Computer Vision Image Enhancement

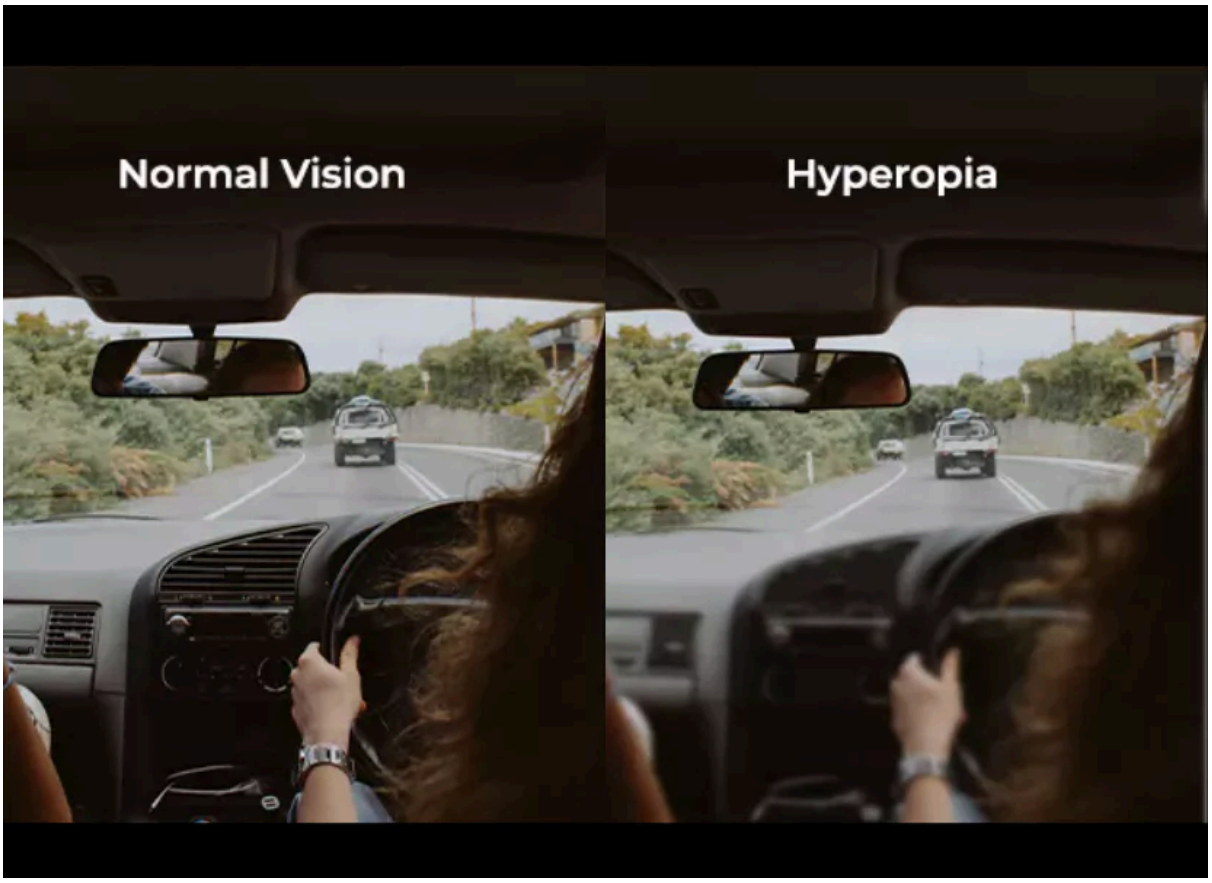
Code

By: Youssef Abdelsalam

Introduction:

Abstract:

The goal of this project is to build a computer vision system that can accurately enhance images and videos into a version that hyperopic people can see. Leveraging computer vision and deep learning, the proposed solution aims to improve virtual reality (VR) experiences for individuals with hyperopia. By developing an image enhancement algorithm, the study addresses limitations of current solutions like headset brackets or prescription inserts. Despite hardware constraints, promising results are achieved, with training and testing accuracies reaching 79.9% and 81.1%, respectively. This approach not only improves VR accessibility but also demonstrates potential for broader applications beyond visual impairment challenges.



