The University of Manchester
Department of Computer Science
School of Engineering

Super Resolution and Frame Interpolation using Autoencoders

Dragos-Constantin Trett

April 2023

Project Supervisor: Dr. Oliver Rhodes
Second Examiner: Prof. Aphrodite Galata

A thesis submitted for the degree of
BSc(Hons) Computer Science
with Industrial Experience

Abstract

This report explores the applications of Autoencoders, focusing on Super Resolution and Frame Interpolation. Autoencoders are a class of Artificial Neural Networks commonly used for encoding information to a latent space representation and decoding it by reconstructing the source information. Super Resolution is the process of increasing the quality of an image, while Frame Resolution is the process of improving the quality of a video by introducing generated frames. Both of these problems have already established solutions and in this report we will attempt to bring to light the possible optimisations and improvements powered by Deep Learning techniques.

As the technology landscape is heading in a direction of integrating specialised hardware for Artificial Intelligence in commercial products, like phones and laptops, Deep Learning techniques will become more widely available to the public. With this trend we can hope that Autoencoders could one day play a significant role in optimising the way data is transferred on the world wide web. Autoencoders can be used to reduce the amount of network traffic, and thus energy consumption, by compressing imagery before transmission or by using lower quality data which it can transform to high quality one.

In this report we will attempt to create proof of concept solutions for Super Resolution and Image Interpolation using Autoencoders. While this kind of technology is far from wide commercialisation, our models show promising results and demonstrate the beginning of a new field of study dedicated to optimising any existing technology with Deep Learning techniques.

Acknowledgements

I would like to thank my supervisor, Dr Oliver Rhodes for his interest, support and continuous guidance throughout the whole project.

I am most grateful for my family and the support they provided throughout my university years. I would like to extend my deepest gratitude to my parents for always supporting my endeavours and to my brother for always guiding me in the right direction and providing me the video materials used in this paper.

Contents

Chapter 1
Introduction

In the past few years there has been a surge of research and development in the field of Artificial Intelligence and more specifically Deep Learning, which is a subset of machine learning powered by Artificial Neural Networks (ANNs).[42] ANNs are a family of technologies focused on mimicking the learning process of the human brain by defining structures of neurons with various connections between them. The surge was supported by great improvements in tools and software libraries intended for implementing the ANNs, like TensorFlow and PyTorch, and also by large companies and governments investing in AI software.

Many of the amazing achievements of ANNs we have seen are focused on Computer Vision (CV), which is the sub-field of Artificial Intelligence focused on processing and understanding visual data, like images and videos.[6] In this project, I will research, implement and present deep learning models based on the Autoencoder architecture[14] and assign them various computer vision tasks.

1.1 Aims & Objectives

To extend the previous statement, the main focus of this paper can be divided in 2 objectives: Super Resolution (SR) and Frame Interpolation (FI), both of which can be achieved with deep learning autoencoders. As a second objective, thru this research, I am hoping to gain more knowledge on the Autoencoder architecture limitations and benefits, as well as other kind of applications and variations from the original concept.

Both SR and FI are image processing procedures belonging to the Computer Vision field that have been researched for the past several decades, the first mentions of them being as early as 1990. However, the accumulate knowledge is mostly focused on using mathematical algorithms to find a solution, so in this project we will attempt to solve both of these problems using autoencoders and then assess and critique their performance.

1.1.1 Autoencoders

Autoencoders are the key concept behind this project. To understand how autoencoders can be applied to our tasks, it is fundamental to understand the building blocks powering them. The next chapter aims to introduce the reader to the field of deep learning and provide an overview of the Perceptron, the neuron representation[24]. It will also present some additional concepts used in my implementations, like convolution[31] [37] and batch normalisation[34] before presenting Autoencoders.

1.1.2 Super Resolution

To begin with, Super-Resolution is an image/video processing technique intended for upscaling low resolution images to high resolution ones, improving the quality. It can be used in various applications such as upscaling video game frames, medical image processing and live streaming.[12]

Regarding SR, this paper is set out to achieve the following objectives:

- Gain an understanding of the conventional methods of SR as well as modern deep learning based methods
- Build, train and test a proof of concept artificial neural network based on the autoencoder architecture that achieves single image SR as well as single image compression on the Modified National Institute of Standards and Technology (MNIST) database[33]
- Analyse, critique and validate the performance and accuracy of the described model

To evaluate the models resulted from the implementations, there are 2 kinds of metrics to be assessed. The first metric is the quantitative one and it can come under different forms. For every output of the autoencoder we can analyse the mean squared error (MSE) of the output pixels compared to the ground truth, as well as its derivative metric the peak signal-to-noise ration (PSNR). We can also analyse the structural differences between the outputs and ground truth by calculating the Structural Similarity Index Metric (SSIM). While the first 2 metrics can show us how close the models are to the expected output, the SSIM can predict the perceived quality of the output images. The second kind of metric is the qualitative one, which represents the effect that SR has on the user experience when viewing upscaled images. The expectations for this model are to improve the quality of the image without deviating from the original input image structure and to successfully compress the source images to a lower dimension representation.

1.1.3 Frame Interpolation

Frame Interpolation (FI) is the process of generating in-between images from a sequence of images or frames. It is used extensively in the movie and animation industry because it aims to improve the general image quality of a video.[18]

In this project, for Frame Interpolation, the aim is to achieve the following objectives:

- Gain an understanding of the conventional methods of FI as well as modern deep learning based methods
- Build, train and test a proof of concept artificial neural network based on the autoencoder architecture that achieves FI on any given input video.
- Analyse, critique and validate the performance and accuracy of the described model

The quantitative evaluation of this model is similar if not identical to the previous task. For qualitative evaluation on the other hand, the process is more complex. First of all, this paper will aim to analyse the singular image output of the autoencoder models. The focus on this will be the image reconstruction accuracy, with a special assessment on any visual artifacts it might create. Second of all, this paper will also aim to analyse the effect that FI has on the user experience when observing the generated videos compared to the source videos.

1.2 Motivation

The main source of inspiration for this project is Nvidia's Deep Learning Super Sampling (DLSS) technology family[28]. DLSS is a system powered by autoencoders that can perform Super Resolution and Frame Interpolation in video games. Its aim is to greatly reduce the performance impact of new video games that require significantly more computational power by using lower quality frame outputs from the game engine to generate higher quality frames and extra in between frames as well. This resulted in better performance for a lower computational cost in all the video games it is being incorporated. However, this technology is mostly limited to video games.

My project aims to demonstrate that this kind of technology can be adopted in many other applications and systems to improve performance and reduce running costs. In today's environment, where the world wide web and especially social media platforms are greatly populated by images and videos, we have to look for ways to reduce our energy footprint by optimising the transmission of information over the internet. The technology created by Nvidia represents only the beginning of a new field of study aimed at optimising the way we consume data on the internet. Autoencoders can allow for some parts of data to be omitted when transmitting, greatly reducing network traffic. They could also allow for the data to be encoded in a lower dimension representation and serve it to the user that can decode it back to the original format. Personally, I do not expect this kind of technology to reach commercialisation any time soon, however, as soon as the computational power of our devices increases enough, companies will be interested in this technology for the cost saving benefit.

Chapter 2
Background and Theory

In this chapter we will introduce the concepts and techniques that powered the neural networks used in this research. Firstly, an overview of the deep learning field will be provided, followed by a presentation of the fundamental building block of neural networks, the perceptron, and how we can use it in more complex models. With the basic notions established we will dive into Convolutional Neural Networks and Autoencoders, which represent the most important neural network architecture types for our research. These can enable our neural networks to perform super resolution and video frame generation techniques so we will also present some existing technologies for this as well as how our implementation was created conceptually.

2.1 Deep Learning

Deep learning is a subfield of machine learning that involves the creation and training of neural networks with many layers. It is inspired by the structure and function of the human brain, where information is processed through a network of neurons. In deep learning, each layer of the neural network learns to recognize different features of the input data, and the output of one layer serves as the input for the next layer. This allows the network to learn increasingly complex representations of the data, ultimately enabling it to make accurate predictions or classifications. Deep learning has been applied to a wide range of tasks, including image and speech recognition, natural language processing and autonomous driving. It has achieved remarkable results in many of these domains and is widely regarded as one of the most promising areas of artificial intelligence.[40]

Deep learning algorithms can be classified into several categories based on their architecture and the type of learning they employ. Some of the most common classifications include:

1. **Feedforward Neural Networks**. This is the most basic type of neural network, consisting of input layer, several hidden layers and an output layer and having information flow in one direction from the input layer to the output layer[36]

2. **Convolutional Neural Networks (CNNs)**. This kind of neural networks is specialized for image processing and computer vision tasks. They use convolutional layers to extract features from images and pooling layers to reduce the dimensionality of the feature maps.[25]

3. **Recurrent Neural Networks (RNNs)**. By using recurrent connections between neurons to capture the temporal dependencies in the data, these neural networks are designed for sequential data such as speech, text and time series data.[19]

4. **Generative Adversarial Networks (GANs)**. Consisting of a generator and a discriminator, this kind of network generates new data samples that are similar to the training data, and evaluates whether the data is real or fake.[9]

5. **Autoencoders**. These are unsupervised learning algorithms that learn to encode the input data into a lower-dimensional representation and decode it back into the original input.[14]

6. **Reinforcement Learning**. This classification defines training an agent to perform actions in a given environment and rewarding or penalising it based on it's performance.[26]

Since the concept of Super Sampling heavily relies on the reconstruction of images under different formats, a more complex description will be provided for the Convolutional Neural Networks and the Autoencoders as well. However, in order to ensure understanding, an overview of the basic elements of neural networks will be presented first.

2.2 Perceptron

A single neuron, commonly known as a perceptron, is the fundamental building block of every neural network. Each input node is connected to a corresponding weight, which determines the strength of the input's influence on the output.[24] The output node computes a weighted sum of the inputs and applies an activation function, such as a step function, to produce an output. This process is known as the forward propagation process of the perceptron and is illustrated in Figure 2.1. Mathematically, it can be represented in linear algebra terms.

$$\hat{y} = g(W_0 + X^T W) \tag{2.1}$$

Where $X = \begin{bmatrix} X_1, \\ X_2, \\ ... \\ X_m \end{bmatrix}$ are the input values , $W = \begin{bmatrix} W_1, \\ W_2, \\ ... \\ W_m \end{bmatrix}$ are the learnable weights and W_0 is the bias term.

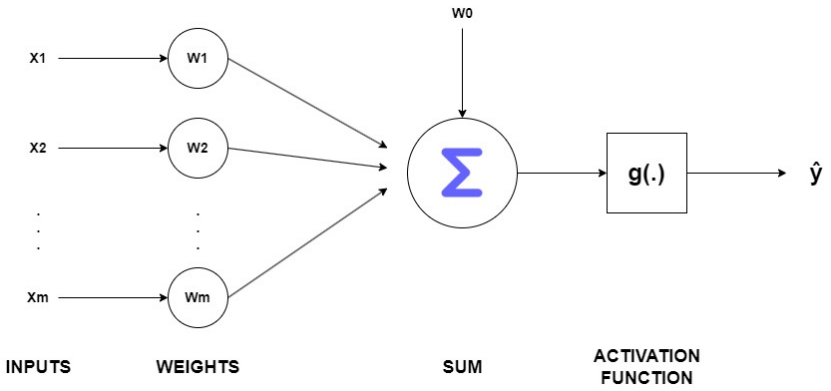

Figure 2.1: Perceptron.

The activation function of a perceptron is a non-linear function that takes the weighted sum of inputs and produces an output. The purpose of the activation function is to introduce non-linearity into the output of the neuron, allowing it to model complex relationships between inputs and outputs. The ReLU activation function (Rectified Linear Unit) is defined as $f(x) = max(0, x)$, where x is the input to the neuron. It is a piecewise linear function that sets all negative values to zero, and keeps positive

values unchanged. The ReLU function is commonly used in neural networks due to its simplicity, computational efficiency, and effectiveness in preventing the vanishing gradient problem. However, it can suffer from a problem known as "dying ReLU", where some neurons can become inactive and stop contributing to the output due to having a negative bias term.[20]

While the perceptron is limited in its ability to model complex patterns, it forms the basis for more sophisticated neural network architectures, such as multi-layer perceptrons (MLPs), which can model non-linear relationships between inputs and outputs.

2.3 Multilayer Perceptron (MLP)

A multilayer perceptron (MLP) is a type of feed forward neural network that consists of three or more layers of nodes. It is also known as a fully connected neural network, as each node in one layer is connected to every node in the next layer.[17] The first layer of the MLP receives the input data and the last layer produces the output of the network. The layers in between are called hidden layers, as their nodes are not directly connected to the input or output. The nodes in the hidden layers apply a non-linear activation function to the weighted sum of their inputs, just like in a single-layer perceptron.[17] The outputs of the nodes in one layer become the inputs to the nodes in the next layer, until the final output is produced, as shown in Figure 2.2.

