a presentation.

  Progress Report Deadline: 5th February

- **Lent weeks 3-4** (1st February - 14th February)
  Evaluate the success criteria and create a skeleton of the report.

- **Lent weeks 5-6** (15th February - 28th February)
  Write up the introduction and preparation sections of the dissertation.

- **Lent weeks 7-8** (1st March- 14th March)
  Write up the implementation section of the dissertation.

- **Easter Vacation** (15th March - 25th April)
  Part one: Write up the evaluation and conclusion sections of the dissertation.

  milestone *Have a draft of all sections of the dissertation*

  Part two: Read through all of the dissertation correcting any mistakes and elaborating on any unclear explanations. Here I will also send it to my supervisors to gather their feedback.

Easter

- **Easter weeks 1-2** (26th April - 9th May)
  Read through the dissertation to make sure everything has clearly been explained and make relevant adjustments based on any advice given. Here I will also double-check that my final write up conforms to any regulations it needs to, e.g. word count and submission format.
  milestone *Have a competed dissertation ready for submission*

  Final Submission Deadline: 14th May

Resource Declaration

I will do most of the development on my own computer (I accept full responsibility for this machine and I have made contingency plans to protect myself against hardware and/or software failure). I will also make sure that recent versions are backed up on both my computer and Github. I will need access to the labs' GPUs to train my neural networks. I will also need approximately 15GB to store the code and the dataset.

References

[1] https://developer.nvidia.com/maxine

[2] Volodymyr Kuleshov and S. Zayd Enam and Stefano Ermon: Audio Super Resolution using Neural Networks. CoRR, abs/1708.00853 2017

[3] Xuecai Hu, Haoyuan Mu, Xiangyu Zhang, Zilei Wang, Tieniu Tan, and Jian Sun. Meta-sr: A magnification-arbitrary network for super-resolution. CoRR, abs/1903.00875, 2019.

[4] Tan, Chuan & Zhu, Jin & Lio, Pietro. (2020). Arbitrary Scale Super-Resolution for Brain MRI Images.

[5] Yamagishi, Junichi; Veaux, Christophe; MacDonald, Kirsten. (2019). CSTR VCTK Corpus: English Multi-speaker Corpus for CSTR Voice Cloning Toolkit (version 0.92), [sound]. University of Edinburgh. The Centre for Speech Technology Research (CSTR). https://doi.org/10.7488/ds/2645.

[6] C. Veaux, J. Yamagishi and S. King, "The voice bank corpus: Design, collection and data analysis of a large regional accent speech database", https://doi.org/10.1109/ICSDA.2013.6709856

[7] Sung Kim and Visvesh Sathe: Bandwidth Extension on Raw Audio via Generative Adversarial Networks. CoRR,abs/1903.09027, 2019

[8] https://cs230.stanford.edu/projects_ spring_ 2018/reports/8289876.pdf