Problem:

According to the WHO, more than 2.2 billion people or approximately 27% of the global population have a near or distance vision impairment.[1] Of those 2.2 billion, at least 1 billion of these cases could have prevented or has yet to be addressed.

Hyperopia or farsightedness is a very common vision condition, where you can see distant objects clearly, but objects nearby may be blurry. So people with hyperopia can't see can't see things that are close to their eyes. One of these things are VR (virtual reality) and AR (augmented reality) headsets.

The VR market has grown tremendously in the past years and is expected to growa lot more in the future. The global VR market was estimated at USD 59.96 billion in 2022 and is expected to grow at a compound annual growth rate of 27.5% from 2023 to 2030.[2] That would be USD 328 billion in 2030.

People with hyperopia have an easier time seeing objects that are at least 6 meters away.[3] Usually, VR headsets, even though are a few inches away for your eyes, use the HMD (head-mounted displays) to "trick" the brain into thinking that the screen is 1.3 to 2 meters away. This means that for people with hyperopia don't see very well when using VR headsets.

Existing Solutions:

Headsets usually come with brackets that can be attached to the headset which gives extra space between the screen and your eyes so that you can wear your glasses. Because you are further away from the screen and the fact that the brackets allow light to seep in, the immersion factor decreases. Also, the brackets are just uncomfortable which is just annoying and can further decrease your immersion. The glasses also cause a losst in FOV (field of view) and may scratch the lenses.

Prescription inserts are a good solution for this but they need to be bought separately.

My Solution:

I propose using computer vision and deep learning to help people with hyperopia by creating an algorithm that can transform the image displayed in the lenses so that people with hyperopia can see it better.

Understanding The Solution:

There are multiple ways ways to make an image clearer for people with hyperopia. The image processing may take a while depending the computer so a good balance of enhancing the image and computation power needs to be considered. I am going to use super resolution which takes an image and upscales it to make it clearer.

Environment:

I do not possess the necessary hardware (such as NVIDIA GPUs) for deep learning. Thus, I decided to train my models on the cloud using Kaggle, as this platform provides a variety of free accelerators (CPUs, GPUs, and TPUs). Kaggle didn't satisfiy my processing power needs because the downside of using a free cloud service was that I had limited memory to work with. I had to constrict my dataset so that it would not exceed Kaggle's RAM limit.

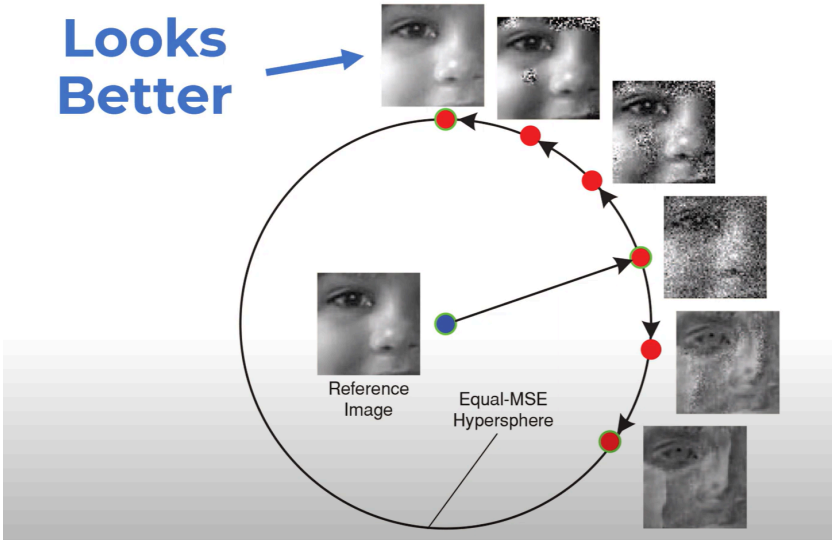
Super Resolution:

There are multiple ways to create super resolution, for simplicity purposes, I am going to only look at single frame super resolution.

