

# 50.021 Artificial Intelligence Project Report

Kevin Yee (1002681), Nikos Chan (1002715), Samson Yu (1002819), Tan Wai Hong (1002894)

Singapore University of Technology and Design

## 1 Introduction

In this project, we explore the task of video frame interpolation by building upon a state-of-the-art (SOTA) model, the Video Frame Interpolation via Residue Refinement (RRIN) model [1] proposed by Haopeng Li et al.

Video frame interpolation is the task of synthesising 1 or more frames in the middle of 2 adjacent frames of a video. This provides temporal super-resolution in videos by generating smooth transitions.

Video frame interpolation is a common but technically difficult task used widely in film and video production. This technique can be used for increasing video frame rate [2], frame recovery in online video streaming or even virtual view synthesis [3].

Increasing the video frame rate is useful because capturing high frame rates on cameras is hard. Very fast shutter speeds to capture high frame rates reduces the exposure time and increases sensitivity in the same illumination which causes noise, especially in dark settings. Videos with high frames per second (FPS) can create a more fluid viewing experience or become slow motion videos. Generating adjacent frames is also useful for creating synthetic frames for virtual reality applications which tend to be computationally expensive.

## 2 Dataset

### 2.1 Vimeo-90k

First Frame	Middle Frame	Last Frame
		

Table 1: Example 3-frame sequence from the frame interpolation subset of Vimeo-90k.

The Vimeo-90k is a large-scale, high-quality video dataset built for 4 specific video processing tasks, namely video deblocking, video super-resolution, video denoising and frame interpolation [4]. It was created by Tianfan Xue et al. for evaluating their algorithm, ToFlow.

The dataset consists of 89,800 video clips downloaded from vimeo.com, and covers a large variety of scenes and actions. It is divided into 2 subsets: the triplet dataset for frame interpolation, and the septuplet dataset for the other 3 tasks mentioned above.

For our use case, we will be using the frame interpolation triplet subset. In this subset, 15,000 video clips from the Vimeo-90k dataset are selected, and processed into sequences of 3 frames, resulting in 73,171 3-frame sequences. The frames all have a fixed resolution of 256x448.

## 2.2 Data Processing

For the frame interpolation subset of Vimeo-90k, since the video clips have already been processed into 3-frame sequences, no further processing was done here to make it suitable for frame interpolation.

The train-test split has already been done for the Vimeo-90k dataset, with 51,312 triplets in the train set, and 21,859 in the test set. The folder paths of the triplets in the train set and test set are located in ``tri_trainlist.txt`` and ``tri_testlist.txt`` in the Vimeo-90k folder respectively.

We divided the train set of Vimeo-90k into a smaller train set and a validation set for model selection. The ratio of the split is 80:20. This split is done using PyTorch's *random\_split* function, and the generator has a *manual\_seed* of 42, to ensure that the training and validation results are more reproducible.

To load the dataset, we created a dataloader with PyTorch's Dataset and Dataloader classes. Since the dataset is too big to be completely loaded into the memory of SUTD HPC JupyterHub's server, our dataloader loads the file paths of each triplet, and only loads the frames during the training and validation loops. The dataloader shuffles both the training and validation sets.

For our image transforms, we use PyTorch's torchvision.transforms library. Specifically, we initially augmented the dataset using horizontal/vertical flipping, and temporal reversion of the triplets. Then, we converted the images to tensors, with their values between 0 and 1. These image transforms are the same as the ones done for the RRIN model. We follow its transforms, since we will be adapting the model for this project. The model will be explained in Section 3.

Due to the time constraints of this project, we decided not to augment the dataset, as each augmentation will increase the training time significantly. This will be explained in Section 4.1.

### 3 Model

#### 3.1 State-of-the-Art: Video Frame Interpolation via Residue Refinement

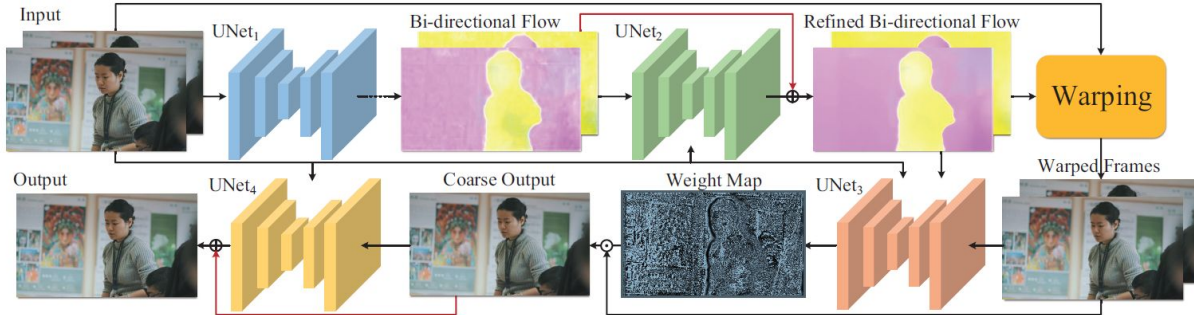


Figure 1: Video Frame Interpolation via Residue Refinement model architecture

Figure 1 shows RRIN's model architecture.

##### 3.1.1 U-Net

Sub-module	In Channel	Out Channel	Depth	#Parameters
UNet <sub>1</sub>	6	4	5	10.99
UNet <sub>2</sub>	10	4	4	2.73
UNet <sub>3</sub>	16	2	4	2.74
UNet <sub>4</sub>	9	3	4	2.73

Figure 2: The details of each U-Net in RRIN as shown in the original paper. The number of parameters is given in million.

RRIN makes heavy use of the U-Net [5] model. As shown in Figures 1 and 2, RRIN's model architecture includes 4 U-Nets.

The U-Net's name is related to its u-shaped architecture. It is a fully convolutional autoencoder with skip connections across its bottleneck layer to help retain useful features from earlier layers. The original U-Net uses convolutional layers with ReLU activations and max pooling layers in the contracting path, and up-convolutions and skip connection concatenations in the expanding path to form a symmetric shape. With high-resolution features and a bottleneck layer to learn salient features, the U-Net is good for tasks that involve the generation of high-definition transformed images, such as image denoising and optical flow estimation.

