

Faculty of Technology, Design and Environment
Oxford Brookes University
School of Engineering Computing and Mathematics

BSc (Single Honours) Degree Project

Programme Name: **BSc (Hons) Computer Science**

Module No.: **COMP6013**

Surname: **Stout**

First Name: **Harry**

Project Title: **A Neural Network-Based Approach to Video Frame Interpolation**

Student No: **19044035**

Supervisor: **Dr Alexander Rast**

2TM Supervisor: **N/A**

(if applicable)

Date submitted: **18/03/2022**

*A report submitted as part of the requirements for the degree of BSc (Hons) in
Computer Science*

At

Oxford Brookes University

Student Conduct Regulations:

Please ensure you are familiar with the regulations in relation to Academic Integrity. The University takes this issue very seriously and students have been expelled or had their degrees withheld for cheating in assessment. It is important that students having difficulties with their work should seek help from their tutors rather than be tempted to use unfair means to gain marks. Students should not risk losing their degree and undermining all the work they have done towards it. You are expected to have familiarised yourself with these regulations.

<https://www.brookes.ac.uk/regulations/current/appeals-complaints-and-conduct/c1-1/>

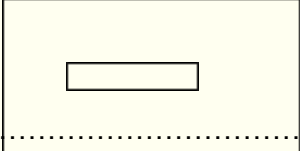
Guidance on the correct use of references can be found on www.brookes.ac.uk/services/library, and also in a handout in the Library.

The full regulations may be accessed on-line at

<https://www.brookes.ac.uk/students/sirt/student-conduct/>

If you do not understand what any of these terms mean, you should ask your Project Supervisor to clarify them for you.

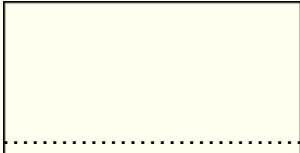
I declare that I have read and understood Regulations C1.1.4 of the Regulations governing Academic Misconduct, and that the work I submit is fully in accordance with them.

Signature  Date 18/03/202
2

REGULATIONS GOVERNING THE DEPOSIT AND USE OF OXFORD BROOKES UNIVERSITY MODULAR PROGRAMME PROJECTS AND DISSERTATIONS

Copies of projects/dissertations, submitted in fulfilment of Modular Programme requirements and achieving marks of 60% or above, shall normally be kept by the Library.

I agree that this dissertation may be available for reading and photocopying in accordance with the Regulations governing use of the Library.

Signature  Date 18/03/202
2

Acknowledgements:

I would like to thank my supervisor, Dr Alexander Rast, for his invaluable help over the course of the entire project during our weekly meetings by providing useful critique, his continued encouragement and for pointing me in the correct direction when it came to research.

I would also like to thank the production company Ligne de Front (<https://www.lignedefront.fr/>) for kindly allowing me to use their film “Megacity Mumbai” as both a strongly performing dataset and a source of screenshots for demonstrating the results of the project.

A Neural Network-Based Approach to Video Frame Interpolation

Harry Stout

19044035@brookes.ac.uk

1. Introduction	1
1.1 Background	1
1.2 Aim	2
1.3 Objectives	2
1.4 Product Overview	3
1.4.1 <i>Scope</i>	3
1.4.2 <i>Audience</i>	3
2. Introduction to Convolutional Neural Networks	4
3. Background Review	5
4. Methodology	6
4.1 Development Process	6
4.1.1 <i>Approach</i>	6
4.1.2 <i>Requirements</i>	7
4.1.3 <i>Design</i>	8
4.2 Technology	16
4.2.1 <i>Software</i>	16
4.2.2 <i>Hardware</i>	16
4.3 Version Management Plan	17
4.4 Use Case	17
5. Implementation and Results	17
5.1 Prototype Experimentation	17
5.2 Final Product Testing	20
6. Professional Issues and Risk	24
6.1 Risk	24
6.2 Professional Issues	27
6.2.1 <i>Legal Issues</i>	27
6.2.2 <i>Social and Ethical Issues</i>	28
6.2.3 <i>Environmental Issues</i>	28
7. Conclusion and Further Work	28
8. Bibliography	30
9. Appendices	31
9.1 Project Log	31
9.2 Model Summary	39
9.3 Resource Links	41

Abstract

Video Frame Interpolation aims to increase the frame rate of video data, thereby increasing the perceived smoothness of the video to the human eye. This is done by taking information from two sequential frames and determining what the middle frame should be based on this information. This frame is then inserted (interpolated) between the two chosen video frames. Most modern television sets are shipped with this functionality built in, however, the algorithm used is based on determining the motion of pixels in the video frame and calculating a middle point (Motion Estimation and Motion Compensation). This method can cause 'ghosting' artifacts in the resultant frame when objects in the frame move behind other objects (occlusion) which require resource intensive post-processing techniques to rectify. Research is being made around employing Convolutional Neural Networks (CNNs) to tackle the task of predicting frames for interpolation, which would solve the issue of requiring post-processing techniques to eliminate artifacts such as 'ghosting' and generally produces favourable results exceeding that of regular methods not using neural networks. However, predicting frames using CNNs takes a considerable amount of processing power, Video RAM, and time to produce a single image, which scales as the size of the input to the network increases. The following project implements a CNN for experimentation on data gathered from movies starring Charlie Chaplin, which are filmed at very low frame rates. The network will increase the temporal resolution of the video data and uses the Structural Similarity Index (SSIM) to determine how accurately the network predicts each interpolated frame.

1. Introduction

1.1 Background

The illusion of motion from viewing separate images occurs somewhere between a minimum of 10 to 12 frames per second. Any slower and the human eye can discern between static images. In the early era of cinema, movies were filmed at around 16 frames per second (this would vary as the cameras were hand-cranked) and were shown at 24 frames per second, which became an industry standard (Kemp, 2019). Today, modern digital devices are capable of filming and showing video files with frame rates reaching 120Hz and above (and even higher for state-of-the-art slow-motion cameras). However, by harnessing the power of machine learning, the structure of the non-existent frames can be predicted, constructed, and inserted into the original film so that it appears smoother to the human eye. Consider how a human would tackle the task of manually drawing a new frame to insert between two

existing sequential frames: They would need to look at the two video frames, one after the other, and then take the time to draw a middle point between all the objects in the frames depending on what kind of object they are seeing. A machine learning agent would tackle the task in a similar way: It would also have access to the two frames, one stacked on top of the other, and then it would use a CNN to predict a middle point per pixel between the two frames, and output this as an entirely new frame. CNNs are traditionally used for image classification problems where the first half of the network is extracting features, and the second half is a dense neural network which takes the features and determines how to classify the image. This kind of network architecture would not work for this project as layers in the second half of the network are flattened out to one-dimensional vectors and the aim of the project is to produce a three-dimensional tensor which would represent an image. Therefore, this project will use a 'encoder-decoder' style architecture to extract a usable image from the network.

1.2 Aim

The aim of this project is to research and develop a system that can use machine learning to accurately predict the structure of a video frame to be interpolated between a preceding frame and a succeeding frame (given as inputs to the CNN). The system will be able to accept an mp4 video file, enhance the frame rate of this file, and then output a new video file via a simple, intuitive graphical user interface.

1.3 Objectives

The first objective of this project is to pinpoint suitable datasets to train each prototype of the model on. Then, research the architecture of the neural network and decide what kind of model to use based on what current research suggests performs the best. Research will need to be made to determine what tools to use to build the software. Once the best framework to use to build the model has been determined: Design and build the first prototype of the neural network, test the neural network, tweak model parameters based on test error, and repeat testing / tweaking until model is performing as expected. After a first prototype has been built that shows the theory behind the project is viable, continue to gather a more complex dataset and test how the model performs at a larger scale.

1.4 Product Overview

1.4.1 Scope

After completion of the project, the resulting software artifact will be able to accept an mp4 video file and output a new video file with an enhanced frame rate. It will do this by interpolating frames between the existing frames to increase the overall frame rate of the video. The software will determine the structure of the interpolated frame by passing two frames from the video file to the convolutional neural network (CNN). The CNN will then extract features from the two frames, determine the characteristics of the features, and will set filter values that it believes will produce a middle ground of the two frames based on the characteristics of the features during the training process. This is an improvement to existing methods as it considers the characteristics of the objects that are moving, rather than just predicting based on the pixel difference mid-point or based on the velocity of the pixels.

1.4.2 Audience

The resulting artifact of this project will have multiple applications in industry. For one, it can be aimed at users who are unable to capture video frames at their desired rate. For example, artists in the stop-motion industry are required to create the elements of each frame by hand, which could take huge amounts of time. The software would be able to interpolate these stop-motion frames, requiring the stop-motion artist to only construct elements for half as many frames, saving time and money.

Another application, which is the main use case for the project, is to use the software to enhance frame rates for movies made in a certain era when high frame rates could not be achieved due to limitations of technology at the time. Also, as the movies are older, they may have lost some of their frames over time due to improper storage of the film canisters etc. The software would be able to restore those frames without having to perform the lengthy process of re-drawing the frames by hand.

The software could also be used for generating artificial slow-motion videos.

Slow-motion videos record at a very high frame rate and play back the videos at a lower frame rate to give the illusion that objects in the video are moving very slowly.

The software could interpolate artificial frames recursively to generate a very high frame rate video, without having to utilize expensive hardware to record at high frame rates. However, there are issues with this as recursively generating the frames will accumulate more error from previously generated frames (Jiang *et al.*, 2018).

The software could be adapted to provide some benefits to the streaming industry by enabling the streaming server to only send half the amount of video data to the client where the client end uses the neural network to then generate the missing frames. This could save a lot of money by reducing the network bandwidth required to transmit the data, however, the client endpoint would need to be a system with a large amount of computational power to be able to predict the frames in a timely manner, something that a typical user does not possess.

2. Introduction to Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a type of deep learning model used primarily for handling machine learning problems relating to large data samples, such as image data. Prior to the adoption of CNNs, artificial neural networks handled image recognition tasks by being provided with a representation of the features of an image, which were manually extracted using other algorithms (Bengio and Lecun, 1997). For example, in the case of the letter 'T', the features extracted would be a horizontal line and a vertical line. With CNNs, the feature extraction is built into the network in the form of trainable filters (kernels) which are 'passed over*' the image to extract specific features that the image contains (this process is repeated many amounts of times to extract multiple features, or feature 'maps').

CNNs contain a few different types of layers, each with their own function. The convolution layer, as described above, extracts features from the data, and produces an output containing representations of the features in the data sample. This output, much like regular neural networks, is then fed to an activation layer, which performs a function on the output to ensure that the network can learn non-linear data during the backpropagation phase of the deep learning algorithm.

The output of the activation layer is then passed to a pooling layer to reduce the spatial resolution of the feature maps (usually by a factor of 2) thereby reducing the dimensionality of the data, grouping features together and reducing the number of trainable parameters. This process aids the invariant characteristic of CNNs where outputs are not sensitive to translations (however, they are sensitive to other types of transformations, such as rotation) of features in the image (Brownlee, 2019).

These layers can be repeated multiple times, depending on how deep a network is required. Finally, the resulting output is compared to the 'ground truth' label, so that the loss of the network can be calculated, the backpropagation process can begin, and the filters can begin to learn what features are present in the image.