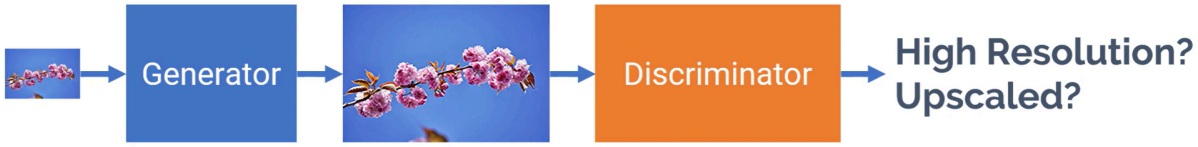
Bilinear Upscaling: Spreads out the pixels and takes the weighted average of neighboring pixels but it still doesn't look very good.

SRCNN[4]**:** Uses convolutional neural networks to train a model by mapping a set of high

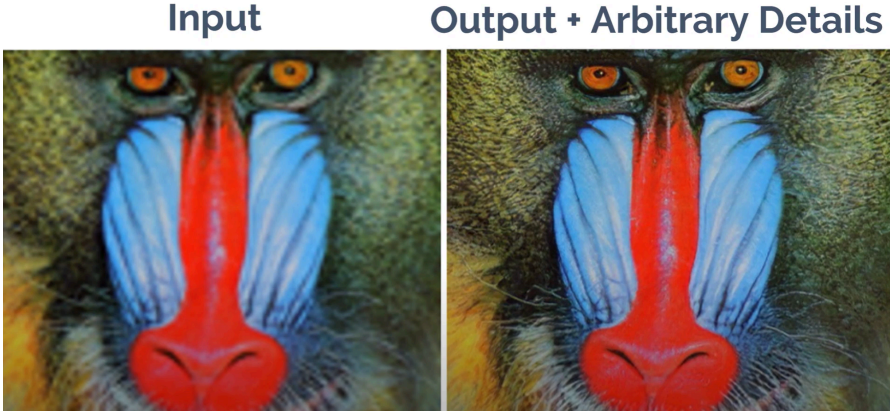
resolution images with low resolution images. Uses the mean squared difference for the loss function which isn't the best because MSE only cares about pixel wise intensity differences and not the structural information about the content of an image.



SRGAN[5]**:** Uses generative adversarial networks to upscale the image. The loss function that is used is the VGG based loss function which uses a VGG-19 model trained on ImageNet and computes the difference of the feature maps, by inputting the high and low resolution images, created after the first few layers. Also uses a discrimator model to try to determine whether the upscaled image was real or fake.



ESRGAN[6]**:** It is a further development on top of the original SRGAN but the loss function is changed to compare the feature maps before activation. The batch normalization layers are removed and more dense layers are added. Uses residual scaling to stablize the network due to the added complexity. The discrimator is relativistic. This model also uses network interpolation to find the middle ground between the GAN model and the MSE model.



I tried to implement the same model architecture from the SRGAN because it works

well and takes less computation power than ESRGAN. Unfortunately the GPU only had 16 GB VRAM which wasn't enough both the generator and discriminator. So I had to use the generator only. On top of that, I also had to use a simpler loss function that doesn't use the VGG.

Dataset:

Due to the topic of super resolution being so popular, there are so many datasets on the internet. I decided to use a dataset[7] from Kaggle that ended up to be around 5 gigabytes of raw image data. The dataset includes the high resolution images with 3 corresponding low resolution images, one has half the resolution, one has a fourth, and one has a sixth. I am only going to use the ones with half the resolution.

All of the images in the dataset have a resolution of 1200 x 800. The reason even the low resolution images have that resolution is because they are all the same size but the each 2 pixels are counted as 1. So in reality, the low resolution images have a resolution of around 600 x 400. Due to the environment, I downscaled the high resolution images to 400 x 400 and further downscaled to 100 x 100 for the low resolution images.

After extracting all the data from the images and saving them to numpy arrays, the final shapes of the arrays were (1254, 400, 400, 3) and (1254, 100, 100, 3). 1254 images, 400 pixels by 400 pixels, 3 for RGB for each pixel, 100 x 100 pixels for the low resolution.