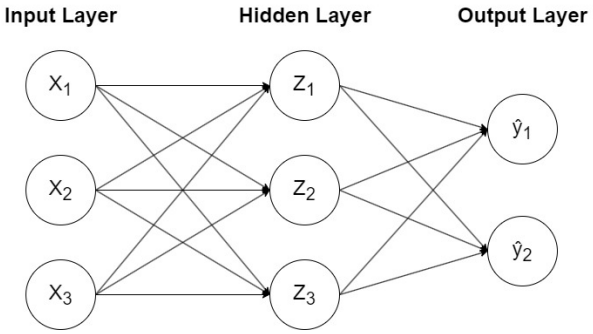

Figure 2.2: Three layer MLP.

They can be trained using supervised learning algorithms, such as backpropagation, which adjust the weights of the connections between nodes based on the error between the predicted output and the actual output.[17] Despite the fact that MLPs are powerful models that can learn complex non-linear relationships between inputs and outputs, they show a lack of effectiveness when interacting with data in the form of matrices, such as imagery. This is a disadvantage that can be addressed using Convolutional Neural Networks.

2.4 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a type of deep neural network most commonly used for analyzing visual imagery. They were inspired by the structure of the human visual system, where neurons in the brain respond to specific regions of the visual field. Unlike MLPs, the underlying architecture of a CNN is built for extracting features from

images.[25] A common CNN architecture would be composed of 3 layers of neurons. The first layer would apply a convolution algorithm over the image using learnable parameters for the kernel, enabling the CNN to extract image features. It would be common in these networks for the second layer to be pooling layer, where the information resulted from the previous convolution is reduced spatially to highlight the most important extracted features. The final layer or layers of the networks would most commonly be fully connected neural layers tasked with interpreting the extracted features and mapping them to the output.

2.4.1 Convolution

Convolution is a fundamental operation in CNNs that is used for local feature extraction. The idea behind convolution is to slide a small kernel, also known as filter, over the input image, performing an element-wise multiplication between the kernel and the corresponding patch of the input image at each location, and summing the results to produce a single output value.[31] In Figure 2.3 we display an example of convolution over a 7 by 7 input matrix using a 3 by 3. In the input matrix we have highlighted in red a patch meant to be multiplied element wise with the kernel, summing to the resulting values to 4 and placing it in the resulted matrix. We can observe that after the operation is completed, our result has decreased in size compared to the input. In this scenario this can be avoided by padding the input matrix on the sides with our preferred values, however, the resulting matrix can also decrease in size if the stride of movement over patches is greater than 1.

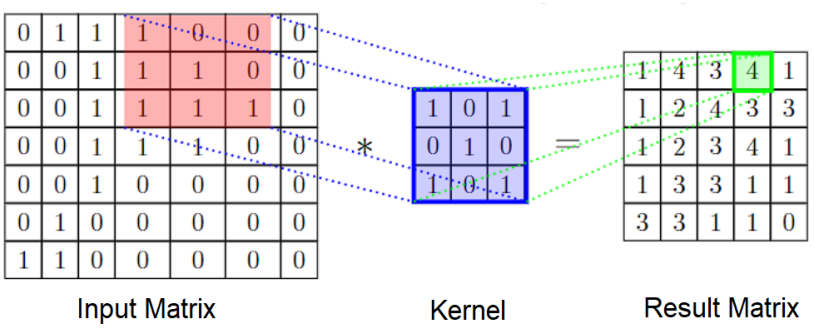

Figure 2.3: Example Convolution with 3x3 Kernel.

Returning to the context of CNNs, the kernel of the convolution is made up of learnable parameters, enabling the network in training to learn what features to extract to minimise error.[25] In Figure 2.4 we illustrate the architecture of a simplistic CNN, containing a convolution layer, followed by a pooling layer and flattened fully connected layers. The convolution operation transforms 1 channel into 8 channels, thus there are 8 different kernels that are applied to the image using a stride of 2. The results of the convolution are then pooled by choosing the maximum value within a window of size 2 and flattened to be passed to the fully connected layers.

2.4.2 Batch normalisation

Batch normalization is a technique used in deep neural networks to normalize the input values to each layer of the network. It was originally designed to address the problem

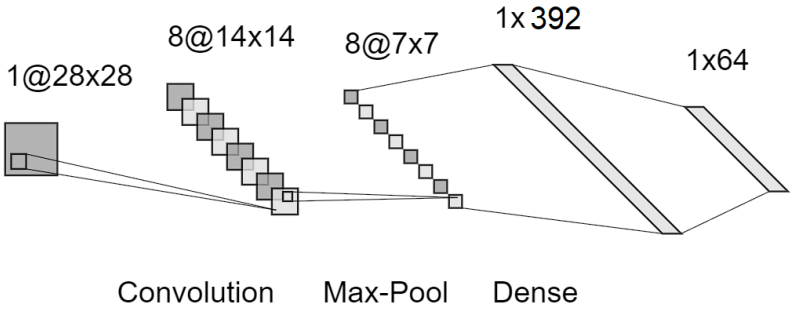

Figure 2.4: Example CNN Architecture.

of "internal covariate shift," which is the change in the distribution of the input values to each layer during training. [22] The basic idea of batch normalization is to apply normalization to the input values of each layer by subtracting the mean and dividing by the standard deviation of the input values within a batch of training examples. This effectively centers and scales the inputs, which can improve the training stability and convergence of the network.[34]

2.5 Autoencoders

Autoencoders are a type of neural network that are used to learn compressed representations of input data. They consist of an encoder, which maps the input data to a compressed representation, and a decoder, which maps the compressed representation back to the original input data.[7] This can be visualised in Figure 2.5.

Most commonly, they use multiple layers of neurons to learn hierarchical representations of the input data. The encoder and decoder networks are typically mirror images of each other, with each layer learning to represent a more abstract and compressed version of the data. Autoencoders are unsupervised, meaning they do not require labeled data, and are often used for tasks such as image and video compression, anomaly detection, and data generation.[14]

Generally, the formal definition of the autoencoder problem is to minimise overall reconstruction error or distortions for functions A and B:

$$\min_{A,B} \sum_{i=1}^{n} \Delta \left(A \circ B \left(x_i \right), y_i \right) \tag{2.2}$$

, where:

- A is the function representing the encoder and B is the decoder,
- Δ is the a dissimilarity or distortion function,
- and $X = \{x_1, x_2, ..., x_n\}$ and $Y = \{y_1, y_2, ..., y_n\}$ are the training and target vectors respectively.

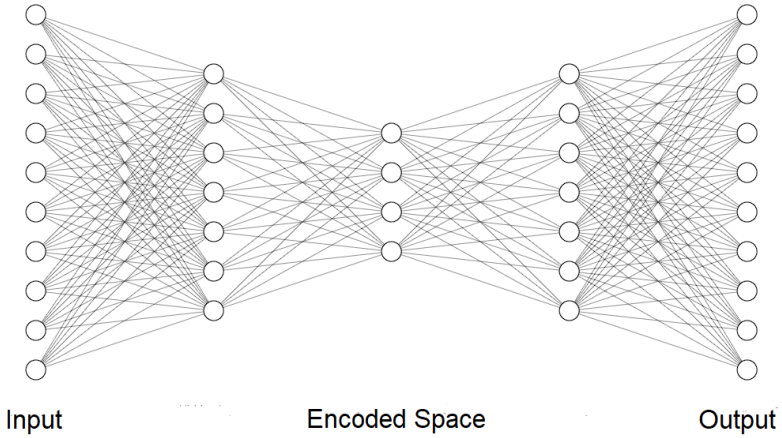

Figure 2.5: Autoencoder example.

A very valuable property of the autoencoders is its adaptability, as it can take different forms depending on the problem circumstances. A fitting example of this would be how we can introduce convolutional layers to better fit the autoencoder with image processing tasks. The overall concept of the autoencoders can be used in many different applications, and when focusing on deep learning and researching published literature we observe that autoencoders have a wide range of applications in various fields such as computer vision, natural language processing, and signal processing.

2.5.1 Image Compression

One of the most common applications is image compression. The autoencoder is tasked with transforming a given image input into its latent space representation and then using it to reconstruct the original image, where the latent space is the bottleneck of the autoencoder and where the decoder is coupled with the encoder.[23] It is most common for these kind of autoencoders to contain convolutional layers with ReLU activation functions and some hidden fully connected neuron layers for the latent space. It can be considered that this process has been successful due to its ability to learn efficient representations of image data, achieving high compression ratios while maintaining good image quality. However, there are some challenges associated with using autoencoders for image compression. One of the main challenges is finding the correct balance between compression ratio and image reconstruction quality.[39] Additionally, training the autoencoders can be computationally expensive, especially when the training dataset is varied and which can limit their practical applications.

Image compression using autoencoders is a promising technique for reducing image sizes with good image quality. By carefully tuning the parameters of the autoencoder and training it on a large data set of images, it is possible to achieve efficient compression with good image quality. However, there are still challenges that need to be addressed in order to make autoencoder-based compression methods more practical for real-world applications.

2.5.2 Anomaly Detection

Another, perhaps more captivating, application of autoencoders is the problem of anomaly detection in various environments and data sets. The autoencoder is once again tasked with reconstructing the input data and is trained with data representing normal circumstances, where there are no anomalies. Because the autoencoder has only observed and captured the most important features and patterns of the normal data, when presented with data containing anomalies the reconstruction error can be observed to increase, suggesting abnormal data.

However,there are other different ways to implement autoencoders for anomaly detection, depending on the type of data and the specific application. As mentioned, one approach is to calculate the reconstruction error and perform thresholding. Another method would be to use variational autoencoders, where the network learns a probabilistic model of the data. This allows them to model the distribution of normal data and generate new samples that follow the same distribution. Anomalous data that deviate significantly from the learned distribution can be identified as anomalies. A great practical example for anomaly detection autoencoders would be Kitsune[41], tasked with detecting any suspicious or ill intended network traffic. The autoencoders are trained online using a sliding window of past network traffic data. The input to each autoencoder is a sequence of packets, and the output is the reconstruction of the same sequence. The loss between the input and output is used to update the weights of the autoencoder in an online fashion.

2.6 Super-Resolution (SR)

2.6.1 Conventional Super Resolution

Super-Resolution (SR) is an image/video processing technique intended for upscaling low resolution images to high resolution ones, improving the quality. It can be used in various applications such as upscaling video games frames, medical image processing and live streaming. Before deep learning reached the point where it can be used to solve the problem of SR, there were already a few other established techniques for solving it. Some of the main traditional methods are based on interpolation[12], a technique that involves introducing new pixels in between the original ones to increase the resolution. Some of the common functions for calculating the new pixel values would be bicubic interpolation and Lanczos interpolation.[15] Figure 2.6 shows the result of different image interpolation techniques.[30] As it can be observed, these kind of methods lack the autoencoder's ability to generate new data and result in blurry outputs, the upscale ratio being very limited.

Other SR methods would be reconstruction based, that utilize a priori knowledge of successive low-resolution image frames of the same scene to solve an ill-posed inverse problem. The problem involves upsampling, deblurring and denoising to obtain a high-quality image. These algorithms fall into two categories: deterministic and stochastic. Deterministic algorithms encode knowledge of the high-resolution image as priors and use a constrained least squares method to regularize the solution. Stochastic methods treat SR reconstruction as a statistical estimation problem and have gained prominence since they provide a theoretical framework for the inclusion of a priori constraints necessary for a satisfactory solution of the ill-posed SR inverse problem. Stochastic methods can handle prior information and noise more explicitly, making them more natural for inclusion of

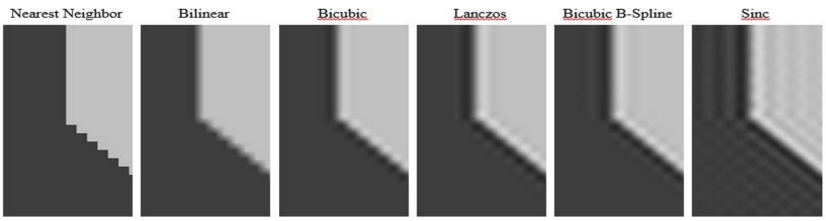

Figure 2.6: .Linear interpolation of a step edge: a balance between staircase artifacts and ripples[30]

prior knowledge.[12], A final item worth mentioning is the example based method, where the solution estimates the high resolution image by searching for patches from the input image into an image database. While this method results in far better results than the previous methods, it requires enormous computational power and time to complete a task. [12]

2.6.2 Deep Learning Super Resolution

While the methods for Super Resolution previously presented achieved good performance in certain scenarios, they are limited in many ways. Deep leaning can be used to perform the task of Super Resolution by teaching the neural network to map the low resolution image to the high resolution one. Some of the most popular deep learning methods for SR are using Generative Adversarial Networks[9], which typically consists of a generator attempting to upscale the image and a discriminator which distinguishes between the the generated high resolution image with the ground truth.

