Upscaling Of Image Quality By Increasing Resolution Using Super Resolution Generative Adversarial Networks

Ryan Ueda Teo Shao Ming
Singapore Polytechnic
School of Computing
Singapore
ryanueda.21@ichat.sp.edu.sg

***Abstract*— The first Generative Adversarial Network, or GAN or short, was first proposed in 2014. Over the last decade, the state of deep learning, especially in the field of artificial image generation, has been rapidly advancing. In this paper we will explore and preprocess the data, experiment with model architecture and go over how GANs work along with their applications.**

Keywords—explore; preprocess; deep learning; GAN; generative; artificial; images; model architecture;

I. INTRODUCTION

After the proposal of the first GAN by Ian Goodfellow and his colleagues in 2014, GANs have been constantly improving year over year. For example, in October 2021, NVIDIA came out with a new ground-breaking model, the StyleGAN3. This new model is able to resolve previous complications of the StyleGAN2, and is able to learn to mimic camera motion, video and animation generation.

The progress of GAN capabilities have improved by leaps and bounds as compared to when they first came out in 2014, only able to handle Low-Resolution images. Aside from artificial image generation, GANs are also capable of photo translation, photo editing, image denoising and much more. State of the art applications include using GANs for medical product development and medical imaging quality enhancement, as well as OpenAI's world-famous image generation program, DALL-E.

In this paper, we aim to delve deeper into upscaling Low-Resolution images up to four times the original resolution, by implementing a Super Resolution Generative Adversarial Network, otherwise known as SRGAN.

II. GANS, SRGANS & HOW THEY WORK

GANs consists of two networks, a Generator and a Discriminator Network.

As the name implies, the generator tries to fool the discriminator by generating artificial images indistinguishable from real images, while the discriminator tries to resist by accurately identifying real and fake images apart. These two contrasting networks work in tandem to learn and train complex data such as images, audio, and videos.

In mathematical terms, the model is effectively a Minimax game, as the generator tries to minimize the loss function, while the discriminator is trying to maximize the loss function. This allows for the two networks to be trained by alternating between gradient ascent and descent. In theory, training is complete when the artificial images generated are indistinguishable from real images.

Fig. 1. GAN Model Architecture Diagram

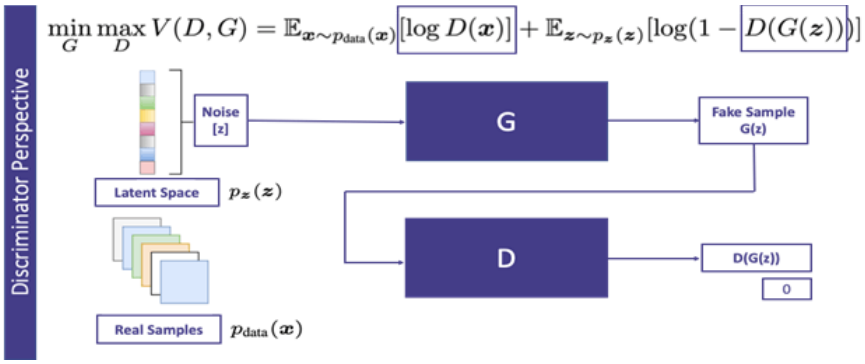
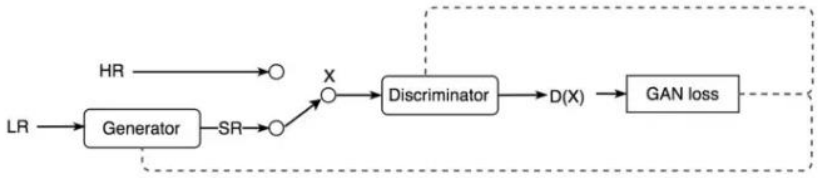


Fig. 2. Objective Function

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

Fig. 3. SRGAN Model Architecture Diagram



Similar to Generative Adversarial Networks (GANs), SRGANs applies a deep network together with an adversary network to produce higher resolution images. During training, High-Resolution images are downsampled to Low-Resolution images. A generator then upsamples the Low-Resolution image to a super resolution image. The discriminator helps distinguish between High-Resolution images and the generated images, and backpropagates the GAN loss in order to train the discriminator and generator.