In RRIN, some modifications are made to the U-Net. The ReLU activations in the contracting path are replaced by Leaky ReLUs, which should help to reduce the "dying ReLU" problem [6]. The number of filters is reduced to 32 from 64 in the original U-Net. Other than UNet<sub>1</sub>, which has

a depth of 5 like the original U-Net, the other U-Nets only have a depth of 4 to increase training and inference speeds.

The two input frames are also passed into each U-Net to help preserve high-resolution features from the input frames and thus generate high-quality outputs.

### 3.1.2 Optical Flow Estimation

In many video processing tasks, optical flow estimation is useful to account for motion that may arise from the additional temporal dimension, and is a huge area of research on its own. Compared to using full image representations to calculate motion, optical flow estimation is more efficient and effective as it reduces redundancy and makes it easier to generate high-quality intermediate images [7]. Optical flow works on the individual pixel level. It tries to estimate how each pixel’s colour/intensity moves on the image plane over time, usually between consecutive frames. There are many handcrafted methods to obtain different forms of optical flow, such as the Lucas-Kanade method [8] for dense optical flow.

Most importantly, optical flow works on the assumptions that the pixel intensities of an object do not change between consecutive frames, and that neighbouring pixels have similar motion [8].

In RRIN, this optical flow is automatically estimated through optical flow warping and gradient updates. The authors use PyTorch’s *grid\_sample* function to warp the input images with the outputs from UNet<sub>2</sub>. This ensures that PyTorch’s backpropagation and hence gradient updates work to help UNet<sub>1</sub> and UNet<sub>2</sub> learn to generate optical flow estimates with images as inputs.

Two optical flow estimates are generated from both U-Nets. The first estimates from UNet<sub>1</sub> accounts for the estimated optical flow from both input frames to each other. With variable  $t \in (0, 1)$  as the interval value between the two input frames to account for arbitrary-time estimates, the authors get the approximate flows between the two input frames and the interpolated frame through bidirectional interpolation [7].

The authors set  $t = 0.5$  by default for their model, since the interpolated frame would usually be the midpoint of the two input frames in temporal terms. However, this value can be changed to generate interpolated frames at different timesteps, as will be shown in Section 4.5.2.

### 3.1.3 Weight Map Generation

Model	Vimeo90K		UCF101	
	PSNR	SSIM	PSNR	SSIM
w/o-RR	34.84	0.9625	34.58	0.9488
w/o-WM	34.90	0.9615	34.83	0.9486
RRIN	35.22	0.9643	34.93	0.9496

Figure 3: The positive effects of the residue refinements, RR, and weight maps, WM, on two evaluation metrics, peak signal-to-noise ratio and structural similarity index measure, as shown in the original paper.

After the optical flow estimation and warping in Section 3.1.2, the outputs are passed into another U-Net, UNet<sub>3</sub>, and a sigmoid activation function, to generate two weight maps. These weight maps are used to capture information about lighting change and occlusion, which are important for generation of realistic interpolated frames.

They are important because of the optical flow assumptions stated in Section 3.1.2, where pixel intensities of an object do not change between consecutive frames, and neighbouring pixels have similar motion. If the intensities change or neighbouring pixels are not present, the weight map can help to check for any lighting change or occlusion, and help to fill in the gaps in the estimations.

Figure 3 shows how the weight maps help to improve the model performance in terms of the peak signal-to-noise ratio (PSNR) and the structural similarity index measure (SSIM) for the Vimeo-90k dataset and UCF101 [9], an action recognition dataset. We will explain these two popular evaluation metrics for image comparisons in Section 4.3. In general, a higher score on the PSNR or the SSIM means that two images are more similar. In this case, this means that the model predictions are more accurate.

### 3.1.4 Residue Refinement

The authors of RRIN propose residue refinement through a U-Net as a way to improve details in the outputs from a preceding U-Net.

The red lines in Figure 1 represent the two residue refinements that are done by the authors in the model architecture. UNet<sub>2</sub> learns the residues of the arbitrary-time optical flow estimates that are generated from UNet<sub>1</sub>. These residues are then added to the original outputs from UNet<sub>1</sub>, which supposedly helps to make the final optical flow estimates smoother in local regions.

These calculations are marked by these equations:

$$\begin{aligned} F_{t \rightarrow 0} &= \hat{F}_{t \rightarrow 0} + \bar{F}_{t \rightarrow 0}, \\ F_{t \rightarrow 1} &= \hat{F}_{t \rightarrow 1} + \bar{F}_{t \rightarrow 1}. \end{aligned}$$

$F_{t \rightarrow 0}$  and  $F_{t \rightarrow 1}$  represent the final flow estimates, while  $\hat{F}_{t \rightarrow 0}$  and  $\hat{F}_{t \rightarrow 1}$  represent the outputs from UNet<sub>1</sub>, and  $\bar{F}_{t \rightarrow 0}$  and  $\bar{F}_{t \rightarrow 1}$  represent the residues learned for  $\hat{F}_{t \rightarrow 0}$  and  $\hat{F}_{t \rightarrow 1}$  respectively.

For the other residue refinement, UNet<sub>4</sub> learns the residues of the coarse interpolated output frame  $\hat{I}_c^t$ , and uses that to overcome its loss of texture and clear edges. The equation here is:

$$\hat{I}^t = \hat{I}_c^t + \vec{I}^t.$$

$\hat{I}^t$  represents the final output of the model and  $\bar{I}^t$  represents the residues of the coarse output  $\hat{I}_c^t$ .

Figure 3 shows how the residue refinements help to improve the model performance in terms of PSNR and SSIM for the Vimeo-90k dataset and UCF101.

### 3.1.5 Adam Optimiser

Our model was trained using the mini-batch Adaptive Moment Estimation (Adam) [10] optimiser, like the original RRIN model. For our custom RRIN, we set our  $\beta_1$  and  $\beta_2$  values as 0.9 and 0.999, which are both values used by the original paper and commonly accepted defaults.

The Adam optimiser uses the squared gradients to scale the learning rate like RMSprop and takes advantage of momentum by using the moving average of the gradient instead of the gradient itself like SGD with momentum. This method is very fast and helps with rapid convergence. It also rectifies decaying gradients and high variance errors. However, it is computationally expensive as it has to calculate more values.

### 3.1.6 Loss Function

Pixel-wise loss functions such as mean square error (MSE) loss and mean absolute error (MAE) loss are frequently used for training image synthesis models. MSE loss gives higher weightage to large errors which is useful when large deviations are undesirable while MAE loss takes the absolute difference equal to the observed deviation.