However, in this paper we will first present and critique the application of autoencoders for the task of Super Resolution. There are many ways in which autoencoders can perform SR and a very simple example is upscaling the input image using one of the traditional SR techniques and passing the resulting image to a Convolutional Autoencoder to perform small improvements. In this case, the autoencoder is not actually tasked with upscaling, but with improving the performance of traditional SR algorithms. One of the advantages of this method is the simplicity in implementation, because the encoder and decoder are mirrored and relatively simple in architecture, containing convolutions and perhaps some hidden fully connected layers. However, one of the disadvantages of this method is that the possible issues created by the traditional SR could translate into visual artifacts in the final reconstructed image.[38]

Another architecture that has gathered significant attention in the past few years is Nvidia Deep Learning Super Sampling (DLSS)[5], which delivers real-time image upscaling and enhancements in certain compatible video game titles on systems running with graphic cards in Nvidia's GeForce RTX line. The first version of DLSS was released in February 2019. DLSS 1.0 is mostly a spatial image upscaler, that required training with video footage from the target video games where it was meant to be used. It is composed of two convolutional autoencoder neural networks, one of which is responsible for improving overall image quality by performing spatial anti-aliasing and edge enhancement, using the current frame and it's motion vectors and essentially replacing traditional anti-aliasing procedures. The architecture of the upscaling autoencoder is a single low resolution image input that outputs the high resolution image. Due to the low mount

of input data, the upscaler can suffer from hallucinations and produce artifacts. As a consequence there was a mixed public response to the first version of the technology.

The second version of Nvidia DLSS was released in April 2020 and it brought exciting new improvements to the previous version. The most important improvement introduced was that the neural network could be integrated in any game without having to train it specifically with footage from that game. [8] DLSS 2.0 can be classified as a Temporal Anti-Aliasing Upsampling (TAAU) implementation and while the autoencoder architecture is not available to the public, some of the architecture of the system can be visualised in the Figure 2.7, released by Nvidia. Because the architecture switched from a single image SR to a multi-frames structure, the autoencoder no longer needs to generate new information, but can use previous frames as extra data.[8] This version of DLSS proved to be extremely successful, as it also introduced the option of not upscaling the image and using it to replace the traditional temporal anti-aliasing techniques, which proved to be better in some cases.[8]

On the 4th of January 2023, for a later update of the third version of DLSS, Nvidia announced the functionality of upscaling old or blurry YouTube videos in Chrome and Edge browsers. This represented the first step of the company towards using the technology for other purposes than video games.[35] With Nvidia's achievements in mind, in this paper we will attempt to implement and critique a convolutional autoencoder that attempts to resolve the problem of single image Super Resolution. We will consider multiple architectures, and implement a proof of concept that can contain not only convolutional layers, but fully connected layers as well.

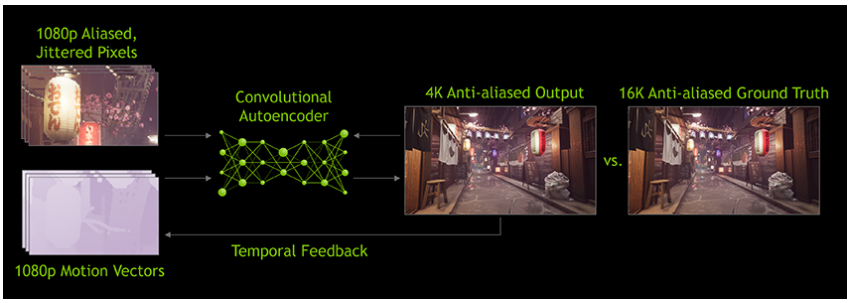

Figure 2.7: The NVIDIA DLSS 2.0 Architecture

2.7 Frame Interpolation

2.7.1 Conventional Frame Interpolation

Frame Interpolation (FI) is the process of generating in-between images from a sequence of images or frames. It is used extensively in the movie and animation industry because it aims to improve the general image quality of a video. Before autoencoders were introduced to this problem, FI could have been achieved with methods based on optical flow.[18] Optical flow methods were the first attempts at frame interpolation, where optical flow describes the apparent movement of pixel values in time-varying images caused by various factors. These methods estimate the motion vectors of each pixel or group of pixels by computing the differences between an initial image and its neighboring frames. The resulting image can be created by averaging the pixels in the input image

pointed by half of the obtained motion vectors. The most simplistic implementations of FI have limitations and require temporal smoothing to avoid aliasing caused by the significant differential between pixels. When the scene contains fast moving objects, the resulting image may generate overlapping objects, holes and other artifacts. This is a persistent issue in this kind of implementation because the algorithm does not have an understanding of the scene structure and it's objects.[18]

2.7.2 Deep Learning Frame Interpolation

Likewise to Super Resolution, Frame Interpolation is a process that can be improved with deep learning. To specifically address the shortcomings of traditional Frame Interpolation, deep learning methods can be applied under the form of a Variational Autoencoder (VAE). For image processing, VAEs typically take the form of encoder with convolutional and fully connected hidden layers and a mirrored decoder. The main alteration from traditional Autoencoders is the use of a probabilistic function instead of a deterministic function. This involves sampling the latent variable z from a continuous latent space defined by mean μ and standard deviation σ, which enhances the usefulness of VAEs for generative modeling. The μ vector determines the center of the input encoding, while σ determines the range of variation. By learning the data distribution instead of a single point, the decoder enables a broad spectrum of encoding for the same input during training.[10] The general architecture of a VAE can be visualised in Figure 2.8. When applied to the task of Frame Interpolation, simple VAEs such as the one presented in Figure 2.8 can fail to generate appropriate in-between images, however the architecture can be altered to achieve better results.[10]

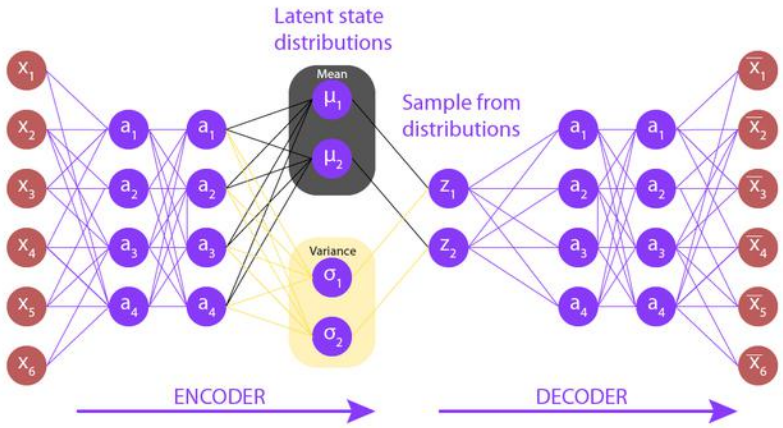

Figure 2.8: Variational Autoencoder [4]

A more popular piece of technology intended for generating frames in videos is Nvidia DLSS 3.0.[28] With this third iteration of DLSS, Nvidia has introduced Optical Multi Frame Generation to generate entirely new frames, which is compatible with GeForce RTX 40 Series GPUs and can greatly improve the performance in video games. To achieve this, Nvidia created an Autoencoder based model capable of taking 4 inputs (current and prior game frames, an optical flow field generated by Ada's Optical Flow Accelerator, and game engine data such as motion vectors and depth) to generate the new frame. The

feature is limited to RTX 40 Series GPUs because the process was greatly optimised by Ada's Optical Flow Accelerator, which runs on Ada Lovelace Architecture.[28]

In Figure 2.9 we can the observe the architecture of DLSS 3.0, which present the Super Resolution model and the Optical Multi Frame Generation model working together, with the later depending on SR. This combination of models can greatly improve performance in video games by simply using the low resolution input frame to generate more high resolution frames. The amount of data the system can generate from little input is quite impressive and can be visualised in Figure 2.10.[28]

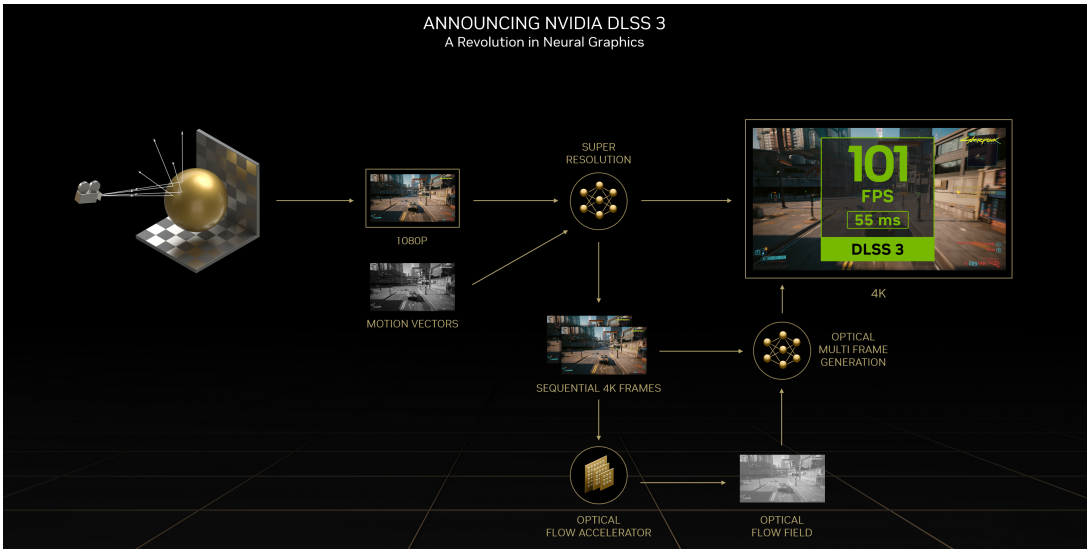

Figure 2.9: NVIDIA DLSS 3.0 [28]

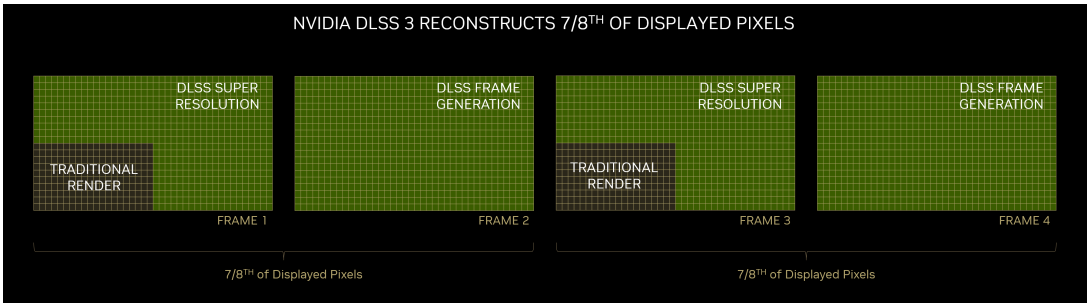

Figure 2.10: NVIDIA DLSS 3.0 throughput [28]

The results that Nvidia has achieved with DLSS 3.0 are undeniably impressive, however they are mostly limited to improving video game performance and quality. Because of the impressive results that Nvidia has achieved with DLSS 3.0 and because the fundamental concept powering it is an autoencoder, we will implement our own version of a convolutional autoencoder model specialised in generating frames in videos.

Chapter 3
Methodology

To make a demonstration for the knowledge presented in the previous chapter, we will implement proof of concept models for both the Super Resolution and Frame Interpolation tasks. These models are intended to showcase the capabilities of autoencoders in the given tasks and highlight any potential issues, so the implementations will not reach production level designs.

Over the course of this chapter we will need to analyse the image reconstruction performance. To correctly do this we will use multiple different metrics. The first metric will be the Mean Squared Error (MSE), which measures the average squared difference between the pixel values of the ground truth image and the reconstructed image. The lower the MSE value, the better the reconstruction quality. Another important metric will be Peak Signal-to-Noise Ratio (PSNR), which is a commonly used metric that measures the ratio of the maximum pixel value to the root mean squared error. The higher the PSNR value, the better the reconstruction quality. The third and final metric will be Structural Similarity Index (SSIM), which is a perceptual metric that measures the similarity between the structure of the ground truth image and the reconstructed image. The SSIM value ranges between -1 and 1, with 1 indicating perfect similarity. To calculate MSE we will be using `torch.nn.MSELoss()`[32]. To calculate PSNR we will be using the already calculated MSE and apply the formula

$$10log_{10}(\frac{MAX^2}{\sqrt{MSE}}) \tag{3.1}$$

, where MAX is defined by the highest pixel value in both original and reconstruction. SSIM can be calculated using ignite.metrics.SSIM[32].

3.1 Super Resolution

To begin, we will first address the problem of Super Resolution. We will implement, test and critique a convolutional autoencoder with fully connected hidden layers for the latent representation. This implementation is inspired by the first version of DLSS[5], deviating from the original concept with the fully connected layers to showcase the autoencoders capabilities of compressing data to a significantly lower dimension. To implement it we will make use of the PyTorch library[32]

3.1.1 Data Set

Choosing an appropriate data set for this task is crucial. We will be using the Modified National Institute of Standards and Technology (MNIST) database, provided by torchvision .datasets.MNIST[33]. MNIST provides images of handwritten numerical digits, with the images having a resolution of 28 by 28 pixels and only one channel for grey pixel values. This image data base will be a great fit for the autoencoder because of the image's structural simplicity that can enable the network to gain a latent space representation

understanding. Since we are attempting to upscale the images, we first require some changes to the default MNIST class in PyTorch. The default class has a method, `__getitem__(self, index: int)`, responsible for returning the item at a given index. The item in question is composed of the 28 by 28 image and its corresponding label. Since autoencoders do not usually require labels to their data, we will replace it with the lower resolution 20 by 20 image representation of the original image. To downscale the image, we have used PyTorch's function torchvision.transforms.Resize((20, 20)). With these changes, the torch.utils.data.DataLoader[32] data loader can now provide our model with pairs of source and ground truth images.