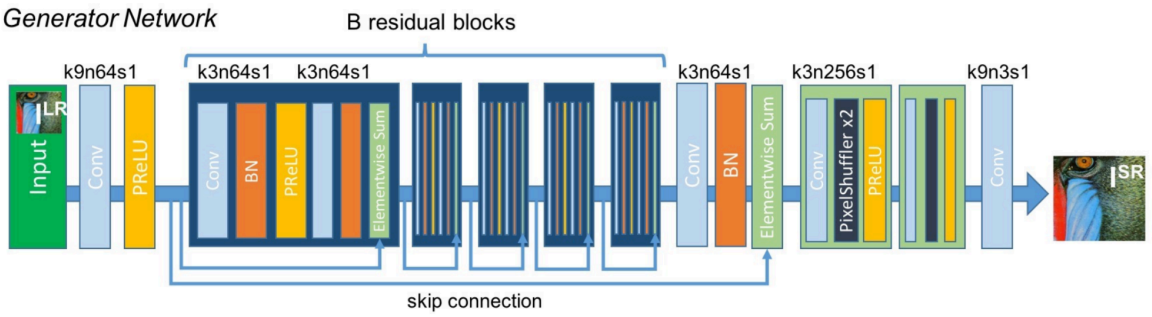
Model:

SRGAN:

This architecture consists of multiple models: a generator, a discriminator and a VGG model.

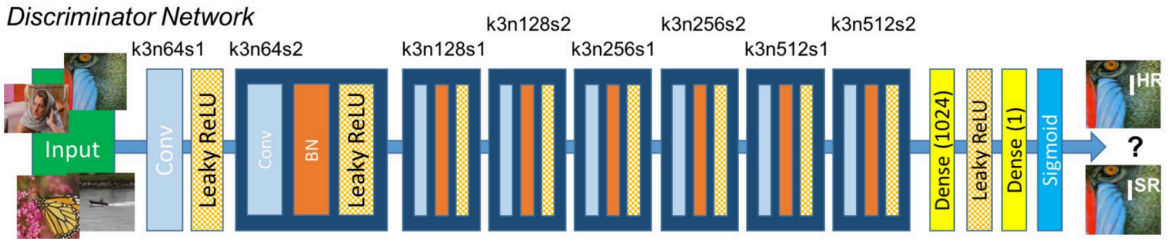
Generator:

The model that actually upscales the image. It consists of many layers, starting with the input layer. It then passes through a convolutional layer with a PReLU activation function. This layer has 9x9 kernel size and 64 channels. After that, there are five residual blocks. Each block consists of a convolutional layer with batch normalization PReLU, another convolutional layer with batch normalization and and elementwise sum. Every layer in the residual blocks have a 3x3 kernel and 64 channels. Then there is an extra convolutional layer with batch normalization and an elementwise sum. After that, there are two upscaling blocks. Each upscaling blocks has a convolutional layer with a pixel shuffler and PReLU activation. The layers in the upscaling blocks have a 3x3 kernel and 256 channels. Then, there is one last convolutional layer with a 9x9 kernel and 3 channels.



Discriminator:

This model is used to classify whether an image is originally high resolution or if it was upscaled by the generator. Both models are trained simultaneously. The model starts of with the input and leads into a convolutional layer with Leaky ReLU activation. This layer has 3x3 kernel and 64 channels. There are then 7 blocks with each containing a convolutional layer with batch normalization and Leaky ReLu. Each layer has a 3x3 kernel and 64 channels. Then, there is a dense layer with 1025 neurons with Leaky ReLU and a dense layer with a Sigmoid activation function.



Loss Functions:

There are two loss functions:

Content Loss: This loss uses the VGG where both the upscaled image and original high resolution image are fed into the pre-trained VGG network. The feature maps obtained from a set convolutional layer are then used for calculating the euclidean distance.

Using the feature maps from the VGG allows the generator to preserve structural features of the images.

Adversarial Loss: This loss is derived from the probabilities of the discriminator model.

The sum of both losses returns the perceptual loss function.

My model:

As I stated before, my environment didn't allow me to implement the complete SRGAN. So I decided to only use the generator model from the SRGAN and the mean squared error loss function. Which is basically the SRResNet.