The original RRIN paper uses the Charbonnier loss function [11], which is popular in flow and depth estimation tasks that require robustness such as this video interpolation task. This is because it tends to encourage global smoothness and robustness to outliers during the training process. The mathematical formula for the loss function is as follows:

$$\beta(\hat{I}^t, I^t) = \frac{1}{|P|} \sum_{p \in P} \sqrt{(\hat{I}^t(p) - I^t(p))^2 + \varepsilon^2},$$

where  $P = [1, H] \times [1, W] \times [1, 3]$ , and  $\varepsilon$  is set to  $1e-6$  empirically during training.

For our custom RRIN, we use the MSE loss as it is one of the most commonly used losses in regression, and is also one of the most common ways to measure the degree of similarity between images. Since our method is more of an exploratory one, we decided to start off with a more vanilla loss function.

## 3.2 Exploratory Changes

### 3.2.1 Why Change?

As mentioned in Section 3.1.4, residue refinements are proposed by the authors to improve the details and quality of generated outputs. Figure 3 shows that this proposed method does indeed work to improve image generation quality and accuracy.

	SoftSplat [12]	<b>RRIN</b>	DAIN [11]	MEMC-Net [13]
#Parameters (millions)	-	<b>19.19</b>	24.02	70.31
Runtime (seconds)	-	<b>0.08</b>	0.13	0.12

Table 2: Number of parameters in millions and runtime for the top 4 state-of-the-art frame interpolation models.

However, one limitation in RRIN is the computationally expensive use of U-Nets in obtaining the residue refinements. As shown in Figure 2, RRIN’s UNet<sub>2</sub> and UNet<sub>4</sub> both contain 5.47 million parameters each. This adds to a total of 19.19 million parameters in RRIN, as shown in Table 2, negatively affecting both its training time and runtime.

The runtime in Table 2 is calculated in seconds for one prediction using images that are 640 pixels wide and 480 pixels high. Let us assume that 24 FPS is the minimum speed needed to capture video while still maintaining realistic motion [14] and we want to double the frame rate of a low FPS 10-minute video from 12 FPS to 24 FPS. On RRIN, the fastest SOTA model, this would take around 9.6 minutes. While this is already a very good performance, we believe it is still a significant amount of time that can be further reduced.

All the other SOTA frame interpolation models in Table 1 have higher runtimes than RRIN. Since SoftSplat’s code is not publicly available, we are unable to determine its exact parameter count and runtime. This means that if we can improve RRIN’s runtime, we would have achieved a lower runtime than the current SOTA models, excluding SoftSplat.

### 3.2.2 Discriminator Network

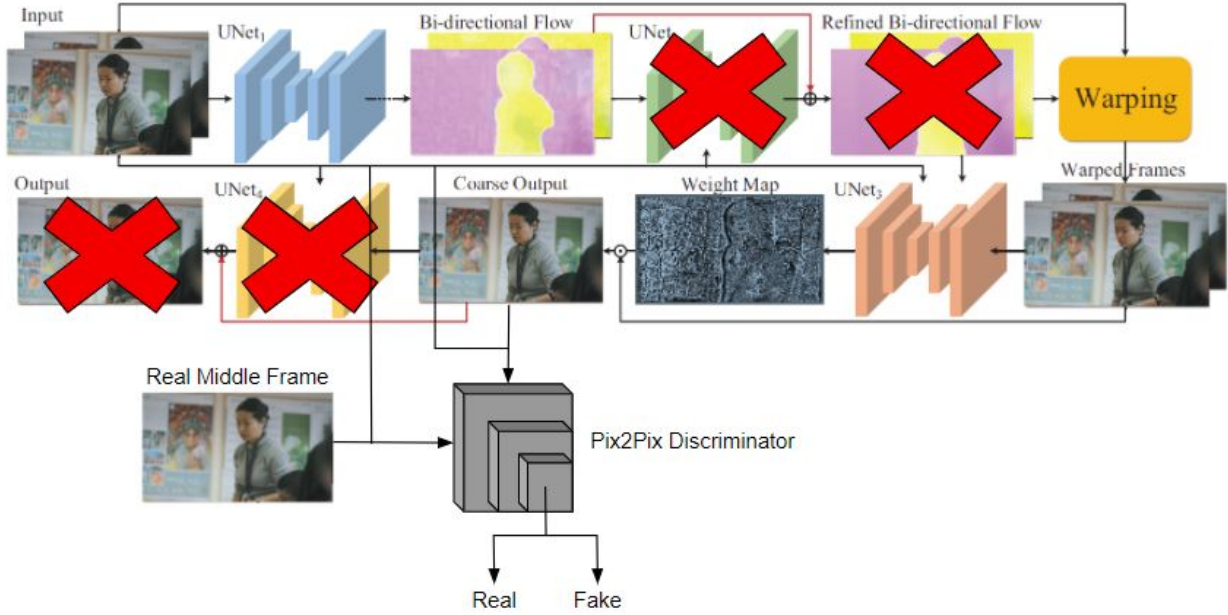


Figure 4: Our proposed changes to the original RRIN, which includes the addition of a discriminator network and the removal of the residue refinement U-Nets.

As a way of reducing RRIN’s runtime while possibly maintaining the same high-quality outputs, we propose the replacement of the residue refinement U-Nets in RRIN with a discriminator from the Pix2Pix [15] generative adversarial network (GAN).

Since the invention of the GAN by Goodfellow et al. [16], GANs have been used for image super-resolution [17], generating realistic human faces (i.e. deepfakes) and realistic style transfer [18]. These use cases hint at the capabilities of GANs in generating high-quality images. The typical loss functions (e.g. MSE and MAE losses) for image reconstruction tasks can only feedback the pixel-wise differences between the generated samples and the real samples but it does not tell the generator whether the generated samples are realistic or not. GANs solves this by introducing a discriminator network whose output serves as a metric to determine how realistic the prediction is. The discriminator’s output is fed to the generator to guide it towards weights that produce more realistic results. Hence, we believe that adding a discriminator network can help us tackle the same issue as the residue refinement U-Nets in the RRIN: lack of details and low-quality generated samples.