3.1.2 Autoencoder Architecture

As discussed in the Autoencoders section, our model will be comprised of an encoder and a decoder. To achieve Super Resolution we will need to modify the concept of the autoencoder, since the input is a lower resolution image and the output should be the higher resolution one. As we have seen there are many ways to perform this and since we are also interested in the latent space representation we will proceed with defining 2 autoencoders, one for low resolution and one for high resolution images, both using the same size of latent space. This way both autoencoders have mirrored implementations of the encoder an decoder and we can combine these models to achieve our task. We can train the low resolution encoder to compress the image in its latent space representation and use the high resolution decoder to reconstruct the high resolution image from the same latent space representation.

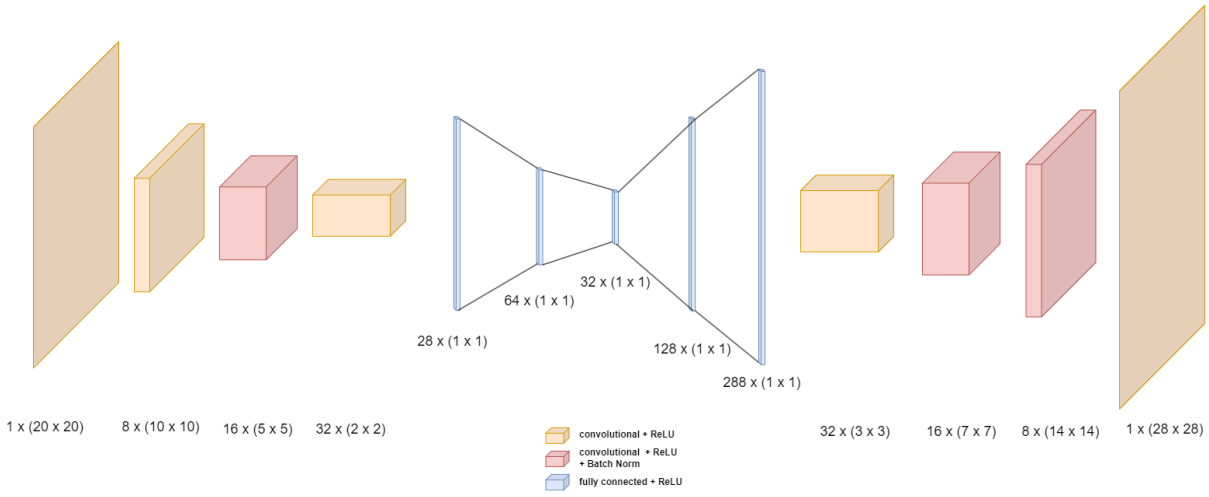

Figure 3.1: My Upscaling Autoencoder Architecture

The detailed architecture of the upscaling autoencoder can be visualised in Figure 3.1, where the left side of the latent space is the low resolution encoder and the right side is the high resolution decoder. As we can observe the first 3 layers are convolutional, increasing the number of channels from 1 to 32. These layers have ReLu activations and after the second convolution we also apply batch normalisation function. Following the convolutions, we have fully connected layers, so we need to flatten the data dimension from the structure [32, 2, 2] to a tensor with length 128. The 2 fully connected layers bring the input to the latent space representation of only 32 values. The decoder takes

this as an input and expands it to a tensor of length 288 using 2 fully connected layers with ReLU activation. The data is unflatten to a tensor with structure of [32, 3, 3] for the convolutional layers that attempt to upsample the data to higher resolution by reducing the number of channels back to 1. These transposed convolutional layer have ReLU activation and batch normalisation to bring the tensor structure to the final output of [1, 28, 28].

3.1.3 Training

The process of training the Super Resolution Autoencoder (SRAE) involves teaching the model the relationship between input data and the corresponding output values. This is achieved by presenting the model with a large set of input-output pairs, and adjusting the model's parameters to minimize the difference between the predicted output and the actual output. Training our SRAE will come under the form of epochs, where an epoch represents pushing all the training images thru the model once. In one epoch, we will send data in batches of no more than 256 images for better performance and after each batch the program needs to calculate MSE and SSIM. Additionally, after each batch we also adjust all the model parameters using the Adam Optimizer [13], an algorithm for first-order gradient-based optimization of stochastic objective functions, provided by PyTorch[32]. The optimizer requires a gradient, calculated by the loss function with respect to the model's parameters. Any metrics calculated during this stage are cached for later visualisation. For the optimizer we will use a learning rate of 0.001 with a weight decay of $1e-05$ and all operations will be executed on a CUDA[29] compatible device, NVIDIA GeForce RTX 3070 Laptop GPU.

3.1.4 Testing

Testing is an important step in developing the model and it will be executed after each training epoch. The process is fairly similar to the training step, however there are a few important distinctions. First difference is the fact that the input low resolution images have never been processed by the model before. Another important change is that the models are setup in evaluation mode, meaning the weights do not change after processing an image. The metrics and images obtained in this step will be used to analyse the models performance.

3.1.5 Results & Evaluation

We start training the model composed of the low resolution encoder and the high resolution decoder and after only one epoch we obtained the results in Figure 3.2, displaying the input and output images. The first problem we notice with the results are the extra high value pixels in the background of the numbers, distributed almost uniformly. These are caused by the convolutional layers. We also observe a decent result in the reconstruction of the digits, signifying that the fully connected layer are already grasping an understanding of the digit structures.

With only 9 extra epochs, the result are greatly improved, as shown in Figure 3.3. The background white pixels are greatly diminished and the digits themselves are less affected by this. Taking a closer look at the digit reconstructions, we observe great performance for most of them, with the exception of those with small loops in their

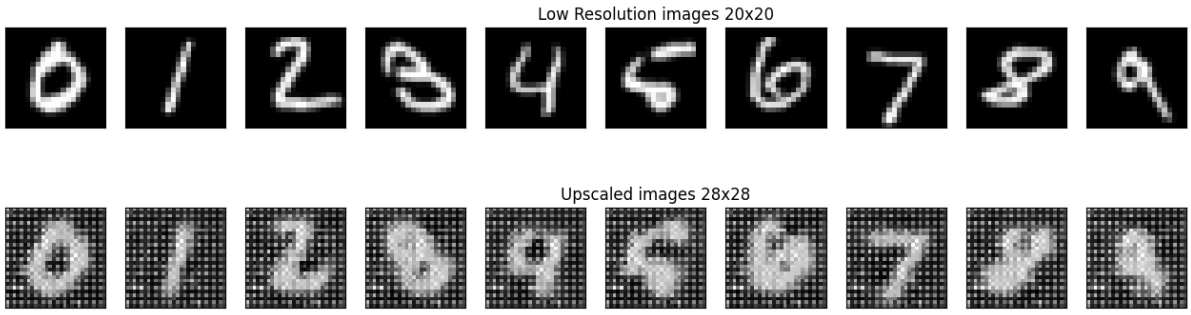

Figure 3.2: Test Results After 1 Epoch

graphical representation. To better understand the issue, Figure 3.4 shows the absolute difference in pixel values between the reconstructed high resolution image and the ground truth high resolution image. It shows us that the main issue with the reconstructions are the edges, however we also see clusters of high differences around the semi-loops in numbers 3 and 6. This can tell us that the model might have trouble with small details and confuse the closed loops of 8 and 9 with the semi-loops.

Figure 3.3: Test Results After 10 Epochs

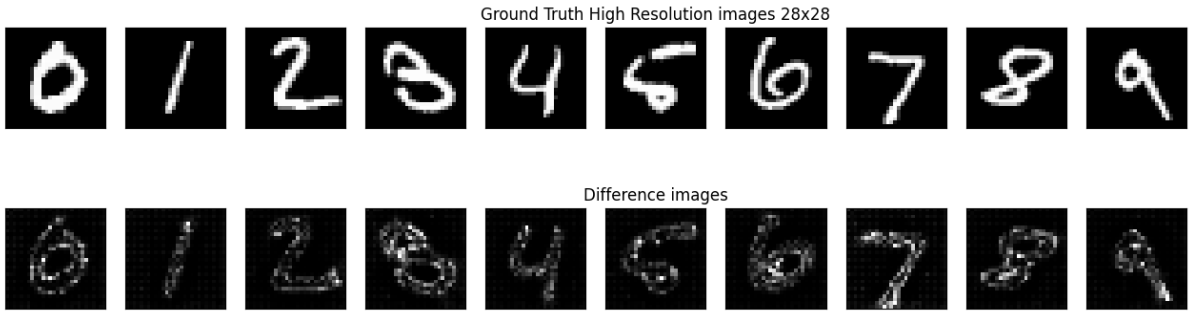

Figure 3.4: Test Results Difference After 10 Epochs

Considering the initial results we begin looking for a fitting number of epochs that can bring us great performance with as little training as possible. To find this value, we can train the model for 100 epochs to visualise the mean squared error after each epoch. Figure 3.5 displays the MSE values for training and testing over the course of 100 epochs.

We observe the initial significant drop in reconstruction loss that is also reflected by the results analysed above. Figure 3.6 displays the same values with a restricted range of 10 to 75 epochs. The most important observation we can make out of this graphic is the fact that the validation MSEs are lower than the training MSEs up the the 30th epoch. After that point, the model shows signs of overfitting, being behind the training MSEs but keeping the same downward trend. From this point we can conclude that an appropriate number of epochs would be either 30 for lower accuracy and less overfitting or around 50 epochs with better accuracy and little overfitting. This is also supported by Figures 3.7 and 3.8, which display the PSNR and SSIM values for epochs 10 to 75.