CNNs are well suited to machine learning problems relating to image data due to their sparsely, repeatedly connected neurons (rather than densely connected in a traditional artificial neural network), which lead to better computational efficiency and reduced risk of overfitting (due to the reduced number of connections) when processing large data samples.

*The 'passing over' of the filters is a helpful visualisation tool, however, the filter weights themselves are static, giving rise to the description of 'repeatedly connected' neurons.

3. Background Review

Many scholarly articles have been produced on the topic of frame interpolation where most, if not all, mention the traditional method of frame interpolation: Pixel synthesis via motion estimation and compensation. This is where an algorithm, given two neighbouring frames, calculates the motion vector of pixels belonging to a certain part of the image known as a 'block' of pixels (where the block is calculated using 'nearest-neighbour' strategies or regular searching algorithms). This motion vector is then used to synthesize the resulting block of pixels by warping the original frame with the motion vector. This can also be performed on a per-pixel basis, which begins to become very computationally expensive. The general consensus of traditional frame interpolation is that, whilst modern advances in this specific technology are able to produce accurate results, a lot of post-processing is required to fix errors that the motion estimation algorithms produce, such as the inability to predict pixels that are heavily occluded (Bao *et al.*, 2021). A machine learning approach to this problem would combine motion estimation, pixel synthesis and post-processing all into one system, however, will most likely still require more processing power than the traditional method of estimating motion vectors.

A common problem with image synthesis using convolutional neural networks is that resolution becomes lost in the convolution/pooling layers, and the output of the network seems blurry. This is inevitable with CNN models as pooling is necessary to reduce the amount of computational power needed to train or make a prediction from the model and to be able to group features together to produce new feature maps. Many papers solve this by introducing a concept known as 'skip' connections: A term coined by Liu *et al.* (Liu *et al.*, 2017) and introduced in the Ranzato *et al.* paper describing the first 'encoder-decoder' style architecture (Ranzato *et al.*, 2007). This concept has been adopted by many future papers, including Badrinarayanan *et al.*, where they propose the 'deep encoder-decoder' network (Badrinarayanan, Kendall

and Cipolla, 2017). Here the max pooling indices of the maximum values of each feature map after the first convolution (encoding layer) are stored in memory and then used in the up-sampling section of the relating decoder layer to produce an output which retains higher detail from the earlier layers. This is a technique which saves memory in the long run at the expense of slight accuracy loss. Other papers such as the 'U-Net' paper (an evolution of the architecture described in the Ranzato *et al.* paper, which has, itself, been used by many following papers) that inspired the architecture of this project are less austere with memory resources and opt to store the entire feature maps and then concatenate them with the relating decoding layers to retain detail (Ronneberger, Fischer and Brox, 2015).

A wide variety of model implementations are used by all papers, however, some hyperparameter details are used more frequently for this kind of machine-learning problem. Some papers refer to the research that using an L2 based loss function can lead to producing blurry results in the output (Zhao *et al.*, 2018), and so opt for L1 based losses (this project uses the Charbonnier loss function, described in *Section 4.1.3.i.*). Papers also tend to use the Adam optimiser, another version of the Stochastic Gradient Descent algorithm but with momentum which allows the model to converge to its optimal parameters much faster than regular gradient descent algorithms (Diederik and Ba, 2017).

4. Methodology

4.1 Development Process

4.1.1 Approach

This project will follow the Agile project management methodology and a prototyping software development model to quickly demonstrate functionality. Testing will occur throughout the development to evaluate the error of predictions that the network makes and tweak hyperparameters accordingly. For the prototype, a new dataset will be used of increasing complexity to test the limits of the network. Firstly, grayscale frames of simple images, then moving on to add dimensionality of colour values, finally, using more complex video frames. Using the 'Scrum' agile framework, several user stories from a backlog (see *Appendix 9.3* for product backlog link) are chosen for each sprint (a period of 2 weeks of programming in the case of this project). After each sprint, the product backlog is re-evaluated, and new user stories are chosen for the next sprint. This process repeats until all the user stories have been considered,

and all the sprint tasks have met their acceptance criteria (see *Appendix 9.1* for the sprint log for this project).

4.1.2 Requirements

Functional Requirements:

- The software must have an interface to be able to upload video files to the network for interpolation.
- The software must have an interface to be able to initiate training of the network / tweak hyperparameters of the network to validate optimal models.
- The software must have an interface for uploading video files to be added to the overall dataset that the model will be trained on.
- The software must implement a CNN able to accept two frames and output a resultant frame. (Interpolate the frames).
- The software should have a pipeline between the two modules (user GUI and CNN) in case the neural network module needs to be outsourced to the cloud for computational resources.
- The software should have a mechanism for the model to be saved after training and loaded for evaluation.
- The GUI should be able to produce a new video file for the user to download.
- The interface should show progress reports during training / forward passes.
- The model should be able to be trained in an 'online' mode.

Non-functional Requirements:

- The neural network must be able to produce an accurate representation of a middle frame, where 'ghosting' is kept to a minimum.
- The neural network should be able to quickly output a frame to interpolate after it has been trained.
- The interfaces should be able to run on all popular operating systems.
- Any exceptions in the software should be handled gracefully to produce a seamless experience.
- The program must be able to handle large datasets without memory being exhausted.

4.1.3 Design

i. The model (full model summary can be found in Appendix 9.2)

The core element of the software is the neural network that will predict the frames to be interpolated in a video to be enhanced. This will require a 'deep encoder-decoder' type architecture, as the output will need to be the same dimensions (minus the axis used for stacking the images in the input) as the input. As the network must retain as much pixel detail as possible in the resulting image, 'skip connections' will need to be used as discussed in the background review section. Thus, the best architecture for this project will be the U-Net architecture (see *Figure 1*). The specification of the U-Net will be that of the original architecture described by Ronneberger *et al.* (Ronneberger, Fischer and Brox, 2015) with some slight changes to suit the project aims and can be seen in *Figure 1*. The filter sizes of the first two convolution cycles are 7x7 and 5x5 instead of the original 3x3 filter size. This design choice was made so that pixels with a large range of movement (pixels that are moving faster) can be captured by the filter where a 3x3 filter would not be large enough to capture pixels at time t_0 and t_1 . The filters have a stride value of 1 and use the 'zero-padding' method to ensure that the height and width of the output is the same as the input. This results in the ability to be able to control how the network shrinks (as the output shrinks naturally during convolution), which makes it easier to concatenate layers together in the 'decoder' section of the network (as the layers must be of equal height and width). It also allows for border detail to be retained, as without padding, border detail is naturally lost during convolution.

The input of the network will accept a tensor of shape $B \times C \times H \times W$ where B is the batch size of the network, C is the number of channels (in the case of the input C is 6, a channel for each RGB channel of the two frames per data sample), H is the height of the frame and W is the width of the frame. The network will then output a tensor of shape $C \times H \times W$ where C this time will be 3 (as the network is outputting a single frame with 3 RGB channels and comparing against labels of the same shape) and H and W are, again, the height and width.

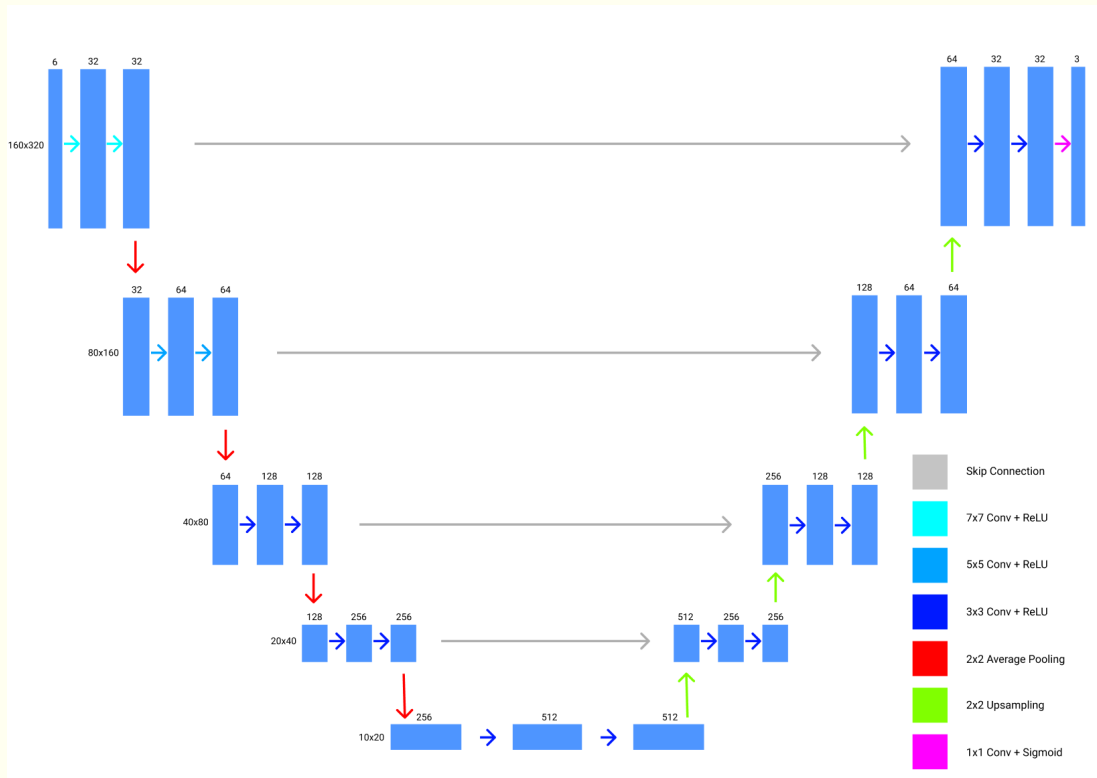


Figure 1 – Model Diagram. The numbers above the blue boxes indicate the depth of each tensor, i.e. the number of channels.

As multiple pooling layers are present in the network, this will decrease the resolution of the original image. When the data reaches the ‘decoder’ section of the network, it will begin to up-sample the frames to their original size and will make duplicates of the existing data to increase the dimensions. When the data reaches the output of the network, the resulting frame will have a very poor resolution as most of the data has been fabricated during the up-sampling transformations. This is not an acceptable solution for the network, as the interpolated frames will need to be of a similar pixel resolution to the original frames so that visual flickering in the enhanced video file is kept to a minimum. The solution to this problem is to introduce a concept known as skip connections, which carry detail through the network. This is done in the ‘decoder’ section of the network where up-sampling takes place. The up-sampling output is concatenated with a layer from the ‘encoder’ section of the network with corresponding height and width dimensions.