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 100, 100, 3)	0	-
conv2d_37 (Conv2D)	(None, 100, 100, 64)	15,616	input_layer[0][0]
p_re_lu_19 (PReLU)	(None, 100, 100, 64)	64	conv2d_37[0][0]
conv2d_38 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_19[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_38[0][0]
p_re_lu_20 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…
conv2d_39 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_20[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_39[0][0]
add_17 (Add)	(None, 100, 100, 64)	0	p_re_lu_19[0][0], batch_normalizat…
conv2d_40 (Conv2D)	(None, 100, 100, 64)	36,928	add_17[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_40[0][0]
p_re_lu_21 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…
conv2d_41 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_21[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_41[0][0]
add_18 (Add)	(None, 100, 100, 64)	0	add_17[0][0], batch_normalizat…
conv2d_42 (Conv2D)	(None, 100, 100, 64)	36,928	add_18[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_42[0][0]
p_re_lu_22 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…
conv2d_43 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_22[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_43[0][0]
add_19 (Add)	(None, 100, 100, 64)	0	add_18[0][0], batch_normalizat…
conv2d_44 (Conv2D)	(None, 100, 100, 64)	36,928	add_19[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_44[0][0]
p_re_lu_23 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…
conv2d_45 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_23[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_45[0][0]
add_20 (Add)	(None, 100, 100, 64)	0	add_19[0][0], batch_normalizat…
conv2d_46 (Conv2D)	(None, 100, 100, 64)	36,928	add_20[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_46[0][0]
p_re_lu_24 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…

conv2d_47 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_24[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_47[0][0]
add_21 (Add)	(None, 100, 100, 64)	0	add_20[0][0], batch_normalizat…
conv2d_48 (Conv2D)	(None, 100, 100, 64)	36,928	add_21[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_48[0][0]
p_re_lu_25 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…
conv2d_49 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_25[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_49[0][0]
add_22 (Add)	(None, 100, 100, 64)	0	add_21[0][0], batch_normalizat…
conv2d_50 (Conv2D)	(None, 100, 100, 64)	36,928	add_22[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_50[0][0]
p_re_lu_26 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…
conv2d_51 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_26[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_51[0][0]
add_23 (Add)	(None, 100, 100, 64)	0	add_22[0][0], batch_normalizat…
conv2d_52 (Conv2D)	(None, 100, 100, 64)	36,928	add_23[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_52[0][0]
p_re_lu_27 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…
conv2d_53 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_27[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_53[0][0]
add_24 (Add)	(None, 100, 100, 64)	0	add_23[0][0], batch_normalizat…
conv2d_54 (Conv2D)	(None, 100, 100, 64)	36,928	add_24[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_54[0][0]
p_re_lu_28 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…
conv2d_55 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_28[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_55[0][0]
add_25 (Add)	(None, 100, 100, 64)	0	add_24[0][0], batch_normalizat…
conv2d_56 (Conv2D)	(None, 100, 100, 64)	36,928	add_25[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_56[0][0]
p_re_lu_29 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…
conv2d_57 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_29[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_57[0][0]

add_26 (Add)	(None, 100, 100, 64)	0	add_25[0][0], batch_normalizat…
conv2d_58 (Conv2D)	(None, 100, 100, 64)	36,928	add_26[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_58[0][0]
p_re_lu_30 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…
conv2d_59 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_30[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_59[0][0]
add_27 (Add)	(None, 100, 100, 64)	0	add_26[0][0], batch_normalizat…
conv2d_60 (Conv2D)	(None, 100, 100, 64)	36,928	add_27[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_60[0][0]
p_re_lu_31 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…
conv2d_61 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_31[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_61[0][0]
add_28 (Add)	(None, 100, 100, 64)	0	add_27[0][0], batch_normalizat…
conv2d_62 (Conv2D)	(None, 100, 100, 64)	36,928	add_28[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_62[0][0]
p_re_lu_32 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…
conv2d_63 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_32[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_63[0][0]
add_29 (Add)	(None, 100, 100, 64)	0	add_28[0][0], batch_normalizat…
conv2d_64 (Conv2D)	(None, 100, 100, 64)	36,928	add_29[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_64[0][0]
p_re_lu_33 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…
conv2d_65 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_33[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_65[0][0]
add_30 (Add)	(None, 100, 100, 64)	0	add_29[0][0], batch_normalizat…
conv2d_66 (Conv2D)	(None, 100, 100, 64)	36,928	add_30[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_66[0][0]
p_re_lu_34 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…
conv2d_67 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_34[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_67[0][0]
add_31 (Add)	(None, 100, 100, 64)	0	add_30[0][0], batch_normalizat…