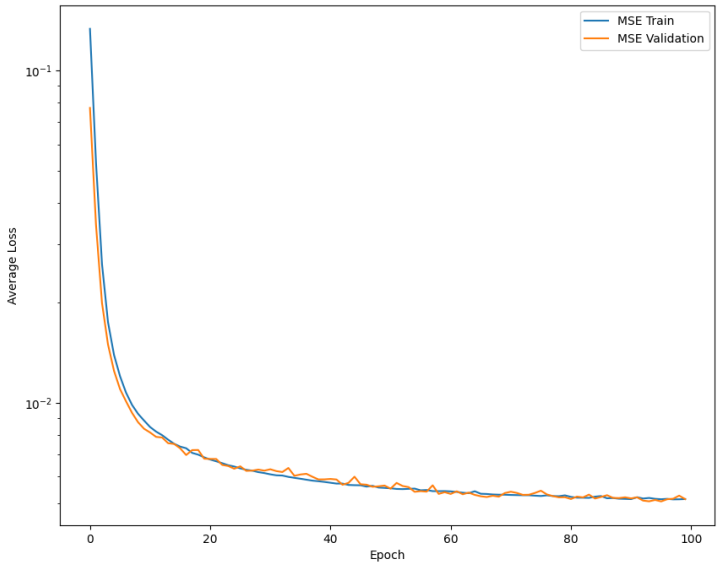

Figure 3.5: Upscaler MSE 100 epochs

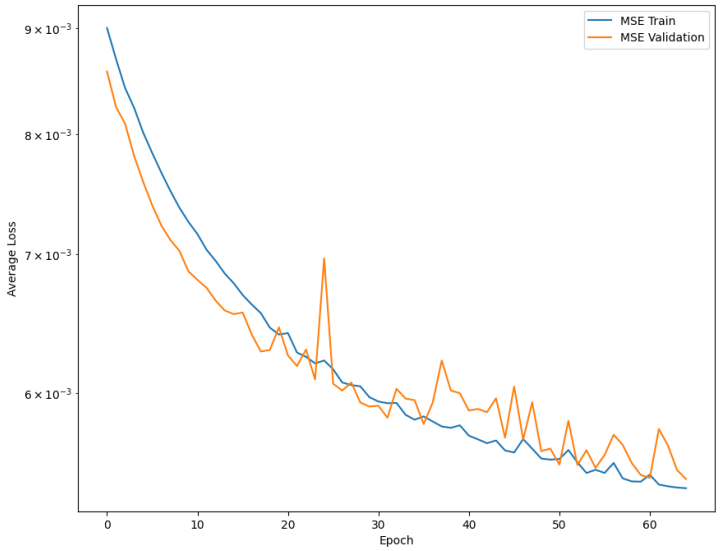

Figure 3.6: Upscaler MSE epochs 10 to 75

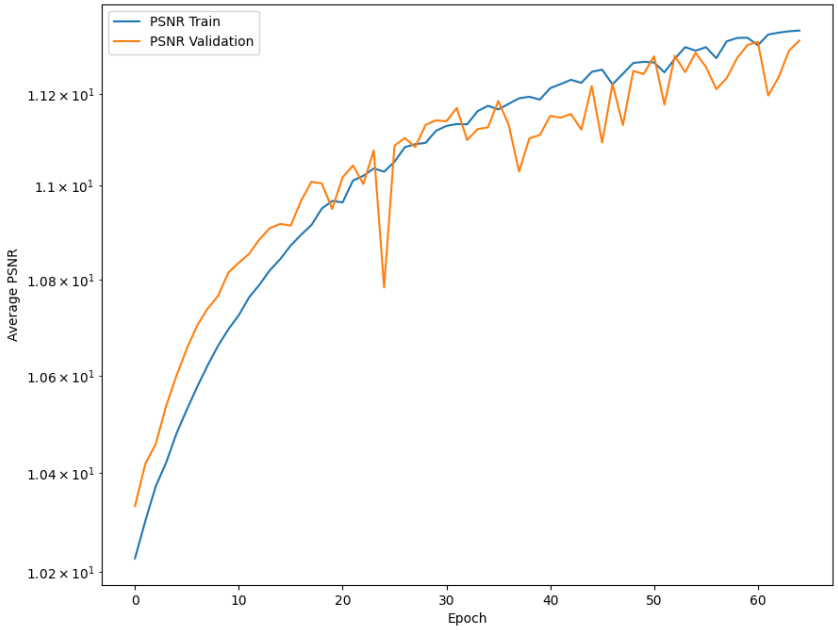

Figure 3.7: Upscaler PSNR epochs 10 to 75

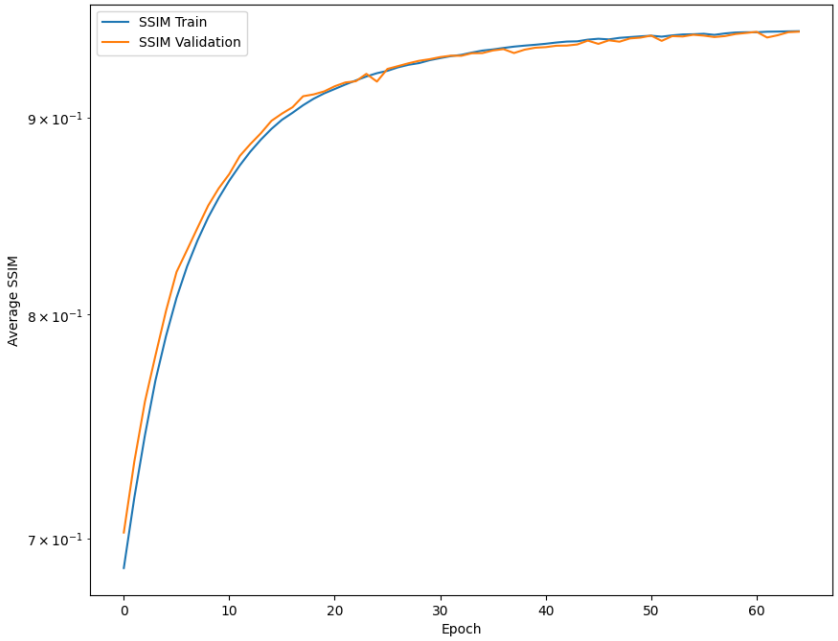

Figure 3.8: Upscaler SSIM epochs 10 to 75

We will now have a closer look at the results provided in epoch 30. Figure 3.9 shows the source low resolution images and their high resolution result at epoch 30. Figure 3.10 shows the ground truth high resolution image and the difference it has to the model generated image. One important distinction to the previous results at epoch 10 is the lack of white pixels in the background, this is mostly reflected by the difference images.

Figure 3.11 shows the source low resolution images and their high resolution result at epoch 50. Figure 3.12 shows the ground truth high resolution image and the difference

Figure 3.9: Test results at epoch 30

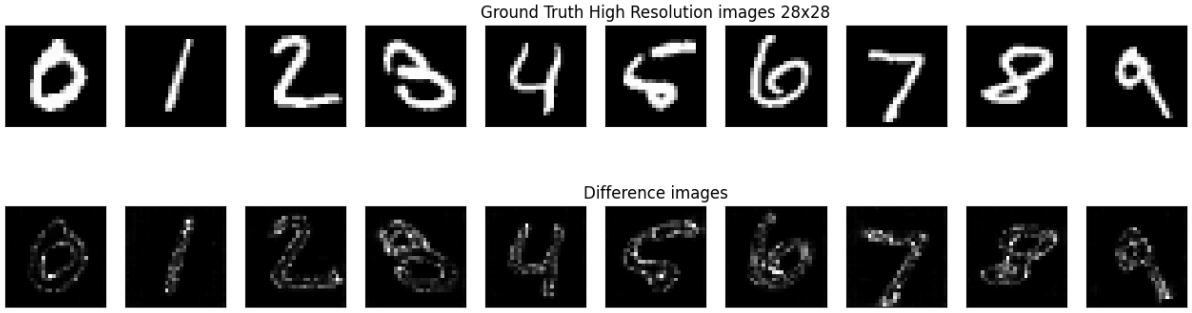

Figure 3.10: Test results difference at epoch 30

it has to the model generated image. Another improvement is that the center of the lines are no longer affected the same as they were before, where dark pixels would be introduced in the lines. Even if increasing the umber of epochs showed better results, a consistent issue that seems to not be affected as much by the number of epochs is the inaccuracy in edges. The issue might be caused simply by the lack of information in the source images. Down scaling the original 28 by 28 image automatically reduces the amount of information regarding edges, making them rougher.

Figure 3.11: Test results at epoch 50

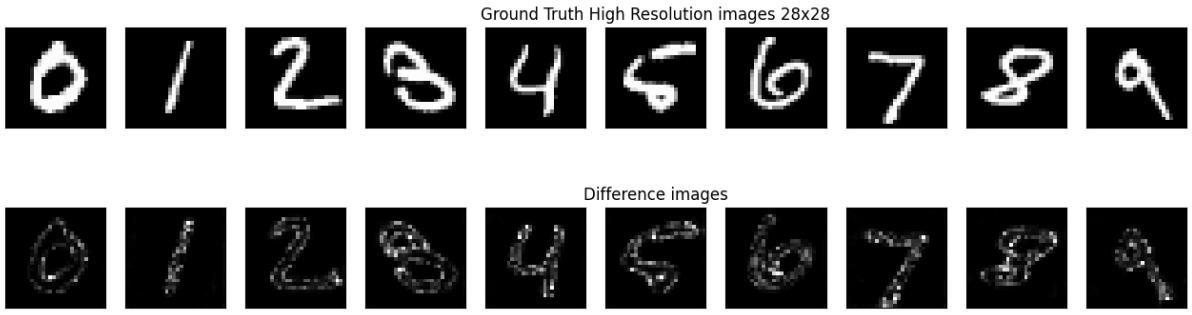

Figure 3.12: Test results difference at epoch 50

3.1.6 Critique and improvements

This implementation for Super Resolution is a great showcase for the capabilities of convolutional autoencoders, successfully compressing the low resolution images into tensors of length 32 and reconstructing decent higher resolution images. As we have seen the process is not without flaws, and one could argue that the latent space representation is to small to properly represent all the features in the source images.

The consistent issue that the autoencoder has for the closed and semi-closed loops in the images could be fixed by introducing the probabilistic approach of Variational Autoencoders (VAEs)[10]. Because of the VAEs disentanglement property[10], the latent space representation can better reflect the features of the input image and perhaps differentiate when a loop is closed or almost closed.

Another possible improvement for the autoencoder would be to change the image data set. Because the MNIST data set has very simplistic images we were able to introduce the fully connected layers that achieved great compression rate to only 32 tensor values. However, my attempts to adapt this autoencoder structure to images from the ImageNet[21] data set were unsuccessful. ImageNet provides higher resolution images than MNIST, meaning the autoencoder would have much more data to work it, but this could also introduce the problem that the encoded space dimension was too small to represent an image. Perhaps the probabilistic approach could help in this scenario, or the fully connected layers could be dropped in favour of more convolutional layers.

Nevertheless, the current implementation achieved it's purposes as a proof of concept application by discovering the potential issues we could face if an attempt was made to create a production level design of upscaling autoencoder.

3.2 Deep Learning Frame Interpolation

In this section we will address the problem of Frame Interpolation. There are many possible implementations possible for this, including making use of calculated motion vectors as we have seen in NVIDIA's DLSS. To demonstrate the autoencoders capabilities we will implement a double image input convolutional autoencoder, however we will not provide it with any additional data besides the 2 consecutive frames with the hopes that it is able to generate in between frames without any additional computations. In this scenario, the model will receive as little data as possible to emphasize its ability to generate new data and its understanding of the relation between the 2 input frames.

3.2.1 Data Set

Because of the problems nature, the data set will need to be formed of different video files. Most of the videos used in this research will be originating from Pixabay[2], posted by different contributors. However, video footage from my personal library will also be used, with a focus on ski videos for their very different and significantly more dynamic scenes, compared to the Pixabay videos. The ski footage also has 60 frames per second (FPS) while the Pixabay videos tend to have 25 FPS.

The autoencoder will be using video frames with a resolution of 360 by 640 pixels so all the collected videos need to be converted to the proper resolution.To achieve this, we have implemented a script using OpenCV[1] to read the video frames, resize them and save them to a new video file. It can also cut the video to a desired number of frames. Alternatively, software like VLC Media Player[3] also offer tools for resizing the video resolution, but not cutting the video in smaller parts. Unlike the previous data set used, MNIST, this new data set will be using 3 channels per frame instead of one to support all Red-Green-Blue (RGB) colours.

In our implementation, the autoencoder will need to be feed 2 consecutive frames and the ground truth in between image for error calculations and backpropagation. To satisfy these requirements we have implemented a custom dynamic Data Loader. The video file encodings do not permit reading a specific frame, as data can be dependant on previous frames. The frames will need to be read sequentially and packed into a list of 3 consecutive frames, where the last frame is repeated as the first in the following frame packet. Due to the large size of the files and the limited amount of memory, a video file is unlikely to fit in the RAM memory under the 3 frames structure. Therefore, the custom data loader will be reading from the video files dynamically when feeding data into the autoencoder. The data loader will have a frames queue that should always contain a number of batches, taking and adding items to it as requested by the model. This was designed to run the CPU bound process of reading the frames in parallel with the GPU bound process of training or testing the model for optimisation, however the implementation it is not concurrent yet. Similarly to the video preparing script, we will be using OpenCV[1] to read and write to video files.

3.2.2 Autoencoder Architecture

The autoencoder architecture tasked with Frame Interpolation is inspired by NVIDIA's DLSS, however we will be limiting the amount of data feed and pre-processing tasks as much possible. Compared to DLSS, my autoencoder will only receive 2 consecutive frames from real life footage instead of video games generated aliased jittered pixel frames and motion vectors. Because of the autoencoders composition of an encoder and a decoder, the architecture could take many forms. For example, an interesting implementation would be to define an encoder responsible for bringing a frame to a latent space representation while using the same neuron weights for both frames. The decoder could then be trained to use 2 latent space representations to generate the middle frame. Another type of possible structure is to calculate the motion vectors and feed them to the autoencoder along with one frame to generate the next frame.

The model will take the form of a double input convolutional autoencoder. The frames have rather large resolution compared to the SR implementation on MNIST, so we will not be using any fully connected layers. This is also because attempting to compress a higher quality detailed image in a small latent space representation would most likely

result in great loss of data or simply require too much processing power during training for my resources.

As such, the encoder will be composed of 6 convolutional layers, all followed by ReLu activation functions, with a identical structure of 3 sequential convolutions for each of the inputs. The first layer executes a convolution with stride 2, halving the image resolution and upsampling the number of channels from 3 to 128. The following 2 convolutions reduce the data both in resolution and channels as well, the second encoder convolution reducing the channels to 64 and the resolution to 90 by 180 pixels. The third and final layer reduces the channels to 16 and the resolution to 45 by 80 pixels. The reasoning behind these layer is to explode the data in many channels for feature extraction and then reduce the dimensions to have meaningful features extracted in the latent space representation. The parallel data generated by the left and right encoder components is concatenated on the channel dimension to be provided to the decoder.

The decoder takes as input the concatenated extracted features and returns the final in between frame. It achieves this by using 3 transposed convolutional layers, all followed by ReLu activation functions. The first layer actually reduces the channel dimension to mirror the final layer of the encoder at 16 channels and doubles the resolution to 90 by 160 pixels. The following transpose convolutional layer increases the same dimension to 64 and it up scales the resolutions to 180 by 320 pixels. The final layer performs the last resolution increase to the input resolution and reduces the channels back to 3 for RGB values.

Figure 3.13 displays the architecture of the Frame Interpolation Convolutional Autoencoder under the form of an UML flowchart. In my implementation, the bottom branch of the encoder receives the first frame while the top branch receives the second frame. The bottleneck of the autoencoder can be considered to be the concatenation function, which is not a neural model function but a PyTorch tensor manipulation function `torch.cat(( left, right), 0)`[32].