The activation function used for the CNN is the ReL (Rectified Linear) activation function (also known as ReLU when implemented in a network) as seen in *Figure 2*. This is used over functions such as the tanh (Hyperbolic tan) and the sigmoid activation functions as these two functions can saturate at -1 and 1 or 0 and 1

respectively. This means that weight changes that lead to an activation output at one of the saturable values will break the backpropagation of values algorithm as the derivative will always be 0 and thus the weights will not change. This is known as the 'vanishing gradients' problem and most often occurs in networks with a large number of weights or a learning rate that is too large. The ReLU activation solves this problem at one end of the scale (where the input to the function is positive), as it is linear after 0 meaning there will always be a value for the gradient (although it will always be 1). This means that the weights will always be able to update (as long as the input of the function is not negative or 0). At the other end of the scale (where the input to the function is negative or 0), a similar issue to the saturation of tanh and sigmoid occurs, known as the 'dying ReLU' problem. This is where the output of the activation function is 0 for inputs that are less than 0 and a similar situation to the 'vanishing gradients' problem occurs where the gradient is always 0 and the weights cannot update. This is often not a problem that occurs too frequently unless the learning rate is set too high and pushes weights towards the negative region. If it were that case that the 'dying ReLU' problem was happening (it is not the case in the network described in this report), then an activation function known as 'Leaky ReLU' can be employed to solve this.

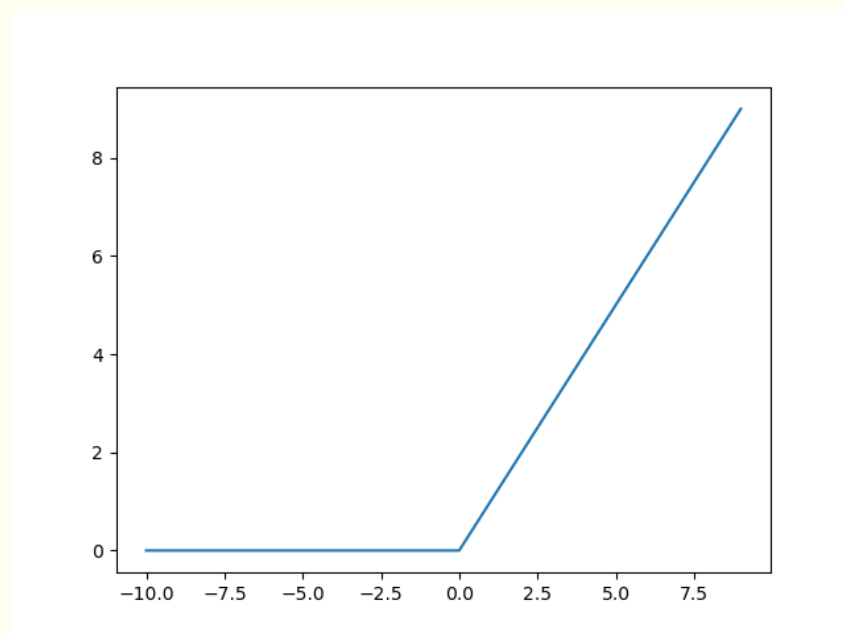


Figure 2 – ReLU activation Function.

The energy function (*Figure 3*) used for calculating the loss between the target label and the prediction of the network is known as the 'Charbonnier Loss': An L1 loss that

acts like an L2 loss at the minimum as a purely L1 loss becomes undifferentiable at the minimum and can have trouble converging (as the derivative is a discrete positive or negative value and is not continuous, so gradient descent cannot occur, only ping-ponging between the two gradients). An L1 loss is used based on the evidence that they produce higher quality results for image processing tasks in machine learning (Zhao *et al.*, 2018) and due to the fact that they are more suited to regression tasks (which is the case for the model described in this report) as they do not heavily fit to outliers. The energy function calculates the sum of the differences for each RGB value in the ground truth label and the prediction. Each difference is squared and then square rooted to produce the magnitude of the difference (L1 loss). The variable ϵ is set to a very low value that allows the loss function to act as an L2 loss close to the minimum whilst keeping the properties of an L1 loss.

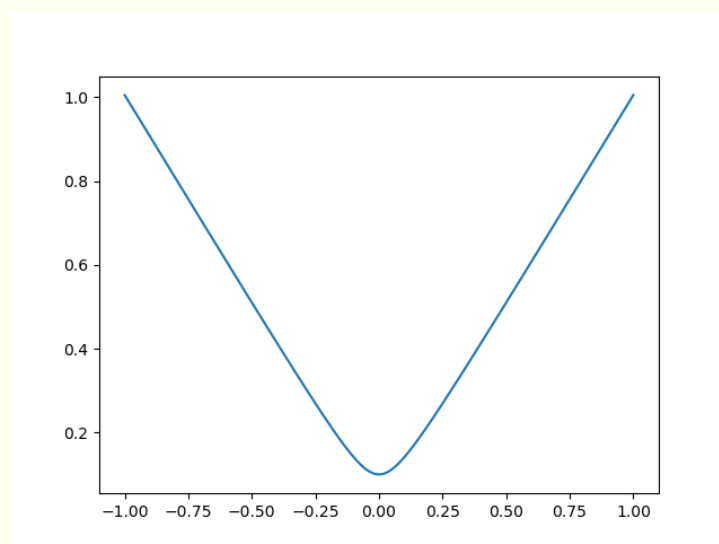


Figure 3 – Charbonnier Loss graph.

where

In *Figure 3* the value of ϵ is set to 0.1 for visualisation purposes on the graph, however, the actual model parameter is set to much lower (0.001).

The gradient descent method for updating the weights being used is the Adaptive Moment Estimation (also known as Adam) algorithm with mini-batches. The name Adaptive Moment Estimation comes from the fact that the Adam algorithm takes advantage of two methods used to aid the convergence of weights to the minimum, building off the base Stochastic Gradient Descent algorithm. The first being ‘momentum’ and the second being ‘adaptive learning rates’. Momentum considers

the gradients calculated in the previous weight updates step and increases the step size accordingly, leading to the rate of convergence being dependent on how far away from the global minimum the current weights are. If the weights are further away from the minimum, the value calculated for the gradients will be larger and so the step size increases. If the weights are close to the minimum, then the 'momentum' aspect of the algorithm begins to vanish and the step size stays the same. If the weights overshoot the local minimum, then the momentum begins to decrease, and the step size becomes smaller as the value for the gradient becomes negative. Adam takes the momentum concept even further by giving higher weightings to more recent gradient changes so that it can change the momentum more accurately on the most recent weight updates without worrying too much about what has happened in the past. This is achieved by calculating the exponentially weighted average of the momentum based on a value Beta (Ng, 2017). With adaptive learning rates, the learning rate changes based on whether the weights are converging in the correct direction. If the gradient is small, then learning is occurring in the correct direction and the learning rate very slowly decays as the weights approach the minimum. If the gradient is large, then the learning rate decreases largely to stop learning happening in the incorrect direction. Again, higher weightings are given to more recent gradient changes so that the learning rate is updated based on the most recent information.

The accuracy metric being used for determining the quantitative performance of the model is SSIM (Structural Similarity Index). SSIM (Wang *et al.*, 2004) is a measure of similarity between two images under the pretence that the human visual system determines similarity of two images based on perceived structural differences rather than the difference in each pixel. For example, a video that had a certain number of 'dead' pixels would make the picture look terrible yet still produce a high accuracy score, whereas this kind of noise would produce a low SSIM score. As well as the structure of the image, the differences in luminance and contrast are included in the SSIM calculation as these differences cause a noticeable change to the human eye, hence why it is a suitable metric for measuring accuracy of image data.

ii. Data format

For the model to train, it will require a large amount of data. This data is gathered in the form of mp4 video files. These video files are stored on the disk as compressed data, which will be of no use to the CNN as it requires pixel colour data for individual

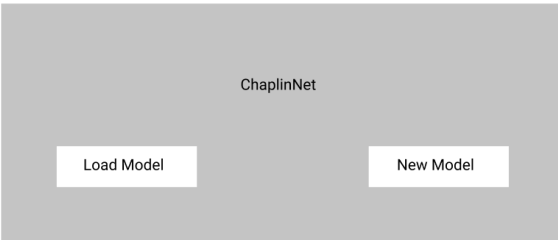
frames, so the frames must be extracted using the scikit-video library. These extracted frames will need to be stored on the disk, as bringing them straight into memory at once will require a huge amount of memory to accommodate the data. For example, a 20-minute mp4 video file at 25 frames per second will contain 30000 frames. If the resolution of the video is 360x504 this is equal to 5,443,200,000 pixels with 3 RGB channels which is equal to 16,329,600,000 RGB values. If each of these values require 1 byte to store the value, the amount of memory required to store the entire uncompressed video file will be 16,329,600,000 bytes or 16.3GB. This is a much too large amount to be able to be stored in memory, so it must be stored on the disk and loaded into memory for training in batches. The CNN also requires that the input datatype is a normalized 32-bit float value between 0 and 1. This data augmentation must be made at run time, as storing the data on the disk with 4 bytes per value will take up too much space.

In video files, frames are stored with a shape of HxWxC (Height, Width, Colour Channel). This data format does not suit the CNN model as it is expecting to receive a shape where the colour channels appear first (the `data_format` parameter for each Keras layer is set to `channels_first`). The model can receive tensors with a data format where the channels appear last, however, it is recommended by TensorFlow that when training using a GPU the data format should be that the channels appear first as this is better suited to the physical architecture of the GPU ****Reference TensorFlow article****. The data is stored on the disk in a channels first format, as augmenting the data during run time increases the time per training step dramatically, leading to increased overall training times.

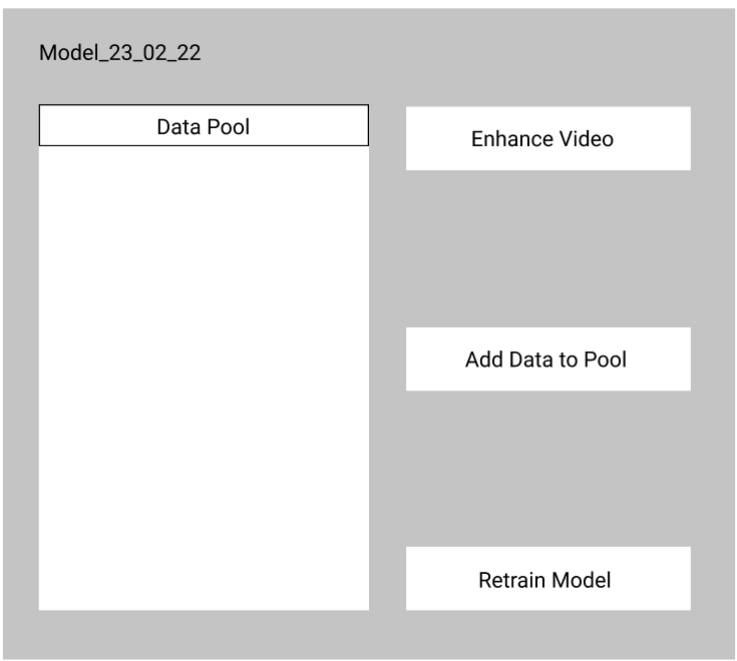
The data has been grouped in mini-batches of 32 samples to take advantage of the GPU parallelisation and increase training speeds. If the batch size were to be any larger, such as 64, there would not be enough GPU Video RAM to accommodate 64 parallel instantiations of the model, so 32 was chosen. Even if memory were an unlimited resource, a batch size of 64 samples would decrease the stochasticity of the gradient descent algorithm leading to a higher potential for becoming stuck in local minima on the error surface (how the error changes in relation to the weights of the network) of the dataset. Having a smaller batch-size increases the potential that an update in weight values will help the network error to escape being stuck in these local minima with the intention that a global minimum will be found.