The Pix2Pix network was chosen because it has been a general-purpose solution to image-to-image translation problems and has been used across different applications that require high-quality outputs, such as image colourisation and next-frame prediction. The Pix2Pix network consists of an U-net generator, like the ones in our RRIN model, and a PatchGAN discriminator. Typically, discriminators take in an image from the original dataset and a generated image and predict the likelihood of whether the image is real or generated. The PatchGAN discriminator works in a similar way, but aims to classify  $N \times N$  patches of images as



real or fake rather than the entire image before averaging all the responses to provide the ultimate output. Supposedly, this penalises structures at the scale of patches that enforces more constraints, encouraging sharp details. For the PatchGAN discriminator, we used the Adam optimiser and its original beta values in the Pix2Pix model:  $\beta_1 = 0.5$  and  $\beta_2 = 0.999$ .

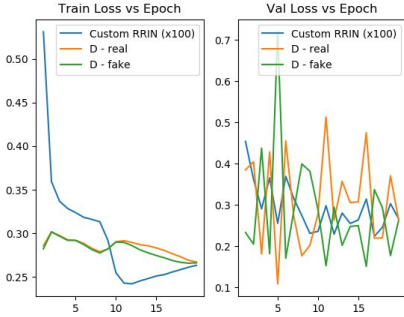
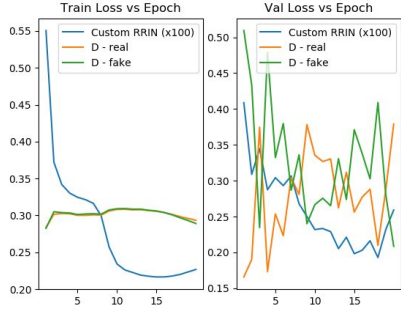
Another hyperparameter we have to set is the weightage between the custom RRIN's MSE loss and the discriminator's adversarial loss used to optimise it. Although the original Pix2Pix model uses the MAE loss instead of our MSE loss, we mostly followed their ratio and set our weightage between the MSE loss and the adversarial loss to 0.999 and 0.001 respectively.

We also removed the batch normalisation layers from the PatchGAN discriminator. This is because we follow the original RRIN model closely and use small batch sizes, as will be further discussed in Section 4.2. Batch normalisation generally does not work well for small batch sizes [19] since it needs a sufficiently large batch size to compute accurate mini-batch statistics.

Finally, the other advantage of using a discriminator network is that it is detachable. This detachability means that the inference time can be significantly lower than the training time, since we would not have to do a forward pass through the discriminator. As mentioned in Section 3.2.1, the inference time of the RRIN is quite significant, even if it is the fastest (possibly with the exception of SoftSplat) among the SOTA methods. We hope this discriminator addition can achieve similarly high-quality results as compared to RRIN, with lower inference time.

## 4 Results

### 4.1 Train and Validation Losses

	Batch Size = 4	Batch Size = 8
<b>Learning Rate = <math>1e-5</math></b>	 <p>Lowest validation loss = 0.224</p>	 <p>Lowest validation loss = 0.193</p>

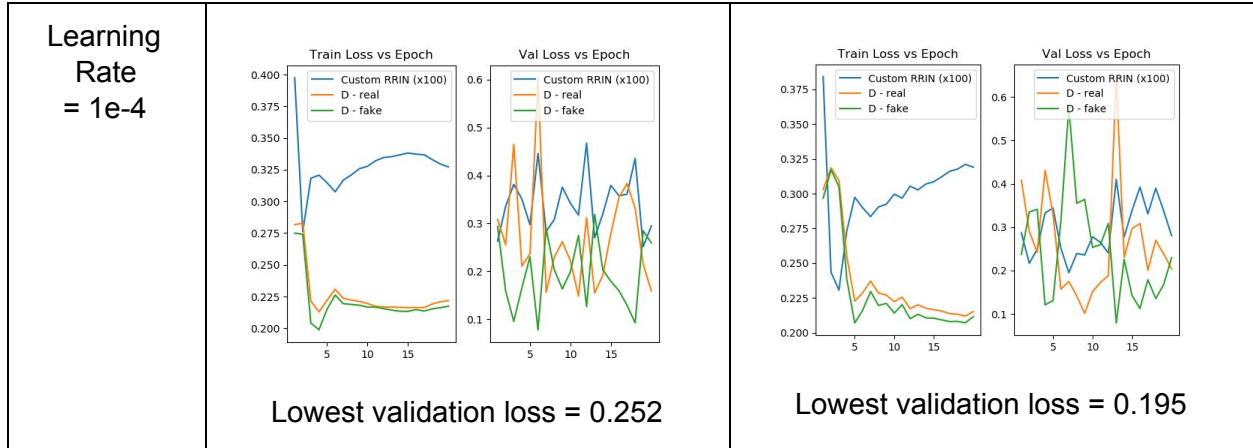


Table 3: Training and validation loss plots

Due to the significant dataset and model sizes, 1 epoch consisting of the training and validation loops takes around 3.5 hours on 1 NVIDIA Tesla V100 if we use the augmented dataset and a batch size of 4. With the unaugmented dataset, the time taken falls to around 1 hour, which is more reasonable with our time constraint for this project.

Hence, we train our model for 20 epochs on 4 hyperparameter settings with the unaugmented dataset, and save the training and validation losses every epoch. For each of the 4 experiments, we save the model with the lowest validation loss for the custom RRIN.

In Table 3, the blue lines labelled “Custom RRIN” are our custom RRIN’s loss, the orange lines labelled “D - real” are our discriminator’s loss on real images, and the green lines labelled “D - fake” are our discriminator’s loss on images generated by the custom RRIN. Since we use the MSE loss for the custom RRIN and BCE loss for the PatchGAN discriminator, we multiply the MSE loss by 100 in the graphs to make it easier to visualise them together.

For the train losses across all 4 experiments, the custom RRIN generator’s performance has an inverse relationship with the discriminator, as expected, since they have an adversarial relationship. However, there are slight differences between the train loss curves of the two different learning rates.

For the learning rate of  $1e-4$ , the custom RRIN generator’s training loss has a huge initial dip below 0.3 before sharply increasing above 0.3. While the generator’s training loss increases, the discriminator’s training loss on both the real and fake images rapidly decreases to a low number. This shows that the discriminator’s strength has dominated over the generator causing the generator’s training to fail due to vanishing gradients [22]. This shows that the learning rate might be too high because there is no training stability for the generator which is reflected in the higher validation losses of the generator.

For the learning rate of  $1e-5$ , the custom RRIN generator’s training loss decreases during the training process and experiences a period of training stability at about 0.25 for batch sizes of 4

and 0.22 for batch sizes of 8. This period of training stability is reflected in the lower validation losses of the generator.