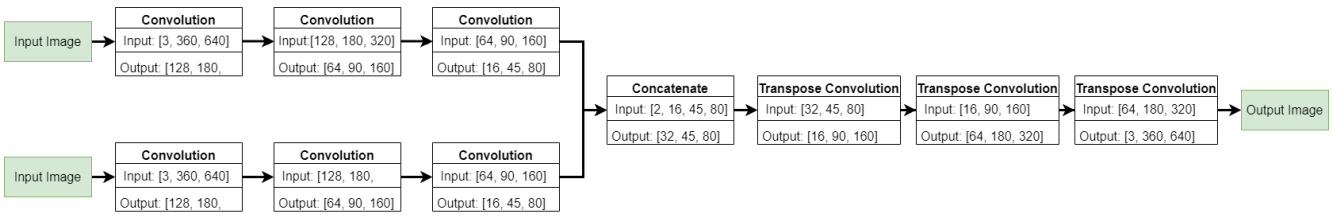

Figure 3.13: Frame Interpolation Convolutional Autoencoder Architecture

3.2.3 Training

The process of training the Frame Interpolation Convolutional Autoencoder consists of teaching the model to map the two input images to an intermediary image representing the in between frame of the inputs. The process is repeatable for a given number of epochs, in each one providing the autoencoder with the same videos. The videos used for training will be composed of only 2 videos obtained from Pixabay. The first video[11] for training is footage from a military exercise. The video is filmed mostly from fixed positions and has dynamic objects in the scene, like humans and helicopters. The second video[27] for training is a fixed position scene of people running and walking on a riverside

street in a city. Overall, the training process is fairly similar to the process for the SRAE, however, in this case I've decided to do backpropagation and error calculations after each generated frame. Considering the heavy load of reading frames from a video, batching the frames in large numbers would result in significantly longer training time as the video would have to be read many more times. If the data loader could be reading frames in parallel, batching some frames together could optimise the training time. Likewise to the SRAE, we gather metrics and adjust the model parameters using the Adam Optimizer[13] with the learning rate set to 1e-04 and the weight decay set to 1e-08.

3.2.4 Testing

To test this model we need to use one of the testing videos to gather metrics and produce at least 2 different video outputs. The outputs will consist of side by side comparisons of the original video and the FPS boosted video. Most of the tests will be run using a video collected from Pixabay[16] that has 25 FPS and shows some military exercises with moving camera angles and dynamic objects. One of the output videos will contain the source video at 12.5 FPS and the generated output video at 25 FPS with half frames generated by the model. The second video will contain the original source video at 25 FPS alongside the fps boosted video at 50 FPS, with half of the frames generated. Additionally, to test the model with a completely different scenery from the training data, we will apply this process to a ski video from my library. The ski video has 60 FPS and it is shot from the perspective of a skier, containing a white landscape with rocks, along with some dynamic movement from other skiers. This video is great for demonstrating the models ability to understand motion because it has different content and FPS rate.

3.2.5 Results & Evaluation

In order to obtain usable results, we first have to determine the appropriate amount of training. To do this, I've trained the model for 10 epochs and tested its performance after each one using the 25 FPS test video. In Figure 3.14 we show the average training and testing SSIM values. First item to observe in this figure is the fact that the testing SSIM values are always in proximity, suggesting that the model can be trained for many more epochs without encountering the overfitting problem. On the other hand, by looking at the actual SSIM values (under 0.9) we observe that it is very unlikely for this model to reach a point where the reconstruction is really close to the ground truth images and to understand why we will have a look at the actual outputs.

In Figure 3.15 we can see side by side the comparison between the first input frame to the model and its output frame while boosting the frame rate from 25 to 50 FPS. While the image might appear to be just blurry, taking a closer look at it and also at Figure 3.16 reveals the fact that the convolutional layers fail produce a uniform output and instead produce an artifact over the image that appears pixelated. In some of the earlier epochs of the model, the entire output image could have its colouring changed based on what is the most predominant colour in the image. In Figure 3.16 we notice the models performance when objects in the scene move with speed. Because the autoencoder is purely convolutional it has trouble understanding the movement of objects and results in it trying to reduce the error by overlapping the frames.

Moving away from the military video test, we will have a look at the models performance when it is presented with a higher frame rate video containing a completely different

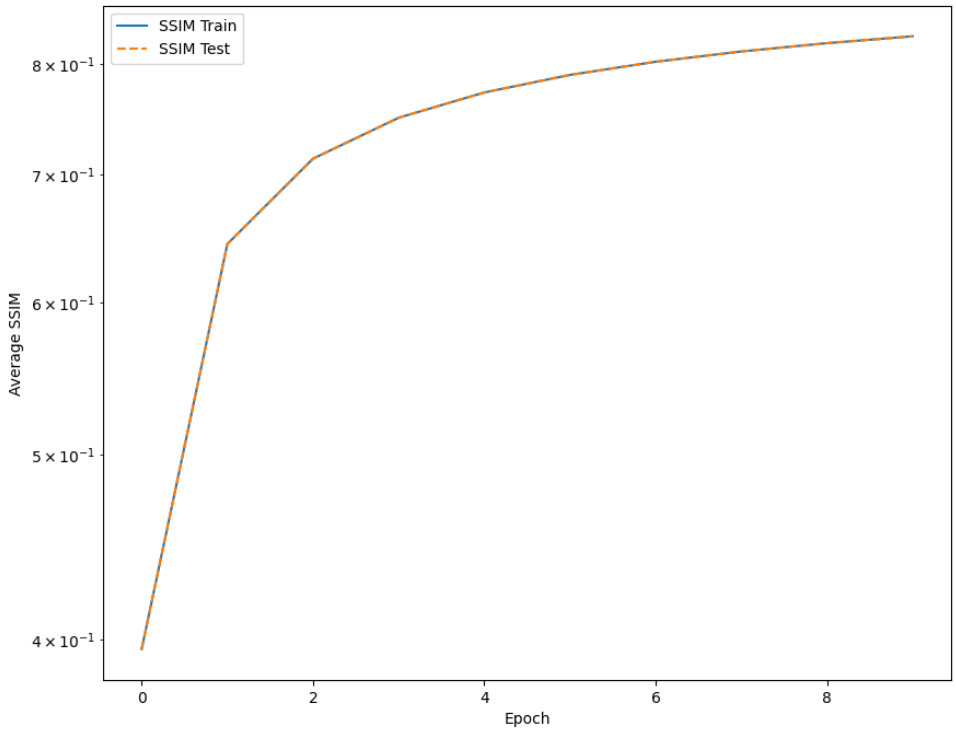

Figure 3.14: SSIM Train VS Test Military Video

Figure 3.15: First source image VS Generated frame 10 epochs Nato Video

scenery. The ski video is natively at 60 FPS but we are attempting to boost its frame rate from 30 to 60 FPS and use the extra ground truth frames to calculate metrics. For this test, the model was trained for only 5 epochs. As we can see in Figure 3.17, the pixelated artifact is much more visible, especially in the sky. Looking at the left bottom we also see a ghosting artifact for the hand, caused by its fast movement.

While the generated frames displayed beforehand show image artifacts, it is worth considering that videos are very dynamic and the scenery and movements can change or not happen at all. To reflect on this, we can have a look at Figures 3.18 and 3.19. The values in the graphs are the PSNR and SSIM indexes, reflecting that some portions of the video are easier to generate frames for than others. To provide some context, the first half of the video, the skier is in a fixed position. The downward spikes in the PSNR graph represent the moments when the camera changes the angle fast, essentially making

Figure 3.16: First source image VS Generated frame 10 epochs Nato Video

Figure 3.17: First source image VS Generated frame 5 epochs Ski Video

all the scenery move and produce errors in the output. In the second half, the skier is going downhill, hence why there are a loot of fluctuations in the PSNR values. As the skier comes to a stop, the PSNR value rises again and the SSIM stabilises.

When taking a look at the full resulting video, boosted from 30 to 60 FPS, we can observe a flickering of the image. This is caused by the same pixelated artifacts over the entire generated images. On a positive note, the ghosting artifacts are not noticeable when playing the video and yet the video does appear to be smoother in its motions. There are many ways in which the pixelated artifact can be fixed. For example, we could use a smoothing filter or even train another autoencoder model to fix these kind of issues. However, I believe that the best solution for this is to allow the autoencoder to use less processed data in the final layers from earlier layers. This can be achieved by using Skip Paths and next up we will define and use them in our Frame Interpolation Autoencoder (FIAE).

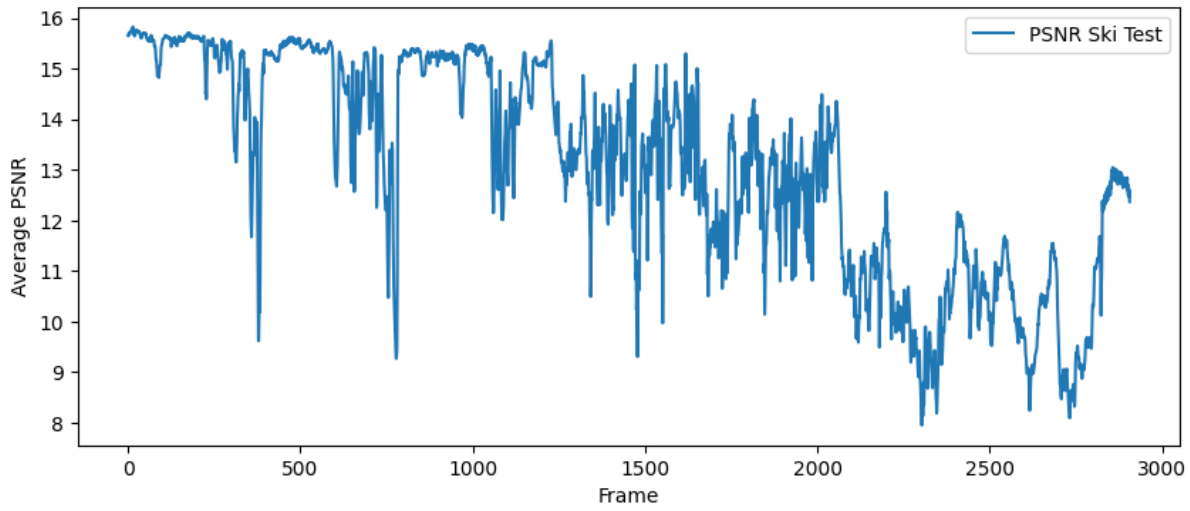

Figure 3.18: PSNR Ski Test

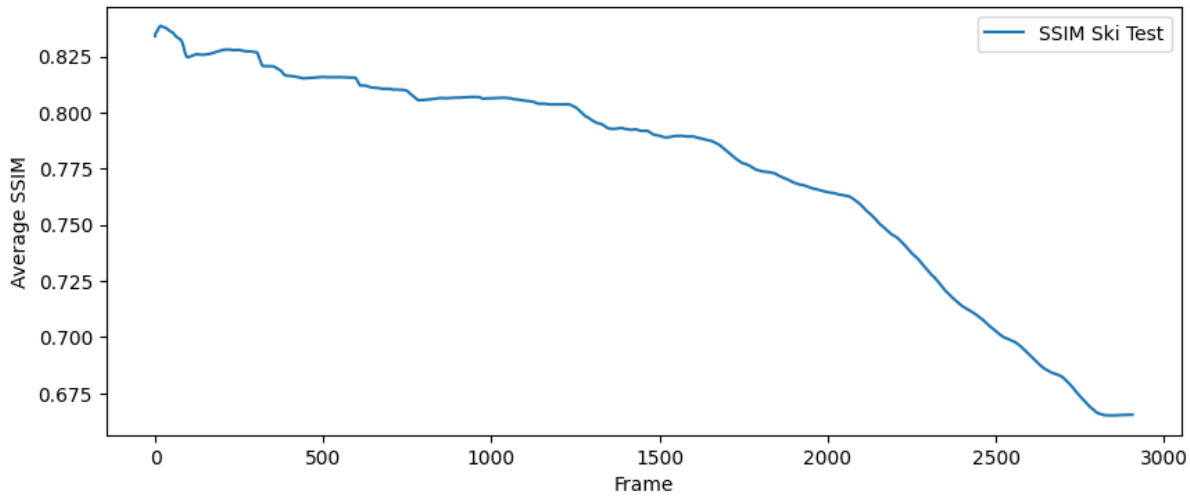

Figure 3.19: SSIM Ski Test

3.3 Deep Learning Frame Interpolation Version 2: Skip Paths

Skip Paths are connections between different layers of the autoencoder, allowing information to skip some of the processing and be introduced later in the process. Besides the benefits that it will bring to our model's accuracy, skip paths also increase the autoencoder's performance. The principal reason for introducing this concept is to resolve the pixelated artifact, however I'm also hopping that it will fix the issue seen in early epochs where a predominant colour could bleed into the rest of the scene. Because the information flowing thru the skip connections is less processed it should enable the autoencoder to return a better quality image.