iii. The GUI

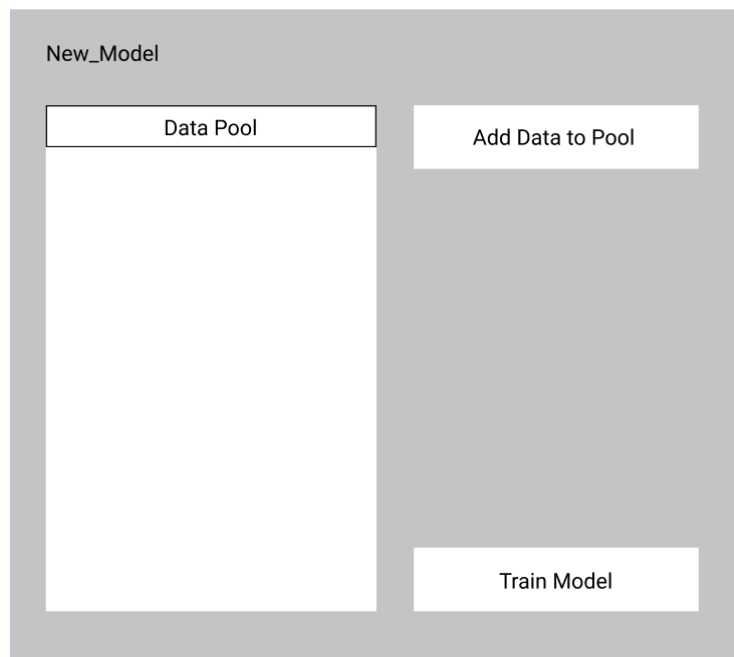
The GUI will be constructed using the PyQt5 framework. It will be a very simple GUI that the user of the software can utilise to create a new model, supply it with a dataset, train and test the model or load a previously trained model. *Wireframes* for each page of the GUI can be viewed below. For the GUI to be responsive while the model is training, or data is being loaded, the model must run on a separate thread of computation to the GUI. This will allow for progress to be reported to the GUI by the model which will be visualised in the form of a progress bar. PyQt5 provides its own high-level thread handling API (QThread), which allows the programmer to create separate threads / set up communication channels between threads (used for sending thread progress signals).



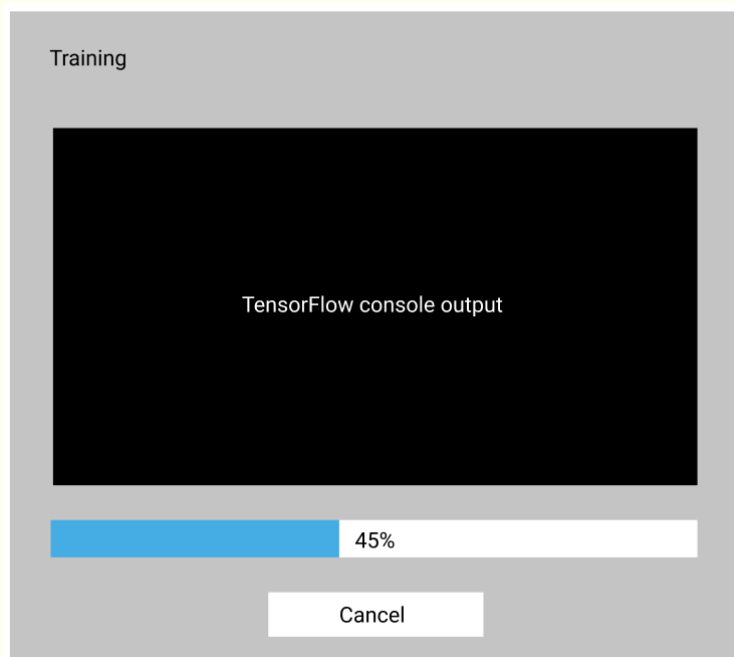
Wireframe 1 – Home Page.



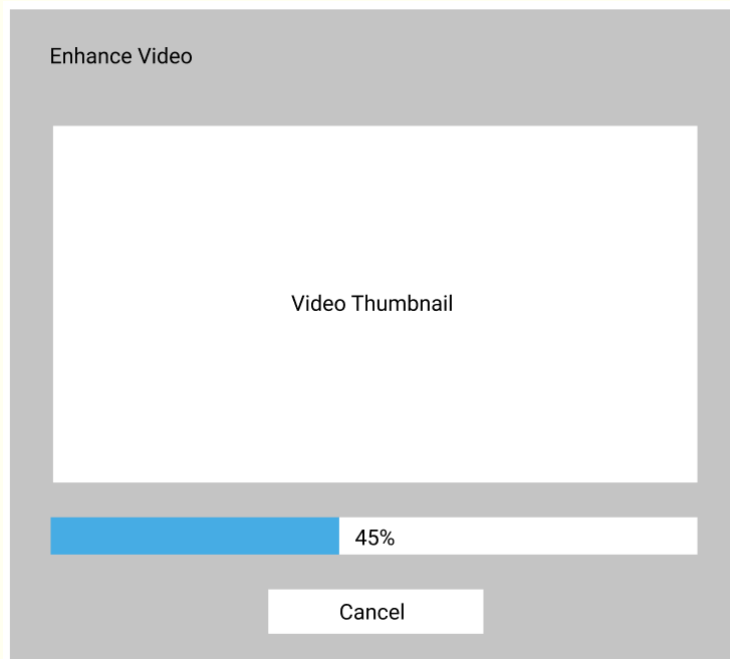
Wireframe 2 – Load Model Page.



Wireframe 3 – New Model Page.



Wireframe 4 – Training Page.



Wireframe 5 – Enhance Video Page.

4.2 Technology

4.2.1 Software

The development of this project will be aided by multiple pieces of software. The Python programming language will be used alongside Visual Studio Code as the Integrated Development Environment. Anaconda will be used as the virtual environment provider and package manager for the Python libraries that will be used to build the project. The following Python libraries will be used: TensorFlow Keras – for constructing the convolutional neural network model, NumPy – for dataset formatting/manipulation, ScikitVideo – for managing IO functions for video files, PyQT – for building the Graphical User Interface, Matplotlib – for displaying results of individual frames, SceneDetect – for detecting cuts in movie files.

4.2.2 Hardware

The process of formatting the dataset and training the model in a timely manner will require adequate computational resources. The most used piece of hardware for machine learning tasks is the GPU. The GPU is suited to these tasks as many of the calculations that are run in training the model can be done in parallel, and the architecture of the GPU allows for many separate threads of computation

(Madijagan and Raj, 2019). An NVIDIA GeForce RTX 3090 GPU with 10496 CUDA cores will be used to train the model. As the complexity of the model increases with each prototype, more computational power may be required and may need to be sourced via alternate avenues, for example, cloud computation.

4.3 Version Management Plan

For version management, a GitHub repository will be used to store the code (with the local repository stored on Google Drive). Commits to the repository will be made after each task in the Sprint and each Sprint will be its own branch to be merged to the master branch after the Sprint is completed.

4.4 Use Case

The main use case for this piece of software is to be able to enhance the frame rate of early-era movies, where the maximum frame rate is limited by the technology of the time. It is usually the case in these movies that the characters are moving comically fast, due to the frame rate being sped up when the films are shown. After passing the video through the software, when the resultant file is played through at the same frame rate, the characters should look like they are moving normally. However, this may ruin the aesthetic for passionate film aficionados, so the film could be played through at a faster frame rate and preserve the feel of the original while also making the movement appear smoother. Movies starring Charlie Chaplin have been chosen to build the dataset to train and test the model.

5. Implementation and Results

5.1 Prototype Experimentation

In the early sprints of the project, the aim was to ensure that the theory behind the project was sound and that it would lead to usable results. Initially, the model would need to predict intermediate frames for black and white videos with very simple objects (moving numbers). The size of the dataset is dramatically reduced to 1520 samples from 76000 to speed up troubleshooting and testing. The architecture is based off information gathered during the literary review stage of the project. The model resembles that of the 'encoder-decoder'. *Figure 4* shows the qualitative results after training the model on 1520 samples.

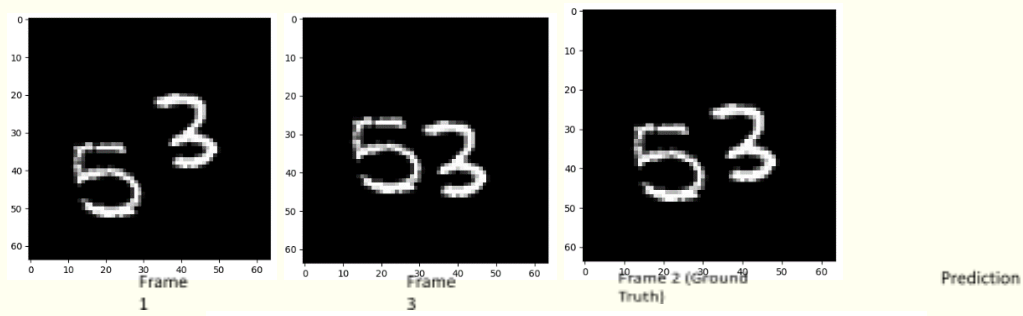
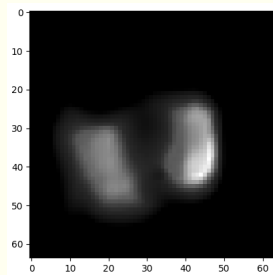


Figure 4 – Prediction of model on first initialisation.



From the results gathered, it can be seen that the CNN is predicting a halfway point of the objects in the frame without any specific detail. This is due to the absence of skip connections where resolution is being lost through the encoding part of the network. A second training progress was carried out in the following sprint, this time including skip connections to ensure edge detail is carried through the network. This can be seen in *Figure 5*.

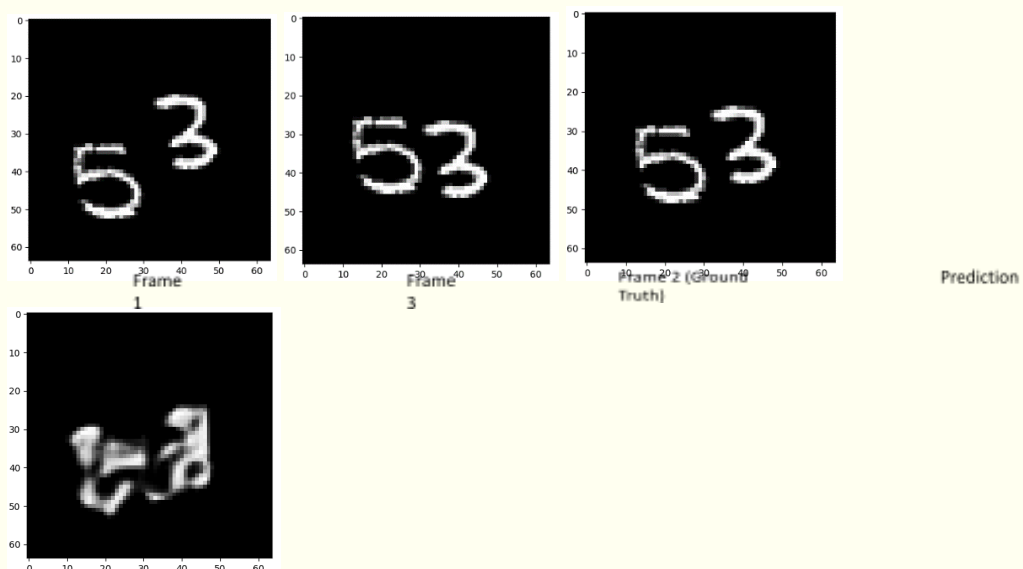


Figure 5 – Prediction of model with skip connections implemented.