Hence, since a lower learning rate of 1e-5 and a higher batch size of 8 provided better training stability as well as lower training and validation loss for the RRIN generator, we chose lr=1e-5 and bs=8 as our final hyperparameters.

We also noted that training stability may degenerate into periods of high-variance loss and corresponding lower quality generated images. Hence, we saved the weights of the best performing model with the lowest validation loss.

## 4.2 Hyperparameter Settings

The original RRIN uses a batch size of 4 and learning rate of 1e-4. We use the same hyperparameters, and vary them slightly. Specifically, we also tried a batch size of 8 and learning rate of 1e-5 for a total of 4 experiments, as shown in Table 3.

In the interest of time, we did not tweak other hyperparameters, since the 4 experiments above already took around 70 hours to train in total.

## 4.3 Evaluation Metrics

$$MSE(y_i, g_i) = \frac{1}{n} \sum_{i=0}^n (y_i - g_i)^2,$$

where  $y_i$  is the original image,  $g_i$  is the generated image, and  $n$  is the number of image samples.

MSE is the most common estimator of image quality measurement metric that measures the average of the square of the errors, which is the pixel-wise difference between the original and generated image.

$$PSNR = 10 \log_{10}(peakval^2/MSE),$$

where  $peakval$  is the maximal in the image data values.

The PSNR is a variation of MSE and also focuses on pixel-wise comparison between the original and generated image. The higher the PSNR, the better the quality of the reconstructed image. It also measures the approximate estimation to human perception of reconstruction quality which is useful to evaluate the interpolated frame in this task.

While image quality assessment metrics such as MSE and PSNR are frequently used because they are simple to calculate and clear in their physical meaning, they may perform poorly in discriminating the visual quality and structural content in images. Hence, we also used another

metric, SSIM, that estimates the perceived quality of images based on the human visual system by modelling the loss of correlation, luminance distortion, and contrast distortion [20].

In our implementation, we use the *structural\_similarity* function in the *skimage.metrics* library to match the implementation of SSIM for RRIN.

## 4.4 Comparisons

In this section, we compare the performance of our custom model on Vimeo-90k’s test set, with SOTA models.

### 4.4.1 Our Custom RRIN

	Learning Rate = 1e-4		Learning Rate = 1e-5	
	Batch Size = 4	Batch Size = 8	Batch Size = 4	Batch Size = 8
PSNR	30.76	30.45	30.40	<b>30.55</b>
SSIM	0.907	0.900	0.907	<b>0.912</b>

Table 4: PSNR and SSIM for our 4 models trained with 4 hyperparameter settings.

From our results, it seems like the validation loss (in Section 4.1) does not translate directly to the two popular evaluation metrics, PSNR and SSIM.

Since the model with learning rate = 1e-5 and batch size = 8 has the lowest validation loss, the second highest PSNR and the highest SSIM, we decided that it is the best model overall. In the next few sections, we will use it as our best model to compare against other SOTA methods and show our results.

### 4.4.2 State-of-the-Art Methods

	<b>Our Best Model</b>	SoftSplat	<b>RRIN</b>	DAIN	MEMC-Net	Linear Interpolation
PSNR	<b>30.55</b>	36.10	<b>35.22</b>	34.71	34.40	25.60
SSIM	<b>0.912</b>	0.970	<b>0.964</b>	0.964	0.962	0.753

Table 5: PSNR and SSIM comparisons with state-of-the-art methods.

In the rightmost column of Table 5, we show the PSNR and SSIM of the naive linear interpolation method to serve as a baseline. In linear interpolation, we survey the pixels in the first frame, and their corresponding pixels in the same position and channel in the third frame. Then, we use their midpoint values in the same position and channel in the interpolated middle frame. We use SciPy’s *scipy.interpolate.interpn* function.

In Table 5, we also include 4 SOTA methods which currently have the best results in terms of PSNR and SSIM for frame interpolation.

## 4.5 Generated Frames

### 4.5.1 Vimeo-90k Test Set: Comparisons with Other Methods for $t = 0.5$






Real First Frame	Real Middle Frame	Real Last Frame
		

Table 6: Frame interpolation for Vimeo-90k triplet `00001/0830`.

The three frames in Table 6 are from sequence `00001/0830` in the Vimeo-90k test set.

	Interpolated Middle Frame ( $t = 0.5$ )
<b>Our Best Model</b>	
SoftSplat	


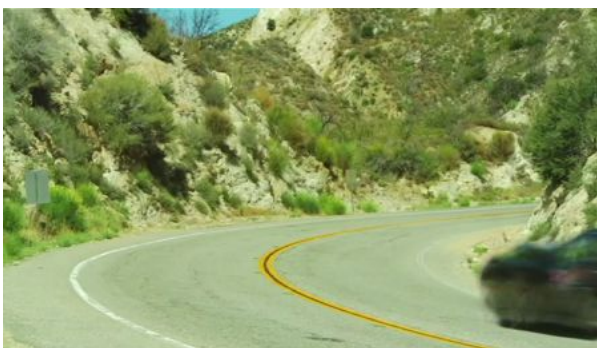

RRIN	
DAIN	
MEMC-Net	
Linear Interpolation	

Table 7: Generated interpolated middle frames for Vimeo-90k sequence `00001/0830` by our best model and the other methods in Section 4.4.2.



#### 4.5.2 Vimeo-90k Test Set: Generate with Varying $t$ Values









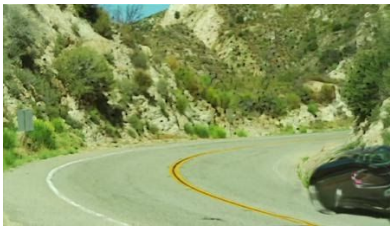


Real First Frame	Real Middle Frame	Real Last Frame
		
$t = 0.1$	$t = 0.2$	$t = 0.3$
		
$t = 0.4$	$t = 0.5$	$t = 0.6$
		
$t = 0.7$	$t = 0.8$	$t = 0.9$
		

Table 8: Frame interpolation for Vimeo-90k triplet `00001/0830` when  $t$  values vary.