conv2d_68 (Conv2D)	(None, 100, 100, 64)	36,928	add_31[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_68[0][0]
p_re_lu_35 (PReLU)	(None, 100, 100, 64)	64	batch_normalizat…
conv2d_69 (Conv2D)	(None, 100, 100, 64)	36,928	p_re_lu_35[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_69[0][0]
add_32 (Add)	(None, 100, 100, 64)	0	add_31[0][0], batch_normalizat…
conv2d_70 (Conv2D)	(None, 100, 100, 64)	36,928	add_32[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 100, 100, 64)	256	conv2d_70[0][0]
add_33 (Add)	(None, 100, 100, 64)	0	batch_normalizat… p_re_lu_19[0][0]
conv2d_71 (Conv2D)	(None, 100, 100, 256)	147,712	add_33[0][0]
pixel_shuffler_2 (PixelShuffler)	(None, 200, 200, 64)	0	conv2d_71[0][0]
p_re_lu_36 (PReLU)	(None, 200, 200, 64)	64	pixel_shuffler_2…
conv2d_72 (Conv2D)	(None, 200, 200, 256)	147,712	p_re_lu_36[0][0]
pixel_shuffler_3 (PixelShuffler)	(None, 400, 400, 64)	0	conv2d_72[0][0]
p_re_lu_37 (PReLU)	(None, 400, 400, 64)	64	pixel_shuffler_3…
conv2d_73 (Conv2D)	(None, 400, 400, 3)	15,555	p_re_lu_37[0][0]

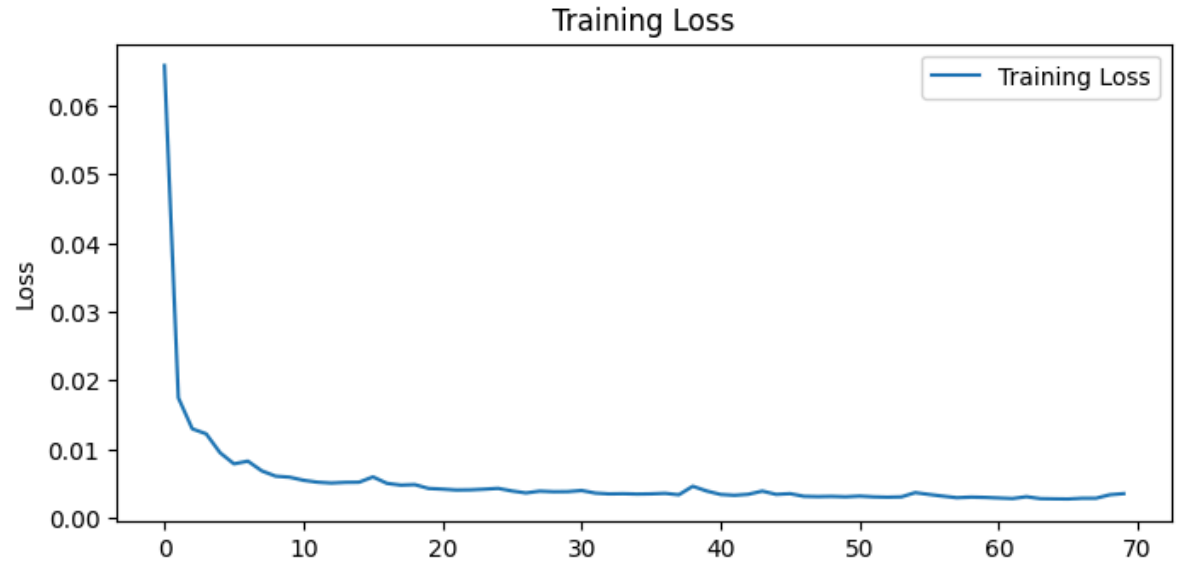
Total params: 1,554,883 (5.93 MB)