From the results gathered from the second training process, it can be seen that the CNN is now still predicting the halfway point, however, more edge detail is being carried through. This detail is not the detail that is expected, so further investigation was performed to see where the network is going wrong. It was found that the dataset had not been formatted properly for a small number of samples. In these samples the two frames per sample are from different video sequences and the label does not match the sample. These data outliers will confuse the network by training the network to attempt to find a middle point between two completely different frames.

The next sprints were dedicated to rectifying the algorithm used to generate the dataset samples. The NumPy library was used here as it provided more user-friendly data manipulation techniques than the TensorFlow Datasets API such as the ability to be able to load in mp4 files frame by frame into memory as arrays rather than storing all data in memory at once. Once the data samples and labels had been properly formatted, they were saved to the disk so that the lengthy formatting process would not need to be completed every time that the model needed to be trained. These sprints were also dedicated to enabling GPU acceleration through TensorFlow via the NVIDIA CUDA API which allowed for the entire dataset to be used as the time taken to train the network was dramatically reduced from about 42 hours on the CPU to about 50 minutes on the GPU. The network was then retrained using the entire, correctly formatted dataset and the results can be seen in *Figure 6*.

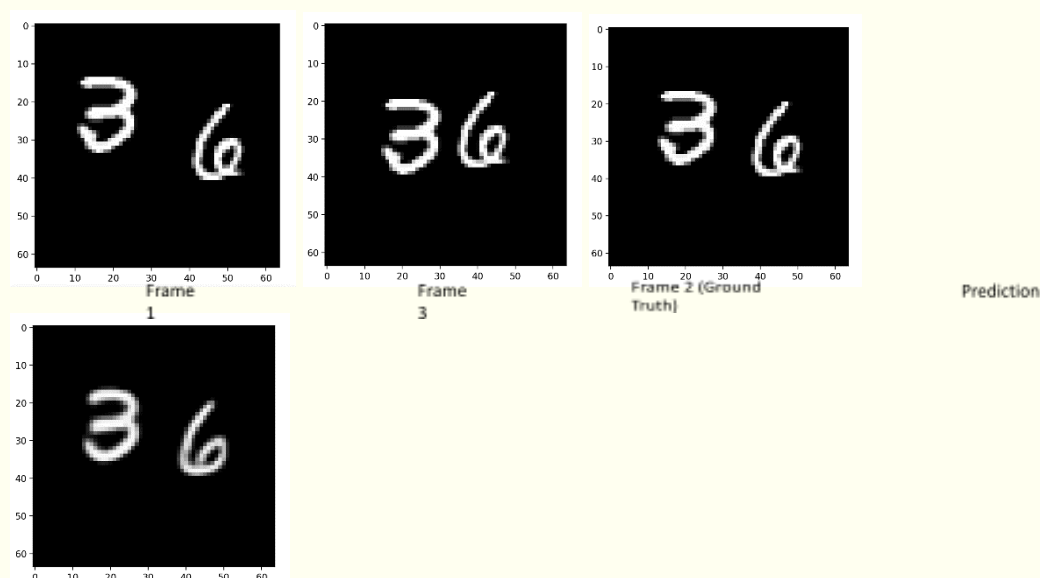
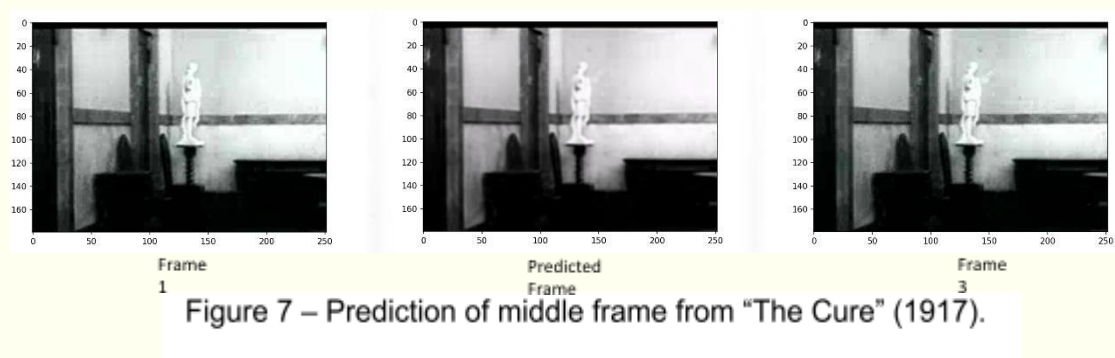


Figure 6 – Predictions with skip connections and accurate dataset.

From these results, it can be seen that the network is now training correctly and producing an accurate representation of a middle frame. Although the predicted frame is accurate, there are still some small details not being predicted correctly such as the line crossover in the middle of the 'six', and the bottom section of the 'three' joining with the middle section. This is due to the lack of data available to the model, so it is not able to predict these more specific features.

Now that the model has proven successful on trivial data, more complex data can be used to evaluate the performance of the model. The prototype model was reformatted so that it can accept complex video frames with colour channels (Sprint 7). This data was split into smaller sections so that the model does not take up too much video-RAM resulting in a model accepting a larger amount of data with a shape of 6x180x252, however, GPU parallelization can be utilized to increase the throughput of the network. The complex data shape is significantly larger than the simple data shape (6x64x64) and so lead to a large increase of time taken to train the model. This increased from 50 minutes for the simple dataset to 30 hours for the more complex dataset. The qualitative results (as seen in *Figure 7*) from this test were promising: The network could accurately predict frames depicting complex images containing many different objects. The network was trained on the movie "Easy Street" (1920) starring Charlie Chaplin and tested on the movie "The Cure" (1917), again, starring Charlie Chaplin.



The results show that the predicted frame is almost exactly accurate to what one would expect the middle frame to look like, however, where "The Cure" has a slight green hue to each frame, "Easy Street" does not have this hue. The predicted frame does not have the green hue as the model is biased towards the non-green hue. Furthermore, the performance of the model is limited to the quality of the data that is used to train the model. As the dataset is not built from the original film rolls and

have probably been through many rounds of compression/decompression to be made available on the internet as mp4 files, the data can sometimes contain 'blocky' artifacts, which will confuse the model when it is trying to predict higher quality frames. Thus, the decision was made to train the model on higher quality data and then use that model to enhance the lower quality videos.

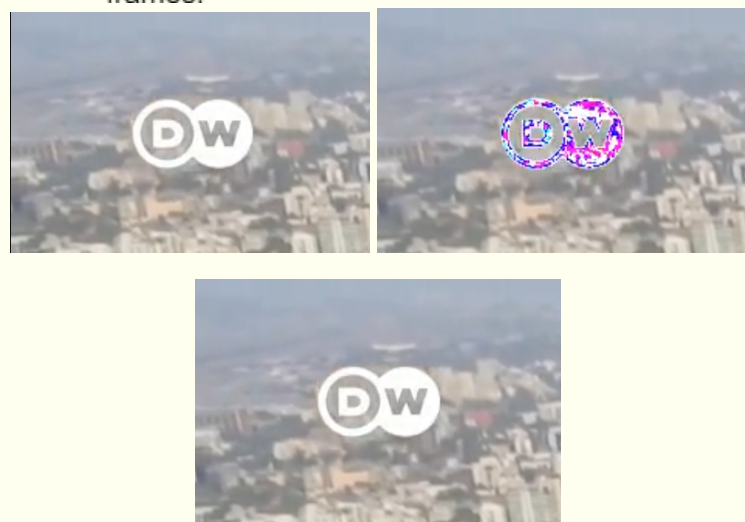
5.2 Final Product Testing

The final iteration of the project was to produce a software artifact that could predict intermediate frames from complex data with three colour channels. After the first prototypes performed to a high standard, it was time to train the model using modern footage that contained complex objects in the frame with a large range of colour. The frame sizes of the selected dataset were too large to be able to fit through the prototype model, so the frames had to be sliced into smaller sections that could each fit through the model. This provided the model with a largely increased amount of data to train on, and so training times were increased.

The model was trained on a video containing complex colour and object data (found at <https://javafilms.fr/film/mumbai-the-infernal-megapolis/>, (LigneDeFront, 2021)) and then ran through the video enhancing algorithm to interweave the predicted frames into the same video (the training samples this time had a smaller temporal distance as there are no ground truth labels to pick out, so the predicted frame could be taken as an accurate representation of a prediction made from unseen data). After this had been completed, the final video could be viewed frame-by-frame to determine qualitative performance. While the video performed very well with slower, moving objects, when it came to predicting movement of faster objects that took up more screen space, a checkering artifact became visible in the predicted frame. A second problem was also present, where static objects or watermarks over the video presented with a random amount of incorrectly predicted pixels of seemingly random colour values (*Figure 8*). Also noticed in the resultant video was that strange artifacts are viewed in scenes with very fast-moving objects that the shutter speed of the camera fails to capture adequately (such as the flapping of a bird's wings). The original frames contain ghosting where the bird's wings are flapping due to the shutter speed of the camera filming the bird not being fast enough to capture the movement. This meant that the network began predicting its own ghosting as it began to believe that is how a bird flaps its wings from the original data. As this is a

limitation of the original data, the network should not be penalized for displaying these artifacts.

Figure 8 – Colour anomalies in predicted frames.



Upon further analysis of the discoloured pixel problem, it became clear that this was happening on static parts of the image with one of the RGB values in *Table 1*.

Colour	R,G,B Value (Decimal)
White	255,255,255
Black	0,0,0
Red	255,0,0
Green	0,255,0
Blue	0,0,255
Magenta	255,0,255
a	

Cyan	0,255,255
Yellow	255,255,0

Table 1 – RGB Colour codes.

It seemed as if sometimes values that should be predicted as 255 were predicted as small values closer to 0, producing once of the colours from the table above. After investigating further, it was found that the array of predicted values from the model had a minimum value of 0 and a maximum of 1.07. This is normal, as the ReLU activation does not have an upper boundary, but it would be desirable if it were predicting values in the range of normalized RGB values (0 to 1). When converting the predicted values to RGB codes, this caused a numeric 'wrap around' as demonstrated in the following theoretical example: The model has predicted (1.0,1.0,1.07) as the RGB values for one pixel. The true pixel is white (255,255,255). When de-normalizing the values: $(1.0 \cdot 255, 1.0 \cdot 255, 1.07 \cdot 255) = (255, 255, 272.85)$. Finally, converting these values to uint8 RGB values: (255,255,16). Hence, a wrap-around occurs and causes the decolorized pixels visible in the final product. To rectify this issue, the final layer was passed through a sigmoid activation to normalize the values between 0 and 1.0.