#### 4.5.3 Vimeo-90k Test Set: Estimated Optical Flows and Weight Maps for $t = 0.5$

	Estimated Optical Flow	Weight Map
--	------------------------	------------

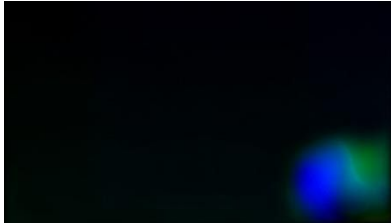

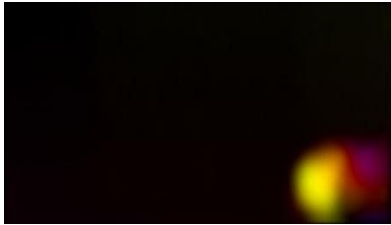
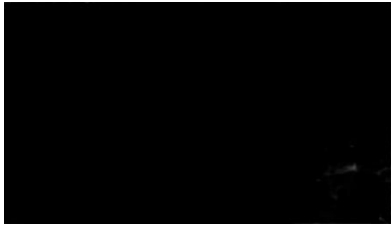
$t \rightarrow 0$		
$t \rightarrow 1$		



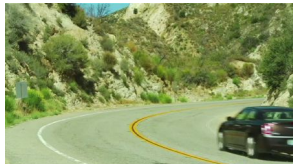
Table 9: Estimated optical flows and weight maps generated by our best model, at  $t = 0.5$ , for Vimeo-90k triplet `00001/0830`.

As expected, the optical flow estimates are seen in the bottom right corner of the frame, where the only moving object, the car, is located. The weight maps that help with lighting and occlusion are also focused on the car.

The visualisations in Table 9 are based on the HSV model. In this particular case, the saturation is set to the maximum, so only the hue and value can vary. Different hues mean different pixel motion directions, and brighter colours mean a greater magnitude of motion in the visualisations. The model returns an optical flow estimate and a weight map for two directions,  $t$  to the first frame (timestep 0) and  $t$  to the last frame (timestep 1). This bidirectionality allows for the generation of more temporally congruent and thus realistic interpolated frames.

#### 4.5.4 Optical Flow Assumption

As mentioned in Section 3.1.2, most optical flow estimates work on the assumption that the pixel intensities do not change too drastically between consecutive frames.

		Real First Frame	Interpolated Middle Frame ( $t = 0.5$ )	Real Last Frame
Pixel Value Increase	0			











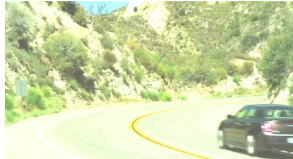
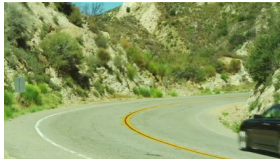

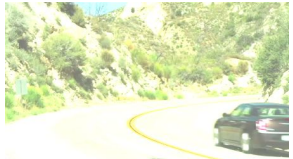
	25			
	50			
	75			
	100			

Table 10: Frame interpolation for Vimeo-90k triplet `00001/0830` when pixel intensities change.

In Table 10, we show the results of our custom RRIN model after testing it on the same Vimeo-90k triplet `00001/0830`, while increasing the pixel intensities by 4 different amounts: 25, 50, 75 and 100. As we can see, the assumption that the pixel intensities do not change too drastically between consecutive frames to account for accurate motion is indeed important for optical flow estimations. The quality of the interpolated middle frame consistently falls as the image brightness/pixel intensity increases.

#### 4.5.5 Happiness Facial Expression







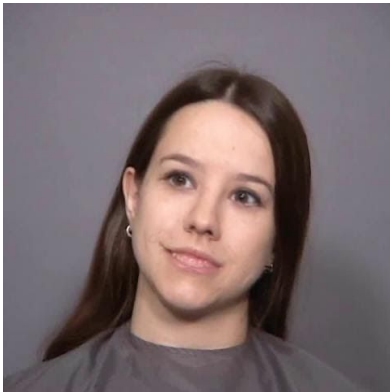
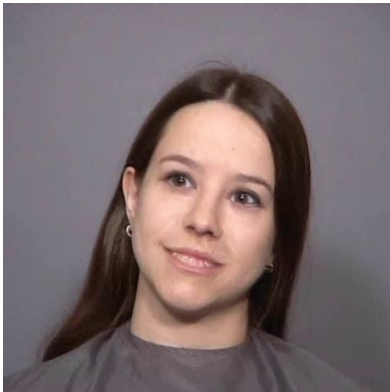
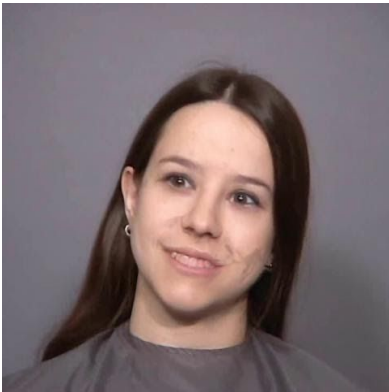

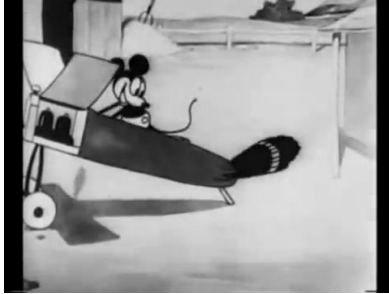
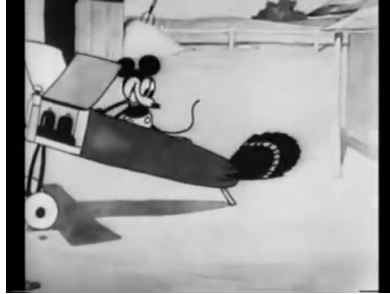



Real First Frame	Real Middle Frame	Real Last Frame
		
$t = 0.1$	$t = 0.2$	$t = 0.3$
		
$t = 0.4$	$t = 0.5$	$t = 0.6$
		
$t = 0.7$	$t = 0.8$	$t = 0.9$



Table 11: Frame interpolation for a facial expression of happiness.

In Table 11, we generated the interpolated frame on a “Happiness” sample from the Human ID Project [21] at the The University of Texas at Dallas. With just the first and last frame provided to the model, the interpolated results from  $t = 0 \rightarrow 1$  in timesteps of 0.1 work surprisingly well with realistic intermediate frames.