The training and testing procedures will remain the same for this model as they were for the simple double input frame interpolation autoencoder.

3.3.1 Skip paths alternative architecture

The architecture of the autoencoder is affected by the skip paths in a limited way. In my model implementation I'm introducing connections between the first convolutional layer and the last one, as well as the second layer and second to last. Figure 3.20 shows the new Frame Interpolation Autoencoder architecture, with the encoder and decoder separated by a dotted line. The 128 channels outputs from the first parallel convolutions are first concatenated to reach 256 channels. This information set is connected to the last layer by concatenating it to the main information flow of the autoencoder using once again concatenation. The exact same process is applied to the second layer, this time generating a 128 channel information set.

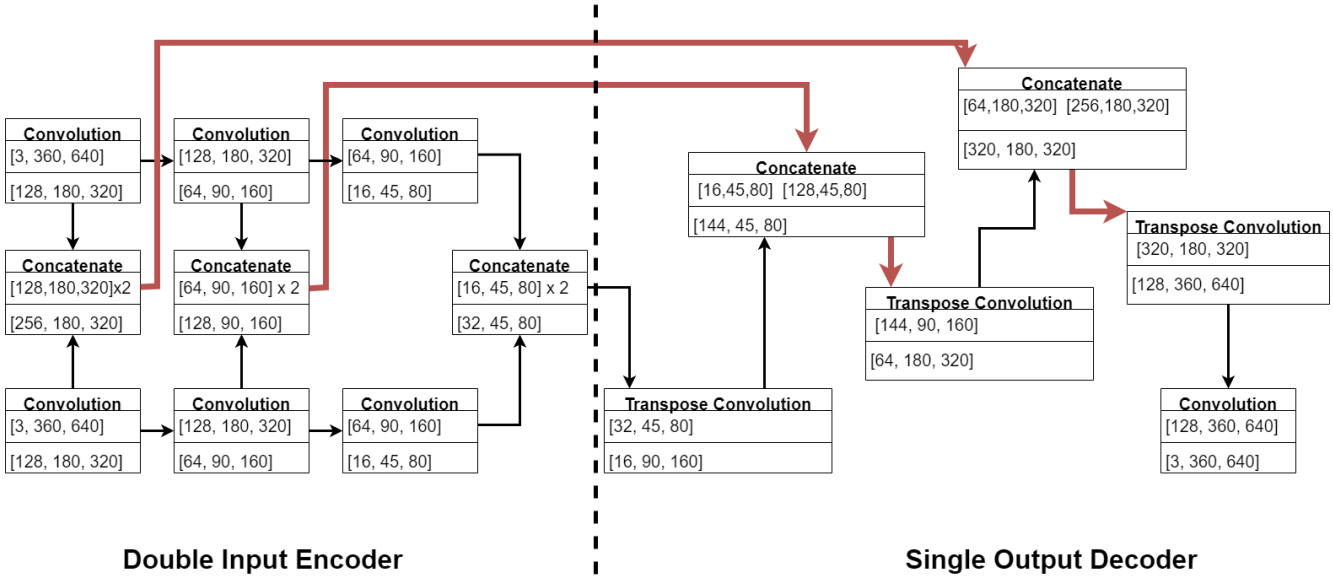

Figure 3.20: Frame Interpolation Convolutional Autoencoder Architecture With Skip Paths

3.3.2 Results & Evaluation

Similar to the first implementation, Figure 3.21 show the average SSIM values for training and testing over the period of 10 epochs. The first and most important item to note from this figure is that the values themselves are significantly higher, easily reaching 0.95 after only a few epochs. This tells us that the model generates better frames but we will have a more in depth look at it later on. The second item worth mentioning is the rate of improvement along the epochs, the improved model reaching a better value faster. This demonstrates that skip paths do indeed improve the model's performance. Likewise to the previous implementation, the model is not overfitted and one could motivate training it for more epochs.

Moving on to review some of the actual frames generated by the improved model, Figure 3.22 shows the same frames as the previous implementation. This time, however, the image is noticeably clearer, having the pixlelating artifact removed from the image. We also do not see any colour bleeding in this epoch (10) or earlier ones. Just from this image we can draw the conclusion that introducing skip paths did in fact allow the

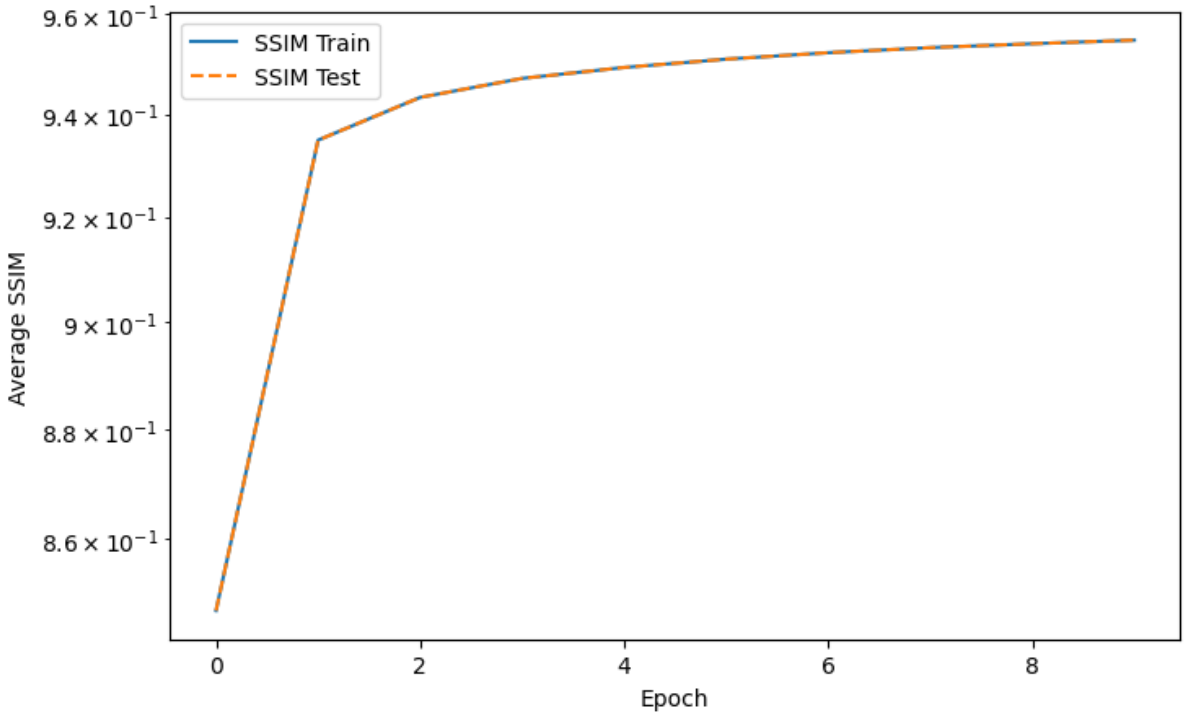

Figure 3.21: Skip Paths Average Train and Test SSIM 10 epochs

autoencoder to use less processed information to output a higher quality frame. On the other hand, in Figure 3.23 we can observe almost the same ghosting artifact when the man moves his head. This time, however, because the image is much clearer the ghosting effect is also slightly reduced. Compared to the previous implementation where the ghosting artifact could have unnatural pixel values, this time around the autoencoder executes a form of motion blur, properly overlapping the first and second frames.

Figure 3.22: Skip Paths, first source image VS Generated frame 10 epochs Nato Video

For the ski video test, the same procedure will be applied. In Figure 3.24 we can observe the exact same frame on the left side and on the right side the generated next frame. On a first look, likewise to the military video, the overall picture quality is greatly improved, showing sharper edges and less noise. Taking a closer look at the ghosting artifact on the skiers hand we also see a significant improvement. However, on the right side of the image, the shadow of the ski pole is clearly out of its place, once again

Figure 3.23: Skip Paths, first source image VS Generated frame 10 epochs Nato Video

Figure 3.24: First source image VS Generated frame 5 epochs Ski Video

demonstrating that fast moving objects are not handled with grace.

Likewise to the first version of FIAE, figures 3.25 and 3.26 show the PSNR and SSIM index values while boosting the ski test video from 30 to 60 FPS. The plots are extremely similar to the previous version, however, it is worth noting that both PSNR and SSIM have significantly higher initial values, with PSNR starting from almost 22 (16 in the previous version) and the SSIM starting from 0.95 (0.85 in the previous version). On the other hand, the PSNR index drops to the low value of 10 in the initial part of video, where the camera position is fixed while the angle changes. This is not far from the previous version, suggesting that while skip paths improved the overall quality they did not improve the accuracy in fast moving scenes.

From the SSIM metric graphic, we can understand that the model creates better frames when the camera position and angle are static. The decline of the SSIM metric, having a delta of approximately 0.05, is significantly less steep, while the previous version has a delta of approximately 0.2. This suggests that the updated model handles dynamic camera scenes far better.

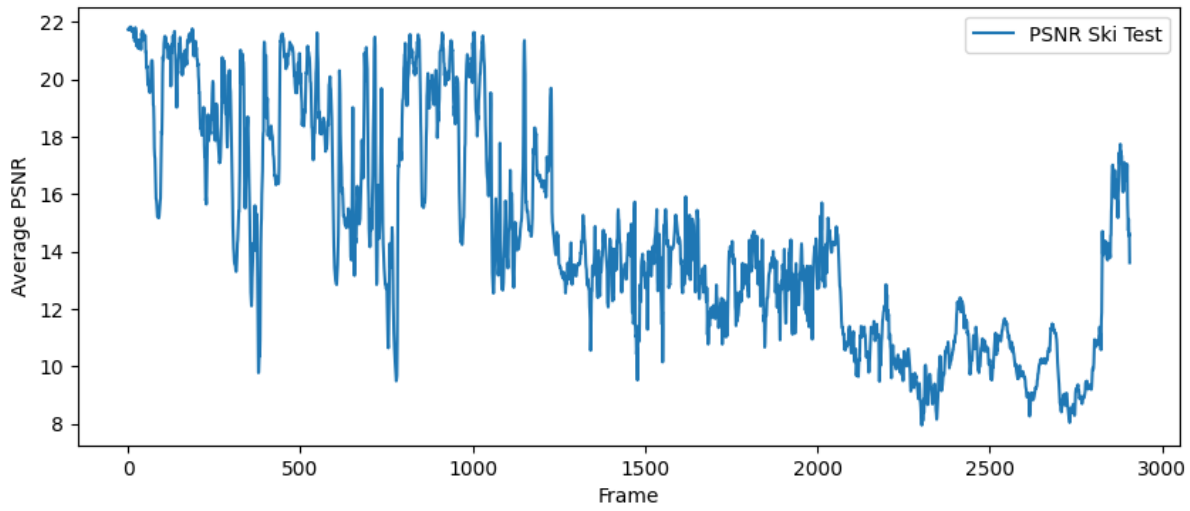

Figure 3.25: PSNR Ski Test FIAE2

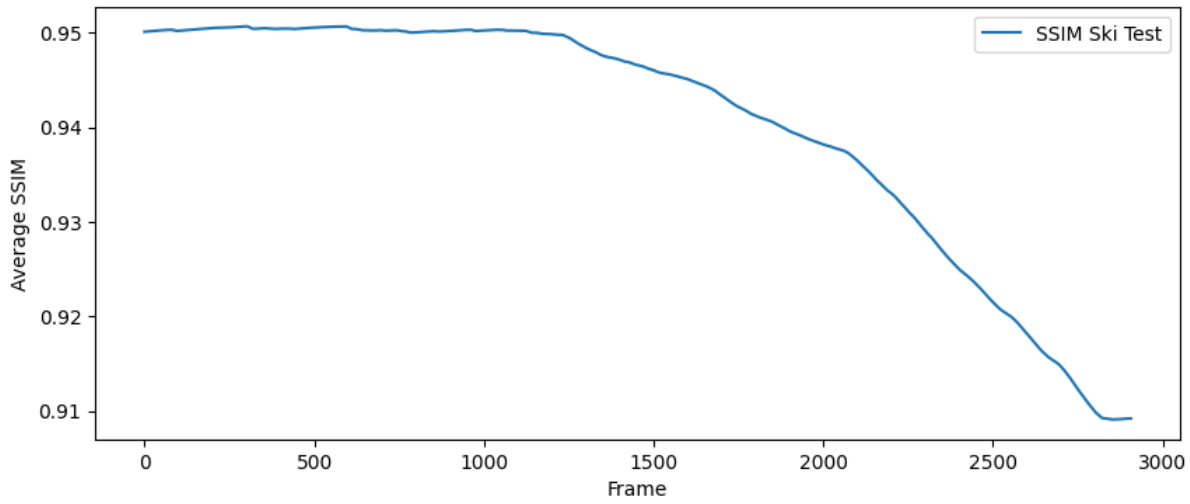

Figure 3.26: SSIM Ski Test FIAE2

3.3.3 Critique and assessment

Everything that has been presented for the task of Frame Interpolation would suggest that the model can successfully boost the frame rate of any video. While this is true, it is required to do an assessment of the final FPS boosted videos where I would analyze the effects that the extra frames have on the user experience. Unfortunately, the effects on user experience when it comes to video quality are highly subjective and sometimes even controversial. It is generally considered that the human eye can only see between 30 and 60 frames per second, however, many people with performant monitors can argue that an increased FPS does bring better user experience. For the scenario where the user wants to have a video at high frame rates, my model can work exceptionally well for boosting FPS (for example 60 FPS to 120FPS). For the scenario where the original video has a noticeably low frame rate and the user wants to increase it (for example from 12 to 24 FPS), my model would improve the overall quality, but at the risk of visible

ghosting artifacts when fast moving objects are in the scene. Besides doubling the frame rate of a video, the model actually allows the video to be boosted as much as wanted. An experiment has been produced where the final FPS boosted video would have 75% of its frames generated by the model. This can be achieved by first generating the middle frame between two original frames, and then applying the same process but using the generated frame as a source frame. The result was not great for low FPS videos and it definitely made the artifacts more visible, however, it is a great demonstration of the models capabilities to generate a significant amount of data. If the ghosting artifact would not be so visible, the 75% FPS boost could prove to be amazing, so I will now discuss some possible improvement to resolve that.