As for the 'checkering' problem, it is obvious that this is occurring as an unpleasant side effect of cutting the video up into smaller parts that can fit into the model. Objects in the original frame that move a distance further than the dimensions of the model over three frames will be moving for too fast for the model to learn how they move, so the model will make a best guess for this scenario, causing the checkering artifacts seen. To solve this, the model would need to be large enough to capture the object's full range of motion, however, this would mean an increase in memory required to store the model and increased training times.

To evaluate the model quantitatively, the dataset was split into a testing dataset and a training dataset after first being shuffled (to eliminate sampling bias). This split is made so that the performance of the model can be based on how it makes predictions to unseen data, as this is how the model will be used in production. More complex test/train splits can be used such as k-fold cross validation splits, which further eliminate any sampling bias that could be present when choosing the test/train splits. However, the dataset is large enough that there is enough data that sampling bias becomes less and less likely to happen, so a simple train/test split will

suffice. K-fold cross validation is also very computationally expensive as the entire training/testing process takes as many times longer as there are ‘folds’ made of the data. The results below gathered the loss values and SSIM accuracy (a value ranging from 0 to 1) across both the entire training dataset and the entire testing dataset where the training dataset was made up of 80% of frames taken from the “Megacity Mumbai” documentary and the test dataset made up of the remaining 20% of frames. Another dataset of frames taken from the Charlie Chaplin movie “The Count” were taken to determine whether the model had become biased and overfitted to data available in the Mumbai documentary. This showed a $\sim .008$ decrease (see *Table 2*) in SSIM accuracy, which suggests some very slight overfitting to the Mumbai documentary data, however, the footage from “The Count” was of lower quality and consisted of blocky artifacts left over from compression, so it is possible that the model is struggling to predict for these situations. Other datasets were also taken from other Charlie Chaplin movies to determine the average time it would take to predict one frame (Time To Predict, TTP). These videos are of varying frame sizes, so it is expected that varying results will be seen depending on how long the network takes to predict a frame. The Loss value results observed may seem like some anomalies are occurring at first, as some datasets report a lower loss (suggesting more accurate predictions) than the training dataset yet yield a lower SSIM score. This is due to the proportion of the frame that has been padded to fit through the model. For example, when splitting the “Easy Street” frames into sections that can fit through the model, the sections will be of dimensions 120x252 and the “Megacity Mumbai” sections will be of dimensions 135x320. The “Easy Street” sections will be padded to fit the 160x320 model dimensions and will have a higher proportion of padding than that of the “Megacity Mumbai” sections. The higher proportion of padding will mean that more pixels in the predicted section will equal those in the ground truth section leading to a lower loss value.

	Loss	SSIM	TTP (s)
Training Results (10 epochs)	0.0121	0.9561	-
“Megacity Mumbai” – Ligne de Front (1080x1920)	0.0123	0.9554	0.36
“The Count” – Charlie Chaplin (480x640)	0.0135	0.9471	0.24
“The Property Man” – Charlie Chaplin (480x630)	0.0234	0.9107	0.23

"Easy Street" – Charlie Chaplin (360x504)	0.0089	0.9373	0.21
"The Cure" – Charlie Chaplin (240x320)	0.0158	0.9096	0.08
"The Floorwalker" – Charlie Chaplin (240x320)	0.0175	0.8785	0.08

Table 2 – Results of network performance.

Real-time qualitative results can be viewed in the video presentation linked in *Appendix 9.3* (A monitor must be used with a minimum refresh rate of 60Hz).

6. Professional Issues and Risk

6.1 Risk

Risk Assessment:

ID	Future Risk	Probability (0-1)	Impact (1-5)	Overall Risk (Probability *Impact)	Resolution
1	Unable to locate a suitable dataset.	0.3	4	2	Ensure that backup dataset is available in case of being unable to locate the perfect dataset.
2	Chosen dataset has too many outliers.	0.3	3	1.8	Ensure that there is a suitable data cleaning protocol in place.
3	Underestimate time taken to train model.	0.7	4	2.8	Allow significant development time for training of the model or purchase high spec computing resources to mitigate high training times.

4	Underestimate time taken for development of prototype.	0.7	5	3.5	Reduce number of prototypes needed to allow for more development time per prototype.
5	Original theory for software does not work in practice.	0.9	5	4.5	Ensure progress is tracked and discussed with project supervisor.
6	Code is lost unexpectedly or becomes corrupted.	0.3	5	1.5	Ensure that version control solution is in place, and all project resources are backed up to the cloud.
7	Dataset formatting is too intense on computing resources and starves memory.	0.9	5	4.5	Ensure algorithm with low complexity is used/that TensorFlow Dataset API is being used correctly.
8	Too much time spent on components that will not be included in the final project.	0.5	3	1.5	Keep a detailed project plan/user story backlog to ensure the most important parts of the project are prioritised.
9	Unable to meet certain deadlines.	0.4	5	2	Follow strict project plan and continue weekly meetings with supervisor.
10	Model overfits data.	0.3	4	1.2	Gather wider range of data to feed into the network to reduce

					sampling bias. Ensure substantial validation processes are employed.
11	Software bugs introduced due to poor software design techniques.	0.5	3	1.5	Employ software design best practices and stick to them.
12	Unable to write efficient TensorFlow code due to unfamiliarity with framework.	0.2	2	0.4	Refer frequently to TensorFlow documentation.

Risk Severity Matrix:

1.0					
0.9					5,7
0.8					
0.7				3	4
0.6			2		
0.5			8,11	1	
0.4					9
0.3				10	6
0.2		12			
0.1					
	1	2	3	4	5

6.2 Professional Issues

6.2.1 Legal Issues

One legal issue that could be associated with this project could arise from the gathering of data to build a training dataset for the model. The first prototype of the project uses the 'Moving MNIST' dataset (Srivastava, Mansimov and Salakhutdinov, 2016) which is licensed under the Creative Commons Attribution 4.0 License, meaning the content of the page is free to be shared and adapted in any way (<https://creativecommons.org/licenses/by/4.0/>). The video used to create the complex dataset was created by Ligne de Front (<https://www.lignedefront.fr/>) and explicit permission to use the video for training and screenshots in the report was given by Ligne de Front via email. Finally, the Charlie Chaplin videos used are all in the public domain, meaning their copyright has expired and they are free to use.

6.2.2 Social and Ethical Issues

Machine Learning famously has many ethical concerns to consider, such as the loss of jobs for 'real' people. The applications of this project could involve decreasing the

workload for stop-motion animators by reducing the number of frames that they have to produce by hand. This could potentially limit the number of jobs available to stop-motion animators, however, it would provide a new job position at the theoretical company for a machine learning engineer who would oversee operating of the software used to generate the artificial frames.

6.2.3 Environmental Issues

Environmental issues that are often associated with machine learning is the amount of energy that is required to train a machine learning model. As more complex machine learning tasks come about, more computing power is required to train these models, and more power means potentially more carbon emissions into the atmosphere. This would depend on the type of data centre chosen to host the computing resources. Some data centres run wholly on solar energy, and, therefore, would be the best place to host machine learning applications that would be beneficial to the environment. Lacoste et al. (2019) propose that machine learning engineers run their 'Machine Learning Emissions Calculator' to consider the environmental impact of machine learning projects.

7. Conclusion and Further Work

Upon the completion of the project, the CNN model developed has been able to learn how to generate an intermediate frame for any two sequential frames it is supplied with after being trained on the "Megacity Mumbai" dataset. The frames predicted show high levels of similarity based on the Structural Similarity Index (SSIM) and are also accurate based on subjective, qualitative evaluation. The maximum SSIM accuracy reached was 0.9554. The predicted frames also show no 'ghosting' artifacts when handling occlusion are present in non-learning methods of frame interpolation such as Motion Estimation and Motion Compensation.

Further work can be undertaken to increase the accuracy of frame predictions for videos that differ from the dataset (as is the case with the "Megacity Mumbai" data and the "Charlie Chaplin" data) by selecting a dataset with a large spread of different kinds of videos. For example, the dataset should be made up of videos with varying colour schemes, different object types and different camera angles to provide a more generalised representation of all the videos that the network may come across.

In terms of the functional and non-functional requirements for the project, all the requirements that the software *must* have were met, and most of the requirements that the software *should* have were also met. The software has not implemented a

pipeline for connecting the network to the user interface, as processing power on the development machine was sufficient to not have to source processing power from the cloud. The software has also not implemented an online training version of the algorithm, as the time for implementing this kind of solution was not factored into the development time of the project, however, this could be a feature implemented in a future version of the project.

Although the network has successfully predicted intermediate frames to perform interpolation, it takes a long time and a large amount of processing power to produce these frames, leading to a long waiting time to up-sample mp4 files. Further work can be undertaken to attempt to increase the prediction speeds of the network so that it could become a suitable replacement solution for current Motion Estimation and Compensation algorithms present in modern television sets. One potential solution would be to determine the smallest model possible that would produce acceptable results which would massively increase the amount of parallelization that the GPU could perform and therefore increase the speed of the network.

The GUI produced over the development of the project is very simple and used only for demonstration of the neural network developed. Further work can be undertaken to add more features to the GUI, allowing the user to experiment further with the neural network by altering values for various hyperparameters of the model to determine what models perform the best for the data they provide.

To further increase the accuracy of the predictions that the network produces, advanced techniques that other papers have proposed and received promising results from can be employed. For example, the network developed over the course of this project is slightly similar to the 'Adaptive Convolutional Neural Network' (Niklaus, Mai and Liu, 2017). One of the techniques in this paper is to take the gradients of the difference between one pixel and its surrounding 8 pixels, convolve the sum of the gradients with the kernel produced by the network, then subtract the value from the value calculated by performing this process on the ground truth frame. This makes the model consider how much the pixel is changing in relation to its surrounding pixels rather than just predicting the amount that the pixel will change, leading to a less blurry result. Implementing techniques such as this one will help the neural network produce higher quality predictions.

Personally, a large amount has been learnt over the course of the project with the most important fact being that data is the number one thing to get correct when it comes to machine learning. If the data that is being fed into a neural network is poor,

then the result is going to be poor. Luckily, for this project, it is easy to see when poor results are being predicted as the result is viewed in image form, however, this may not be the case for other projects. For example, a neural network might be predicting housing prices and could output a reasonable result even if the data fed to the network was incorrect. It would be much harder to spot an error in data in this scenario, so it is very important that the data is checked and formatted correctly. This became more and more obvious throughout the project as this was the issue that came up the most times when troubleshooting and will be first thing addressed in future projects.