#### 4.5.6 Sample Generation from the Movie “Plane Crazy” by Walt Disney

Real First Frame	Real Middle Frame	Real Last Frame
		
$t = 0.1$	$t = 0.2$	$t = 0.3$
		
$t = 0.4$	$t = 0.5$	$t = 0.6$

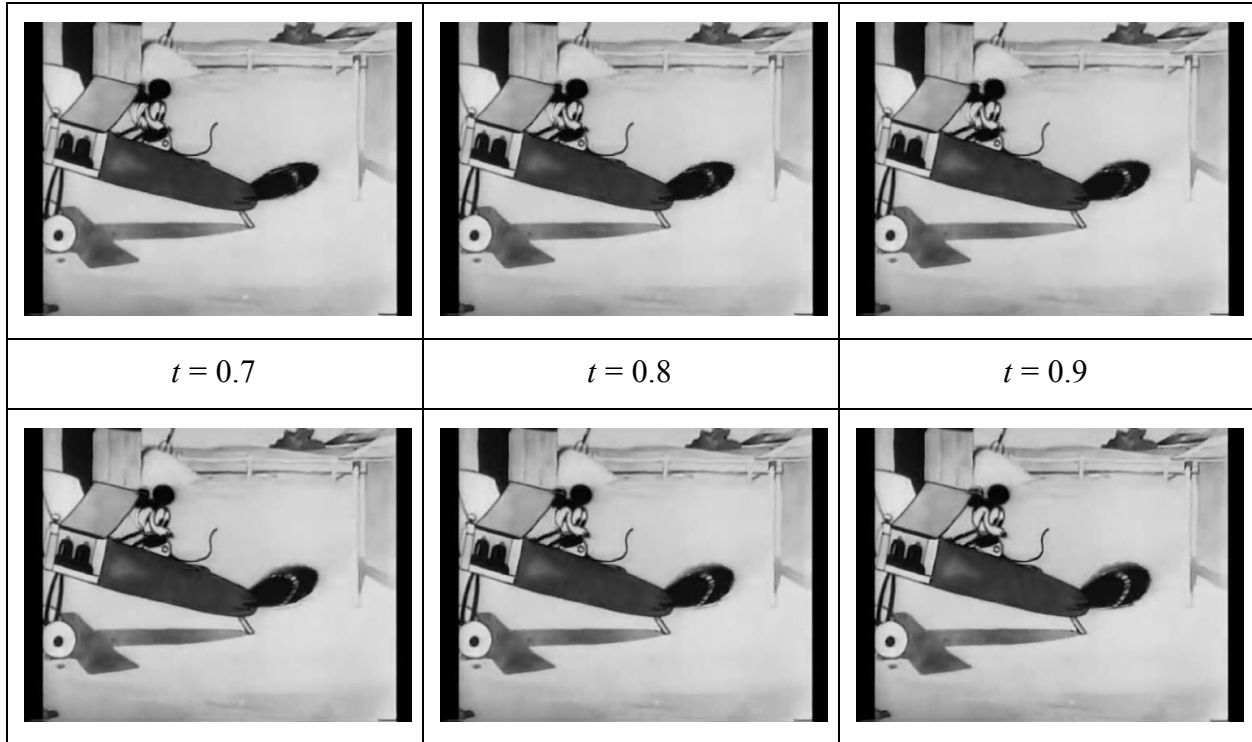


Table 12: Frame interpolation of sample frames in the movie “Plane Crazy” by Walt Disney.

In Table 12, we show the results we have obtained from using our model to increase the FPS of an older video with low FPS, which is a common use case of frame interpolation. If you observe how the **tail of the plane** changes relatively smoothly, this initial qualitative results show that our model works decently well, even in this grayscale setting which our model is not trained on.

#### 4.5.7 Sample Generation from the Music Video for “Territory” by The Blaze

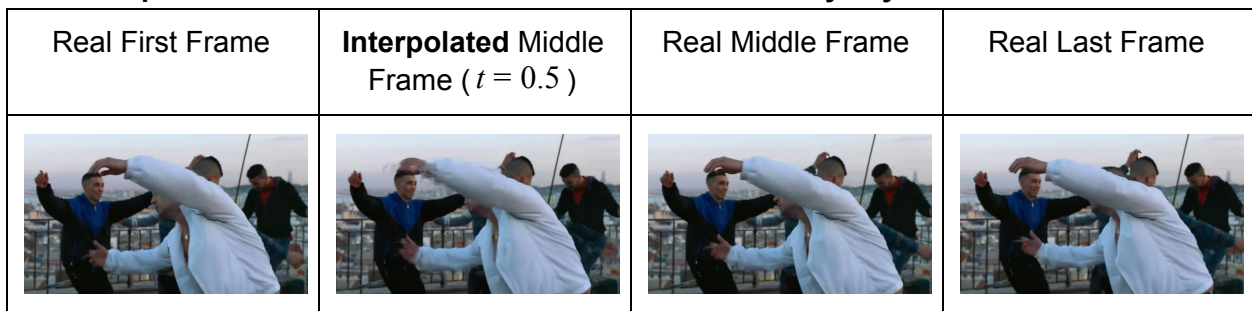


Table 13: Sample taken from video frame interpolation of the music video for the song “Territory” by The Blaze.

In Table 13, we have generated an interpolated frame for the music video of the song “Territory” by The Blaze. We also extended this generation to a very short portion of the music video, along with a similarly short portion of the music video of the song “Palm of My Hand” by ZHU. The full



video is available at this link:

[https://drive.google.com/file/d/1TpMJn5s-1EX0\\_kDmWvkUTts4vDj18llo/view?usp=sharing](https://drive.google.com/file/d/1TpMJn5s-1EX0_kDmWvkUTts4vDj18llo/view?usp=sharing).

## 5 Code

Other than the ZIP file submitted along with this report, our code (excluding the GUI) will also be available at <https://github.com/SamsonYuBaiJian/video-interpolation/>.

The features available and changes made to the original RRIN in our code are shown in the README.

## 6 Graphical User Interface

The GUI demonstrates the ability of our model to get the interpolated frame in between two frames. The operation is quite simple. Select the 2 target frames, then upload them to the server in order and press the “Process” button. After a short while, the interpolated frame will be shown.

The interpolated frame can be animated so that it appears in between the original frames of the video, just like a slowed-down version of the original video. This helps evaluation of the visual performance of the model based on seeing what the output looks like in context.

Behind the scenes, the two selected frames are uploaded to a small Flask server which ensures the images have the same resolution and then preprocesses the images to ensure that they are padded to a multiple of 64 in both dimensions. The images are fed through a pretrained model and the output is sent back to the web browser, where it is visualised. Since we are only doing inference and not training, GPU compute is not used for this.