I believe that one of the ways to fix the ghosting artifact seen in both models is to provide the model with the motion vectors of the 2 sequential frames. As it is at the moment, our FIAEs are more focused on generating a high quality image rather than understanding the relationship between moving objects positions in the scene. By providing it with motion vectors, the model could better focus on image quality while using the extra information to resolve conflicting differences between the input frames. This is also supported by the fact that Nvidia's DLSS also uses motion vectors and does not suffer from ghosting artifacts.

Another option resolve this artifact would be to introduce the probabilistic aspect of Variational Autoencoders with the risk of loosing image quality due to the reduced latent space representation. From the architectures of the FIAEs, we can observe that they are not focused on compressing the data to a latent form, especially when we introduced skip paths which are contradictory to the compression process. A variational autoencoder with motion vectors, could be a perfect combination to maintain image quality without ghosting artifacts and to also maintain the concept of data compression. This, however, is outside of the scope of this project, since running such a model would require significant computational power which I do not have available. The increased computational power required is suggested by the results in the Super Resolution Autoencoder, which had low dimension input and no extra data such as motion vectors but still required significant processing power. To put this into context, DLSS is trained on Nvidia's Saturn V supercomputer[5].

Chapter 4
Conclusion and further work

In the last chapter of the this paper we will reflect on the accomplished achievements and provide a short summary of the supporting elements that made this possible. We will also have a discussion regarding the possible improvements and further work.

4.1 Summary and Achievements

Over the course of this project there were a multitude of topics presented. To ensure understanding of the concepts powering the implemented models, the second chapters begins by presenting the Deep Learning field as a whole, along with some of its classifications of models. With Deep Learning as a starting point, this paper presents the foundational elements of most modern artificial neural networks. In a gradual way, we introduce more complex concepts that use the building block of Perceptron, presenting the MLP and Convolutional Neural Networks, both of which are essential in the later work. With a base of knowledge established, we introduce the most important concept in this paper, the Autoencoder. An overview is provided along with the formal definition. Some applications of autoencoders are also presented to provide some context to its abilities. Afterwards, we introduce the main objectives of this paper, Super Resolution and Frame Interpolation. Both of them are conceptually presented and followed by conventionals solutions. The conventional methods stand to demonstrate the complexity of the problem with traditional methods and emphasise the ease of adaptability to different problems that Autoencoders posses. While Nvidia decided to keep its technology proprietary and private, an overview of the available information is provided to motivate our model designs.

The implemented model solutions are then introduced and evaluated in great depth. For each of the tasks we present the proposed neural network architecture as well as its development environment(data sets, libraries used). Evaluating the models came under different formats, depending on the task, presenting a critique of the single image reconstructions and an analysis of the produced videos. Over the course of this project, a few important objectives have been achieved. First of all, the Background section provided a great amount of information which can and has been used to understand discovered issues or to propose better implementations. As a direct technological output of this project, we have generated 3 different models that accomplish their task in a more than sufficient manner and serve as proof of concept for future implementations, when the hardware available will no be considered insufficient.

4.2 Reflection and Future Work

While the model for Super Resolution is limited to a set of handwritten digits, and thus it is essentially a proof of concept, the rate of compression is impressive, reducing the dimension from 400 and even 784 (for the 28 by 28 encoder) different pixel values to only a vector of 32 features (values). The compression rate is more than promising

and even comparable with other lossy compression algorithms. As mentioned in the evaluation of the model, I believe that Variational Autoencoders could greatly improve the performance. Additionally, the model could be adapted to larger file sizes by also increasing the latent space. Even a large latent space with thousands of feature vectors could provide a good compression rate, since modern images used on the web are usually in higher definition. Deviating from the original scope of this model, a great idea for future work for Super Resolution would be to adapt the autoencoder architecture to accept multiple video frames to better upscale videos as well. The significant rise in data sent to the autoencoder can have a great impact its performance, as we have seen with DLSS.

Compared to SR, the Frame Interpolation models can be considered more than proof of concept. Even if they are not focused on compression, the second second version provides great performance and it successfully generates in between frames without negatively affecting the source video. The only issue remaining is the ghosting artifact, however even with this the resulting videos are not negatively affected. To improve its performance and accuracy, as it was mentioned in the evaluation, there are multiple options available. Introducing motion vectors as inputs to the model could help it resolve the moving objects conflicts and mitigate the resulting artifacts by removing the necessity to understand how the scene changed and instead focus on image reconstruction. Additionally, introducing a latent space representation and probability by migrating to a Variational Autoencoder could reintroduce the concept of compression, however it could also introduce more artifacts or reduce the overall quality image.

For both SR and FI, the high level scope is to bring to light the possible optimisations that we could have for our daily internet content consumption. Companies that serve users with large amount of digital imagery and videos could greatly benefit from a deep learning based compression algorithm. For example, YouTube might reach a point in the future where holding all of its video library becomes to expensive to maintain and it could result in them deleting data or introducing new compression algorithms based on deep learning techniques, like autoencoders. Besides the cost of maintaining data, transmission cost is also an important factor, since the data centers spend vast amounts of resources just to serve data to users. This could be improved by compressing the data before transmission. I personally believe that the kind of technologies described in this paper will become commercialised for our benefit in a few short years, reducing the energy consumption for our entertainment and perhaps even improving the overall experience.

Bibliography

[1] Opencv. *https://opencv.org/*.

[2] Pixabay: Stunning free stock video footage & clips. *https://pixabay.com/*.

[3] Vlc media player. *https://www.videolan.org/*.

[4] Variational autoencoders. *https://www.geeksforgeeks.org/variational-autoencoders/*, 2022.

[5] Anjul Patney NVIDIA Andrew Edelsten, Paula Jukarainen. Truly next-gen: Adding deep learning to games & graphics (presented by nvidia). *https://www.gdcvault.com/play/1026184/Truly-Next-Gen-Adding-Deep*, 2019.

[6] Anastasios Doulamis Athanasios Voulodimos, Nikolaos Doulamis and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *https://doi.org/10.1155/2018/7068349*, 2018.

[7] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. *http://proceedings.mlr.press/v27/baldi12a.html*, 27:37–49, 2012.

[8] Andrew Burnes. Nvidia dlss 2.0: A big leap in ai rendering. *https://www.nvidia.com/en-us/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering/*, 2020.

[9] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

[10] Paulino Cristovao, Hidemoto Nakada, Yusuke Tanimura, and Hideki Asoh. Generating in-between images through learned latent space representation using variational autoencoders. *IEEE Access*, 8:149456–149467, 2020.

[11] Defence-Imagery. Soldiers military army helicopter. *https://pixabay.com/videos/soldiers-military-army-helicopter-446/*.

[12] Anand Deshpande and Prashant Patavardhan. Survey of super resolution techniques. *ICTACT Journal on Image and Video Processing*, 9, 02 2019.

[13] Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. *https://arxiv.org/abs/1412.6980*, 2014.

[14] Raja Giryes Dor Bank, Noam Koenigstein. Autoencoders. *https://arxiv.org/abs/2003.05991*, 2020.

[15] Shreyas Fadnavis. Image interpolation techniques in digital image processing: An overview. *International Journal Of Engineering Research and Application*, 4:2248–962270, 11 2014.

[16] FlickrVideos. Nato military war. *https://pixabay.com/videos/nato-military-war-army-spearhead-375/*.

[17] J.F. Mas H. Taud. Multilayer perceptron (mlp). *Geomatic Approaches for Modeling Land Change Scenarios* *https://doi.org/10.1007/978-3-319-60801-3_27*, pages 451–455, 2018.

[18] Scherzer O. Hinterberger, W. Models for image interpolation based on the optical flow. *Computing*, 66(3):231–231, 2001.

[19] Joseph Barfett Errol Colak Shahrokh Valaee Hojjat Salehinejad, Sharan Sankar. Recent advances in recurrent neural networks. *https://arxiv.org/abs/1801.01078*, 2017.

[20] Zheng Hu, Jiaojiao Zhang, and Yun Ge. Handling vanishing gradient problem using artificial derivative. *IEEE Access* *https://doi.org/10.1109/ACCESS.2021.3054915*, 9:22371–22377, 2021.

[21] ImageNet. About imagenet. *https://www.image-net.org/about.php*, 2014.

[22] Christian Ioffe, Sergey; Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *https://arxiv.org/abs/1502.03167*, 2015.

[23] Eero P. Simoncelli Johannes Ballé, Valero Laparra. End-to-end optimized image compression. *https://arxiv.org/abs/1611.01704*, 2016.

[24] Laveen N. Kanal. Perceptron. *https://dl.acm.org/doi/abs/10.5555/1074100.1074686*, 2003.

[25] Ryan Nash Keiron O'Shea. An introduction to convolutional neural networks. *https://arxiv.org/abs/1511.08458*, 2015.

[26] Yuxi Li. Deep reinforcement learning: An overview. *https://arxiv.org/abs/1701.07274*, 2017.

[27] Life-Of-Vids. Running people sports. *https://pixabay.com/videos/running-people-sports-run-walk-294/*.

[28] Henry C Lin and Andrew Burnes. Nvidia dlss 3: Ai-powered performance multiplier boosts frame rates by up to 4x. *https://www.nvidia.com/en-gb/geforce/news/dlss3-ai-powered-neural-graphics-innovations/*, 2022.

[29] NVIDIA. Cuda toolkit. *https://developer.nvidia.com/cuda-toolkit*, 2014.

[30] Pankaj Parsania and Dr V.Virparia. A review: Image interpolation techniques for image scaling. *International Journal of Innovative Research in Computer and Communication Engineering*, 02:7409–7414, 01 2015.

[31] Victor Podlozhnyuk. Image convolution with cuda. *https://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/convolutionSeparable/doc/convolutionSeparable.pdf*, 2007.

[32] PyTorch. Pytorch documentation. *https://pytorch.org/docs/stable/index.html*, 2023.

[33] PyTorch. Pytorch mnist. *https://pytorch.org/docs/stable/index.html*, 2023.

[34] Andrew Ilyas Aleksander Madry Shibani Santurkar, Dimitris Tsipras. How does batch normalization help optimization? *https://arxiv.org/abs/1805.11604*, 2019.

[35] NVIDIA Studio. Introducing rtx video super resolution - 4k ai upscaling for chrome & edge video. *https://www.youtube.com/watch?v=XA-tQpQqD7U&ab_channel=NVIDIAStudio*, 2022.

[36] Daniel Svozil, Vladimír Kvasnicka, and Jiří Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and Intelligent Laboratory Systems* *https://www.sciencedirect.com/science/article/pii/S0169743997000610*, 39(1):43–62, 1997.

[37] Francesco Visin Vincent Dumoulin. A guide to convolution arithmetic for deep learning. *https://arxiv.org/abs/1603.07285*, 2016.

[38] Wenming Yang, Xuechen Zhang, Yapeng Tian, Wei Wang, Jing-Hao Xue, and Qingmin Liao. Deep learning for single image super-resolution: A brief review. *IEEE Transactions on Multimedia*, 21(12):3106–3121, 2019.

[39] Yimin Yang, Q. M. Jonathan Wu, and Yaonan Wang. Autoencoder with invertible functions for dimension reduction and image reconstruction. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(7):1065–1079, 2018.

[40] Yoshua Bengio Geoffrey Hinton Yann LeCun. Deep learning. *https://doi.org/10.1038/nature14539*, 2015.

[41] Yuval Elovici Asaf Shabtai Yisroel Mirsky, Tomer Doitshman. Kitsune: An ensemble of autoencoders for online network intrusion detection. *https://arxiv.org/abs/1802.09089*, 2018.

[42] Jinming Zou, Yi Han, and Sung-Sau So. Overview of artificial neural networks. *https://doi.org/10.1007/978-1-60327-101-1_2*, 2009.

List of Figures