8. Bibliography

- Badrinarayanan, V., Kendall, A. and Cipolla, R. (2017) 'SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12), pp. 2481-2495. doi: 10.1109/tpami.2016.2644615.
- Bao, W. *et al.* (2021) 'MEMC-Net: Motion Estimation and Motion Compensation Driven Neural Network for Video Interpolation and Enhancement', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(3), pp. 933-948. doi: 10.1109/TPAMI.2019.2941941.
- Bengio, Y. and Lecun, Y. (1997) 'Convolutional Networks for Images, Speech, and Time-Series'.
- Brownlee, J. (2019) 'A Gentle Introduction to Pooling Layers for Convolutional Neural Networks'. Available at: <https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/>.
- Diederik and Ba, J. (2017) 'Adam: A Method for Stochastic Optimization', *arXiv pre-print server*. doi: arxiv:1412.6980.
- Jiang, H. *et al.* (2018) *Super SloMo: High Quality Estimation of Multiple Intermediate Frames for Video Interpolation*.
- Kemp, J. (2019) *Film on Video*. 1st edn. London: Routledge.
- LigneDeFront 2021. Mumbai: The Infernal Megapolis. Java Films: Deutsche Welle.
- Liu, Z. *et al.* (2017) 'Video Frame Synthesis using Deep Voxel Flow', *arXiv pre-print server*. doi: arxiv:1702.02463.
- Madijagan, M. and Raj, S. S. (2019) 'Chapter 1 - Parallel Computing, Graphics Processing Unit (GPU) and New Hardware for Deep Learning in Computational Intelligence Research', in Sangaiah, A.K. (ed.) *Deep Learning and Parallel Computing Environment for Bioengineering Systems*: Academic Press, pp. 1-15.
- Ng, A. 2017. Understanding Exponentially Weighted Averages. YouTube: DeepLearningAI.
- Niklaus, S., Mai, L. and Liu, F. (2017) 'Video Frame Interpolation via Adaptive Convolution', *arXiv pre-print server*. doi: arxiv:1703.07514.
- Ranzato, M. A. *et al.* 'Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition'. 2007: IEEE.
- Ronneberger, O., Fischer, P. and Brox, T. (2015) 'U-Net: Convolutional Networks for Biomedical Image Segmentation', *arXiv pre-print server*. doi: arxiv:1505.04597.
- Srivastava, N., Mansimov, E. and Salakhutdinov, R. (2016) 'Unsupervised Learning of Video Representations using LSTMs', *arXiv pre-print server*. doi: arxiv:1502.04681.

Wang, Z. *et al.* (2004) 'Image Quality Assessment: From Error Visibility to Structural Similarity', *IEEE Transactions on Image Processing*, 13(4), pp. 600-612. doi: 10.1109/tip.2003.819861.

Zhao, H. *et al.* (2018) 'Loss Functions for Neural Networks for Image Processing', *arXiv pre-print server*. doi: arxiv:1511.08861.

9. Appendices

9.1 Project Log

20/09/21 – 03/10/21 Sprint 1 S1 Weeks 1 & 2

Sprint number: 1	Start date: 20/09/21
User story: As the user of the software, I would like the software to handle reformatting of the data so that I do not have to provide a specific data format.	
Estimated effort: 10 hours	Actual effort: 15 hours
Task 1: Use tensorflow_datasets API to import moving_MNIST dataset from the cloud.	Acceptance criteria: TensorFlow tfRecord file appears in the Dataset directory.
Task 2: Use tensorflow.data API to split moving_MNIST dataset into samples and labels.	Acceptance criteria: Dataset is split into samples and labels one by one in a 50/50 ratio.
Task 3: Use tensorflow.data API to cast moving_MNIST dataset into float16 datatype.	Acceptance criteria: Dataset takes up less space in memory and on the disk than with float32 datatype.
Task 4: Use tensorflow_datasets API to stack frames into preceeding and succeeding frames.	Acceptance criteria: Dataset sample shapes are now (64,64,2).
Reflection: This aim of this sprint was to download the moving_MNIST dataset and sort it into a format that would work with the proposed model. I found that the tensorflow.data API was not very useful for manipulating the data as the data is only brought into memory when the model is training. The map() method for the Dataset objects can be used to run a function on each item in the dataset when it eventually is enumerated by the model, however, two items in the dataset cannot be stacked together in this way. After spending some time trying to find a workaround, I decided it was best to use a different tool and install the NumPy library.	

04/10/21 – 17/10/21 Sprint 2 S1 Weeks 3 & 4

Sprint number: 2	Start date: 04/10/21
User story: As a user of the software, I would like to be able to train a Convolutional Neural	

Network model on data that I have gathered so that I can predict video frames.	
Estimated effort: 10 hours	Actual effort: 20 hours
Task 5: Use tensorflow.keras to build a sequential model based on a UNet architecture.	Acceptance criteria: When the program builds the model, after running the summary() method on the Model object, a summary is shown which describes a UNet architecture.
Task 6: Manipulate the training samples/labels into a format that will be accepted by the model.	Acceptance criteria: The samples and labels are split into two arrays of even length which correspond to the x/y parameters of the model.fit() method.
Task 7: Train the dataset on the model and evaluate the results.	Acceptance criteria: The model trains without throwing an exception about incorrect data format.
<p>Reflection: The aim of this sprint was to start building the Convolutional Neural Network model that would be able to train on the dataset. TensorFlow's Keras library offers two different methods to build models: The functional method or the sequential method. As I was not aware that there were two different methods, I began building the model using the sequential method. It eventually became clear that the sequential method was not suitable for adapting the UNet architecture, as it is not able to combine two layers. As the name suggests, it is only for constructing models layer by layer.</p>	

18/10/21 – 31/10/21 Sprint 3 S1 Weeks 5 & 6

Sprint number: 3	Start date: 18/10/21
User story: As a user of the software, I would like datasets to be able to be saved after they have been formatted so that I do not have to build the dataset every time I run the program.	
Estimated effort: 3 hours	Actual effort: 5 hours
Task 8: Use NumPy to save dataset to disk so that the lengthy formatting process does not have to run every time.	Acceptance criteria: Datasets are stored in the Datasets directory.
<p>Reflection: The aim of this story was to reduce the amount of time taken to run the training process. Previously, it would be required that the dataset would need to be loaded into memory, and then formatted, which would take up even more memory. After changing the flow of the program to include saving the formatted dataset to the disk, the program can just use the numpy.load() function to bring the already formatted data into memory and start training straight away.</p>	

Sprint number: 3	Start date: 18/10/21
User story: As a user of the software, I would like to be able to train a Convolutional Neural Network model on data that I have gathered so that I can predict video frames.	
Estimated effort: 3.5 hours	Actual effort: 2 hours
Task 9: Convert model to use the keras functional API to enable construction of shared layers for skip connections.	Acceptance criteria: The model can concatenate two layers to implement the 'skip connections' feature of the UNet architecture.
Reflection: The aim of this story was to change the method of building the model from using the Keras Sequential API to using the Keras Functional API. Now that 'skip connections' have been implemented, the model predictions have started to look closer to what they should be. The predictions are no longer blurry representations of the spatial properties of the ground truth frame, they now also include edge detail. However, this edge detail is not like that of the ground truth frame.	

01/11/21 – 14/11/21 Sprint 4 S1 Weeks 7 & 8

Sprint number: 4	Start date: 01/11/21
User story: As a user of the software, I would like to be able to train a Convolutional Neural Network model on data that I have gathered so that I can predict video frames.	
Estimated effort: 9.5 hours	Actual effort: 10 hours
Task 10: Convert original moving_MNIST dataset into NumPy so that the data can be formatted correctly without errors.	Acceptance criteria: moving_MNIST can be brought into memory as a NumPy array for more flexible manipulation methods.
Task 11: Use NumPy to split data into training/testing dataset.	Acceptance criteria: Training and test data is split in a ratio of 80 to 20.
Task 12: Use NumPy to split training/testing samples from labels.	Acceptance criteria: Samples and labels are separated from each other in different NumPy datasets.
Task 13: Use NumPy to stack training/testing samples into preceding and succeeding frames.	Acceptance criteria: Sample frames are stacked in such a way that the sample data still equals the length of the label data. The data shape should be (64,64,2).

Task 14: Train the correctly formatted dataset on the model and evaluate the results.	Acceptance criteria: The model produces good qualitative results.	
<p>Reflection: The aim of this sprint was to switch the data formatting tool to NumPy so that I could manipulate the data more efficiently and be able to correctly format the data for the model. Using NumPy made formatting the data trivial, however, the entire dataset must now be loaded into memory to be able to manipulate it. This was not disastrous as currently the moving_MNIST dataset is small enough to be able to fit into memory, however, once the model begins training on more complex datasets, a different process must be taken to ensure memory is not exhausted.</p>		

15/11/21 – 28/11/21 Sprint 5 S1 Weeks 9 & 10

Sprint number: 5	Start date: 15/11/21	
User story: As the user of the software, I would like to be able to run computations for the neural network on a GPU so that the network trains and performs faster.		
Estimated effort: 4.5 hours	Actual effort: 6 hours	
Task 15: Switch to anaconda environment so required python packages can be installed easier.	Acceptance criteria: Virtual environment is now Anaconda instead of venv. Now able to install packages from conda repositories.	
Task 16: Install GPU version of TensorFlow.	Acceptance criteria: TensorFlow-gpu package is now installed from conda repository and can be seen by running 'conda list'.	
Task 17: Install NVIDIA GPU into hardware.	Acceptance criteria: Hardware and drivers are installed, and the OS can see the GPU in Device Manager.	
Task 18: Install NVIDIA CUDA/Cudnn software.	Acceptance criteria: TensorFlow can recognise the GPU as a usable device and automatically selects it to train models on.	
Task 19: Determine a suitable batch size for GPU parallelisation and retrain the model.	Acceptance criteria: The model can fit in the GPU memory. Training the model takes substantially less time than when trained on a CPU.	
<p>Reflection: The aim of this sprint was to enable GPU acceleration for the CNN model. After attempting this using an AMD GPU, it became clear that the functionality required by TensorFlow did not support AMD hardware, and that TensorFlow only supported GPUs that could provide the CUDA API. The CUDA API is mostly offered by NVIDIA hardware, so the AMD GPU was switched out for a NVIDIA GPU. The installation of the multiple libraries required for enabling GPU acceleration on TensorFlow was no easy task, and it is suggested that these libraries are installed using the conda package manager instead of the pip package manager. It is for this reason that I switched the virtual environment from venv to Anaconda.</p>		

29/11/21 – 12/12/21 Sprint 6 S1 Weeks 11 & 12

Sprint number: 6	Start date: 29/11/21
User story: As a user of the software, I would like to be able to train the model on more complex video files so that I can enhance more complex videos.	
Estimated effort: 12 hours	Actual effort: 12 hours
Task 20: Rewrite data pipeline so that training data can be formatted and trained using batches so that memory resources are not exhausted.	Acceptance criteria: The dataset no longer consumes all the memory, however, it is loaded by the model in batches when it is needed.
Task 21: Install scenedetect python library so that complex video files can be split into individual scenes and 'cuts' won't interfere with formatting.	Acceptance criteria: Importing scenedetect does not result in an error.
Task 22: Load and split complex video files using NumPy and scenedetect into scenes.	Acceptance criteria: Samples or their corresponding labels do not contain any blank frames where the cuts are performed.
Task 23: Test that scenes are splitting properly.	Acceptance criteria: After viewing all the individual cuts of the video file, no cuts have been missed by scenedetect.
Task 24: Merge all formatted scenes' frames together so that 'ragged' NumPy arrays aren't created.	Acceptance criteria: The resulting dataset is not batched by length of scene, the sample/label pairs are stored one after the other.
<p>Reflection: The main aim of this sprint was to be able to split an mp4 video file into accurate sample/label data which would not be interfered by 'cuts' in the video file. This was achieved by using a python library 'scenedetect' which scans a video file and looks for large differences in frame data to determine when a cut has happened. This required some fine tuning of the splitting function to ensure that cuts weren't missed. To be able to then save the formatted data to the disk, the original video file needed to be loaded in using a python generator, which only loaded a few frames into memory at a time rather than the entire file.</p>	