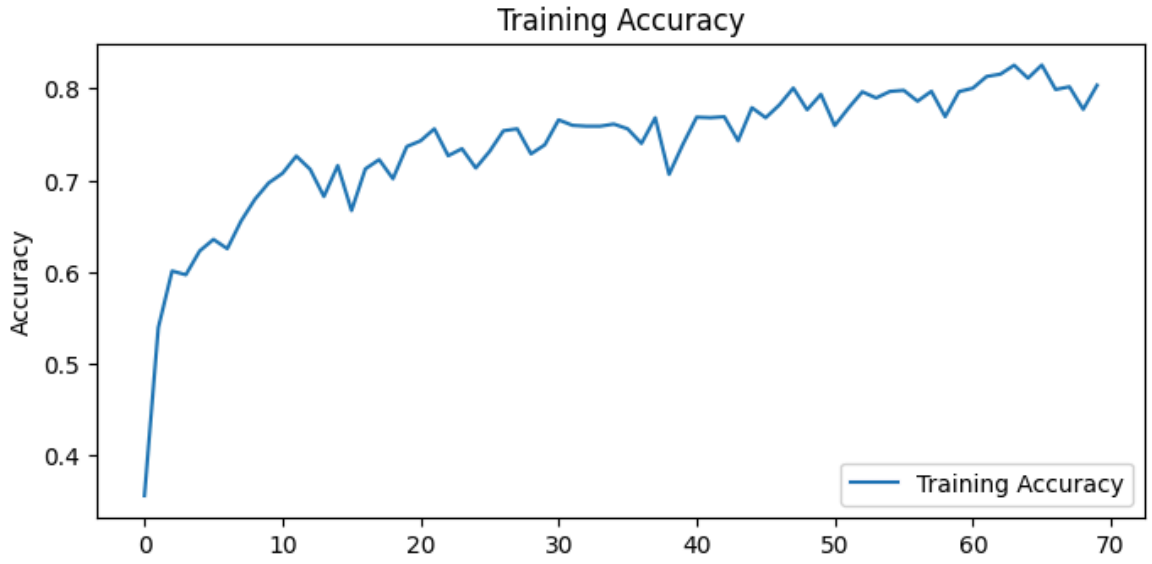
Trainable params: 1,550,659 (5.92 MB)

Non-trainable params: 4,224 (16.50 KB)

Results:

I ran my model for 70 epochs which was excessive but I wanted to make up for the lack of data and loss functions. My final training accuracy was 79.9% and the final testing accuracy was 81.1%. Considering the environment and what it allowed me to do and how small the dataset was, the accuracy scores are pretty high.





Challenges and Future Improvements:

Data: There are a lot of great available datasets online but due to the environment and the limited RAM, I couldn't use too many images. A future improvement would be to gather more training data and have the resources to be able to use them.

Model: Because I had limited processing power, it was unrealistic to have the multiple models training at the same time so I couldn't use the full SRGAN architecture. Future improvements would include having better hardware to train models so that I will no longer be limited by GPU VRAM and processing speeds so that I'll be able to use the complete SRGAN and possibly even ESRGAN.

Further Application:

This algorithm has numerous potential applications. First, it can help people suffering from hyperopia use VR headsets. Furthermore, more features such as color adjustment and contrast increases may aid hyperopic people more. Second, it can be applied to video surveillance systems where captured footage often has inadequate resolution.

References:

1. https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment#:~:text=Prevalence,has%20yet%20to%20be%20addressed
2. https://www.grandviewresearch.com/industry-analysis/virtual-reality-vr-market
3. https://my.clevelandclinic.org/health/diseases/hyperopia-farsightedness
4. https://arxiv.org/abs/1501.00092
5. https://arxiv.org/abs/1609.04802
6. https://arxiv.org/abs/1809.00219
7. https://www.kaggle.com/datasets/quadeer15sh/image-super-resolution-from-unsplash