The optical flow estimates and weight maps may be viewed superimposed on each input frame as well, to see how well they correlate to the motion happening between the frames.

Some configuration options are available:

- $t$  controls the interpolation timestep, and is a number in the interval (0,1),
- downsample controls whether the images are resized so that the largest dimension does not exceed 512 pixels,
- outformat controls the output file format of the interpolated image.

The downsample option sacrifices visual fidelity for lower time and memory cost. Interpolating between very large frames (like 1920 x 1080 resolution) on the online demo may fail due to memory constraints.

You may interact with our model online at <https://ai-demo.lepak.sg/> or clone the repository at <https://github.com/nik0sc/ai-demo> to run the demo offline (which is likely to be faster).

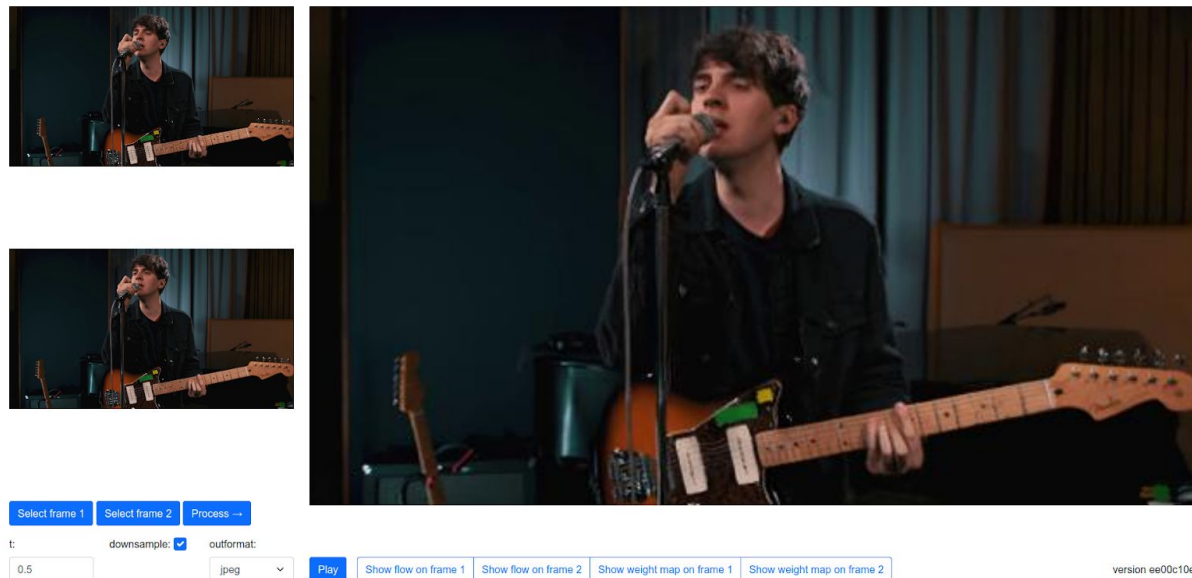


Figure 5: A screenshot of our GUI.

## References

- [1] H. Li, Y. Yuan and Q. Wang, "Video Frame Interpolation Via Residue Refinement." *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 2020*, pp. 2613-2617, doi: 10.1109/ICASSP40776.2020.9053987.
- [2] Xia, M., Yang, G., Li, L. et al. "Detecting video frame rate up-conversion based on frame-level analysis of average texture variation." *Multimed Tools Appl* 76, 8399–8421 (2017). <https://doi.org/10.1007/s11042-016-3468-1>
- [3] Flynn, John, et al. "Deepstereo: Learning to predict new views from the world's imagery." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.
- [4] Xue, Tianfan, et al. "Video enhancement with task-oriented flow." *International Journal of Computer Vision* 127.8 (2019): 1106-1125.
- [5] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *International Conference on Medical image computing and computer-assisted intervention*. Springer, Cham, 2015.
- [6] Lu, Lu, et al. "Dying relu and initialization: Theory and numerical examples." *arXiv preprint arXiv:1903.06733* (2019).

- [7] Jiang, Huaizu, et al. "Super slo-mo: High quality estimation of multiple intermediate frames for video interpolation." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.
- [8] Lucas, Bruce & Kanade, Takeo. (1981). An Iterative Image Registration Technique with an Application to Stereo Vision (IJCAI). [No source information available]. 81.
- [9] Soomro, Khuram, Amir Roshan Zamir, and Mubarak Shah. "UCF101: A dataset of 101 human actions classes from videos in the wild." *arXiv preprint arXiv:1212.0402* (2012).
- [10] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).
- [11] Bao, Wenbo, et al. "Depth-aware video frame interpolation." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019.
- [12] Niklaus, Simon, and Feng Liu. "Softmax Splatting for Video Frame Interpolation." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.
- [13] Bao, Wenbo, et al. "Memc-net: Motion estimation and motion compensation driven neural network for video interpolation and enhancement." *IEEE transactions on pattern analysis and machine intelligence* (2019).
- [14] Brunner, D. (2017, March). Frame Rate: A Beginner's Guide: Blog: TechSmith. Retrieved from <https://www.techsmith.com/blog/frame-rate-beginners-guide/>
- [15] Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.
- [16] Goodfellow, Ian, et al. "Generative adversarial nets." *Advances in neural information processing systems*. 2014.
- [17] Ledig, Christian, et al. "Photo-realistic single image super-resolution using a generative adversarial network." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.
- [18] Karras, Tero, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019.
- [19] Lian, Xiangru, and Ji Liu. "Revisit Batch Normalization: New Understanding from an Optimization View and a Refinement via Composition Optimization." *arXiv preprint arXiv:1810.06177* (2018).
- [20] Sara, Umme & Akter, Morium & Uddin, Mohammad. (2019). Image Quality Assessment through FSIM, SSIM, MSE and PSNR—A Comparative Study. *Journal of Computer and Communications*. 07. 8-18. 10.4236/jcc.2019.73002.
- [21] A. J. O'Toole et al., "A video database of moving faces and people," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 5, pp. 812-816, May 2005, doi: 10.1109/TPAMI.2005.90.

[22] G. (2020). Common Problems. Retrieved August 16, 2020, from <https://developers.google.com/machine-learning/gan/problems>