10/01/22 – 23/01/22 Sprint 7 S2 Weeks -2 & -1

Sprint number: 7	Start date: 10/01/22
------------------	----------------------

User story: As a user of the software, I would like to be able to train the model on video files in colour format so that I can enhance colour video files.	
Estimated effort: 8.5 hours	Actual effort: 12 hours
Task 25: Reformat model input to accept data in a 6*Height*Width shape for RGB channels.	Acceptance criteria: The model accepts samples with 3 colour channels per frame with a shape of 6*Height*Width.
Task 26: Reformat model output to train against labels with 3*Height*Width shapes.	Acceptance criteria: The model outputs a frame with 3 colour channels with a shape of 3*Height*Width.
Task 27: Train model on colour dataset and evaluate performance.	Acceptance criteria: The model trains on colour video files and produces an accurate representation of a middle frame.
Reflection: The aim of this story was to reformat the model so that it could accept a sample with the dimensions of a typical video file. The original model trained on data that had dimensions that could go through 4 iterations of average pooling transformations. The dimensions of the video file are 360x504, which are not easily divided by 2^4, so padding was added to the input of the model to ensure that the samples could go through the pooling transformations without producing dimensions that would be different when the up-sampling transformations were applied.	

Sprint number: 7	Start date: 10/01/22
User story: As a user of the software, I would like the model to train faster so that I do not have to wait days for it to converge.	
Estimated effort: 2 hours	Actual effort: 3 hours
Task 28: Change data format to NCHW from NHWC.	Acceptance criteria: Data is in a format of 'channels-first', which is more suited to GPU training.
Task 29: Remove map function from dataset, save the data how it should be on the disk (minus data type)	Acceptance criteria: Data is stored on the disk in the correct format.
Reflection: The aim of this story was to speed up the training of the model, with TensorFlow not having to perform unnecessary transformations on the data, which would slow down the time-per-step in the training loop (5 hours of training time saved doing this). Also, changing the data format to NCHW is recommended by TensorFlow for GPU training.	

Sprint number: 7	Start date: 10/01/22
------------------	----------------------

User story: As a user of the software, I would like to be able to save the model weights to disk so that I do not have to train the model every time I run the program.	
Estimated effort: 0.5 hours	Actual effort: 1 hours
Task 30: Implement a solution for saving model parameters after training has completed.	Acceptance criteria: Model weights are saved to the disk in the "E:\Model" directory.
Reflection: The aim of this story was to ensure that model weights can be saved to the disk, so that useful training progress is not wasted, and the model can be loaded in a matter of seconds rather than days.	

24/01/22 – 06/02/22 Sprint 8 S2 Weeks 1 & 2

Sprint number: 8	Start date: 24/01/22
User story: As a user of the software, I would like to view an enhanced video with interpolated frames that increases the frame rate and provides a smoother looking video.	
Estimated effort: 3 hours	Actual effort: 2 hours
Task 31: Install scikit-video so that video files can be saved as an mp4 file to the disk and loaded from the disk.	Acceptance criteria: Individual frames can be loaded into memory using scikit-video.
Task 32: Write an algorithm that will interpolate predicted frames in between original frames of the video which also handles 'cuts'.	Acceptance criteria: Algorithm produces groups of three frames: Frame 1, Predicted Frame, Frame 3.
Task 33: Use scikit-video to write the enhanced video to the disk.	Acceptance criteria: Frames are saved to an mp4 file on the disk one by one.
Reflection: The aim of this story was to produce an algorithm to take the video frames into memory one by one, split them into sections that could fit through the model, predict the middle ground for each section, stitch the sections back together to create the full frames, then save the sequence to the disk. The difficulty of this was handling how to split the sections up regardless of their original size by calculating the smallest amount they could be split while also being smaller than the model dimensions.	

Sprint number: 8	Start date: 24/01/22
------------------	----------------------

User story: As the user of the software, I would like to be able to pass video files to the system so I can use them as a dataset.	
Estimated effort: 10 hours	Actual effort: 10 hours
Task 34: Use scikit-video to read the specified video file to be used as a dataset from the disk.	Acceptance criteria: The video is loaded frame by frame into memory.
Task 35: Ensure that the video file is in a format that can be sent through the model for training.	Acceptance criteria: The frames are passed through an algorithm in batches to group them into the correct dataset format.
Task 36: Run the complete system to evaluate the qualitative results of an enhanced complex video file.	Acceptance criteria: Software produces accurate results for complex video files.
<p>Reflection: The aim of this story was to pass the video to the system frame by frame and then split the video into sections that could fit through the network, then format the sections into samples and labels. One problem that arose with this is that if the original frame could not be divided exactly so that it would equal the network dimensions, the section would have to be padded which led the network to be trained on 'zero' data. If, in the future, a video were to be enhanced that could be split exactly without padding to fit into the network, the edges of the predicted frame would not be predicted correctly as the network is expecting to see 'zero' data rather than actual data.</p>	

07/02/22 – 20/02/22 Sprint 9 S2 Weeks 3 & 4

Sprint number: 9	Start date: 07/02/22
User story: As a user of the software, I would like to be able to train the model using a GUI to upload datasets and enhance a video.	
Estimated effort: 20.5 hours	Actual effort: 15 hours
Task 37: Install PyQT python library to build the GUI.	Acceptance criteria: PyQT is among the installed packages for the python environment.
Task 38: Reformat code to conform to an Object-Oriented style.	Acceptance criteria: Code is now structured as classes.
Task 39: Build a GUI window for uploading and enhancing the mp4 file.	Acceptance criteria: A simple GUI with loading bars for long-running processes.

Reflection: The aim of this story was to create the GUI for this project. This required that multithreading be implemented so that there could be a progress bar for long running tasks. It also required that the python code written for this project be rewritten in an Object-Oriented style. This had the effect of the model becoming much easier to be saved to the disk as this process required that custom layers/loss functions (the Charbonnier function) to be subclasses of the original layer/loss classes.

9.2 Model Summary

Model summary as outputted by using `tf.keras.Model.summary()`.

Layer (type)	Output Shape	Param #	Connected to
===			
input_1 (InputLayer)	(None, 6, 160, 320)	0	[]
conv2d (Conv2D)	(None, 32, 160, 320)	9440	['input_1[0][0]']
conv2d_1 (Conv2D)	(None, 32, 160, 320)	50208	['conv2d[0][0]']
average_pooling2d (AveragePooling2D)	(None, 32, 80, 160)	0	['conv2d_1[0][0]']
conv2d_2 (Conv2D)	(None, 64, 80, 160)	51264	['average_pooling2d[0][0]']
conv2d_3 (Conv2D)	(None, 64, 80, 160)	102464	['conv2d_2[0][0]']
average_pooling2d_1 (AveragePooling2D)	(None, 64, 40, 80)	0	['conv2d_3[0][0]']
conv2d_4 (Conv2D)	(None, 128, 40, 80)	73856	['average_pooling2d_1[0][0]']
conv2d_5 (Conv2D)	(None, 128, 40, 80)	147584	['conv2d_4[0][0]']
average_pooling2d_2 (AveragePooling2D)	(None, 128, 20, 40)	0	['conv2d_5[0][0]']
conv2d_6 (Conv2D)	(None, 256, 20, 40)	295168	['average_pooling2d_2[0][0]']
conv2d_7 (Conv2D)	(None, 256, 20, 40)	590080	['conv2d_6[0][0]']
average_pooling2d_3 (AveragePooling2D)	(None, 256, 10, 20)	0	['conv2d_7[0][0]']
conv2d_8 (Conv2D)	(None, 512, 10, 20)	1180160	['average_pooling2d_3[0][0]']
conv2d_9 (Conv2D)	(None, 512, 10, 20)	2359808	['conv2d_8[0][0]']
up_sampling2d (UpSampling2D)	(None, 512, 20, 40)	0	['conv2d_9[0][0]']
concatenate (Concatenate)	(None, 768, 20, 40)	0	['up_sampling2d[0][0]', 'conv2d_7[0][0]']
conv2d_10 (Conv2D)	(None, 256, 20, 40)	1769728	['concatenate[0][0]']
conv2d_11 (Conv2D)	(None, 256, 20, 40)	590080	['conv2d_10[0][0]']
up_sampling2d_1 (UpSampling2D)	(None, 256, 40, 80)	0	['conv2d_11[0][0]']

```

concatenate_1 (Concatenate)      (None, 384, 40, 80)  0
['up_sampling2d_1[0][0]',
                                                                    'conv2d_5[0][0]']

conv2d_12 (Conv2D)                (None, 128, 40, 80)  442496
['concatenate_1[0][0]']

conv2d_13 (Conv2D)                (None, 128, 40, 80)  147584
['conv2d_12[0][0]']

up_sampling2d_2 (UpSampling2D)    (None, 128, 80, 160)  0
['conv2d_13[0][0]']

concatenate_2 (Concatenate)      (None, 192, 80, 160)  0
['up_sampling2d_2[0][0]',
                                                                    'conv2d_3[0][0]']

conv2d_14 (Conv2D)                (None, 64, 80, 160)  110656
['concatenate_2[0][0]']

conv2d_15 (Conv2D)                (None, 64, 80, 160)  36928
['conv2d_14[0][0]']

up_sampling2d_3 (UpSampling2D)    (None, 64, 160, 320)  0
['conv2d_15[0][0]']

concatenate_3 (Concatenate)      (None, 96, 160, 320)  0
['up_sampling2d_3[0][0]',
                                                                    'conv2d_1[0][0]']

conv2d_16 (Conv2D)                (None, 32, 160, 320)  27680
['concatenate_3[0][0]']

conv2d_17 (Conv2D)                (None, 32, 160, 320)  9248
['conv2d_16[0][0]']

conv2d_18 (Conv2D)                (None, 3, 160, 320)  99

=====
===
Total params: 7,994,531
Trainable params: 7,994,531
Non-trainable params: 0
=====

```

9.3 Resource Links

GitHub Repository:

<https://github.com/HarryES95/COMP6013-ComputingProject>

Poster:

https://drive.google.com/file/d/1FWiLEaQNUUXSJgdc_jkJ3uilMPYxapPo/view?usp=sharing

Video Results:

https://drive.google.com/drive/folders/1aGin4mLuzRdeukR2PUHzGZLnP73yiIM_?usp=sharing

Product Backlog:

<https://docs.google.com/spreadsheets/d/12e1UTqzl-8uSb1fK5YhXv06Ww0MIF7Gf/e/dit?usp=sharing&oid=104075507556969975529&rtpof=true&sd=true>