Fig. 3. Adversarial Loss

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$$

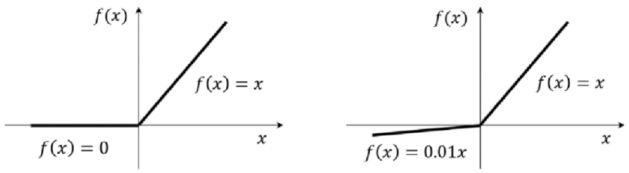
Fig. 4. Perceptual Loss

$$l_{VGG/i.j}^{SR} = \frac{1}{W_{i,j} H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2$$

$\phi_{i,j}$ **The feature map for the j-th convolution (after activation) before the i-th maxpooling layer.**

In an SRGAN, the loss function of the generator comprises of the following losses: reconstruction loss and adversarial loss. The reconstruction loss can be calculated pixel-wise using the mean squared error between high-resolution and super-resolution images. In the context of an SRGAN, perceptual loss is calculated by measuring the MSE of features extracted by a VGG-19 network. Finally, to train the discriminator, the usual GAN discriminator loss is used as the loss function.

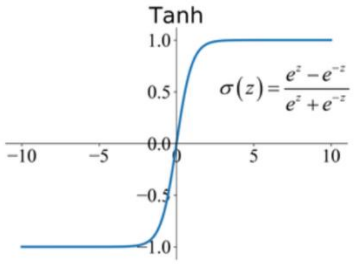
Fig. 5. ReLU & LeakyReLU Activation Functions



ReLU activation function *LeakyReLU activation function*

Generally, it is commonly accepted in the deep learning community that the Rectified Linear Unit (ReLU) activation function is extremely capable, and often the best solution for many deep learning problems. However, in GANs, the Leaky Rectified Linear Unit (LeakyReLU) activation function is preferred as the standard. This is because it allows for gradients to flow more easily through the network. When using a ReLU activation function, negative values are truncated to 0, causing a blockage of gradients flowing through the network. As a result, gradients will not propagate through those neurons and weights will not be updated. By allowing a small negative value to pass through instead of 0, LeakyReLUs attempt to solve this issue, as it is detrimental for proper gradient flow in the network due to it being the generators only way to learn from the discriminator.

Fig. 6. Tanh Activation Function

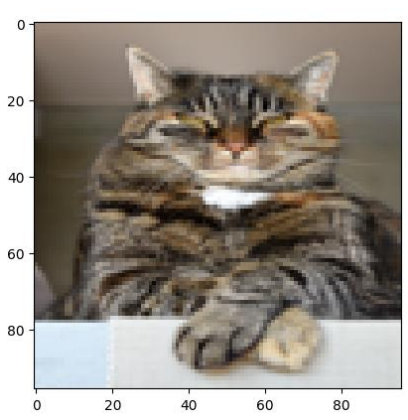


In GANs, a tanh activation function is typically used. This is due to the fact that images are typically normalized to be in the range [-1, 1].

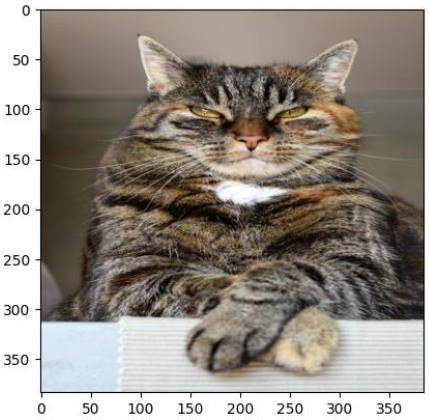
III. DATASET

The dataset consists of two types of images, Low-Resolution and High-Resolution coloured images. The Low-Resolution images consists of 100 images in the shape (96, 96), while the High-Resolution images consists of 100 images in the shape (384, 384)

Fig. 7. Low-Resolution & High-Resolution Dataset Samples



Low-Resolution



High-Resolution

Low-Resolution

High-Resolution

Some samples of the Low-Resolution and their High-Resolution counterparts shows the vast contrast in image quality between the two variants.

IV. METHODOLOGY

Before creating the model architecture, we first define a custom Class called the SubPixelConv2D layer. This layer performs upsampling on input tensors using the depth_to_space function. Methods to build and configure the layer are included within this class as well. This will be used later on in the Generator network.

Now that we have defined our customs class and functions, we can create our Generator model.

The model starts with an input layer, which takes in the input shape and 64 neurons. Next, there are 3 blocks of residual layers, in which a BatchNormalization layer is sandwiched between two convolutional layers. The output of these layers are then added to the input, creating a shortcut connection. After the residuals blocks, there is a final convolutional layer with BatchNormalization, and the addition of the input of the first convolutional layer. The model is followed with two more convolutional layers with 256 neurons, and SubPixelConv2D layers with an upsampling factor of 2, as well as a ReLU activation function.

All convolutional layers have been fitted with a padding of ‘same’ and a HeNormal kernel initializer. Lastly, the final layer is a convolutional layer with 3 neurons and a tanh activation function, representing our output.

Fig 8. Generator Model Architecture

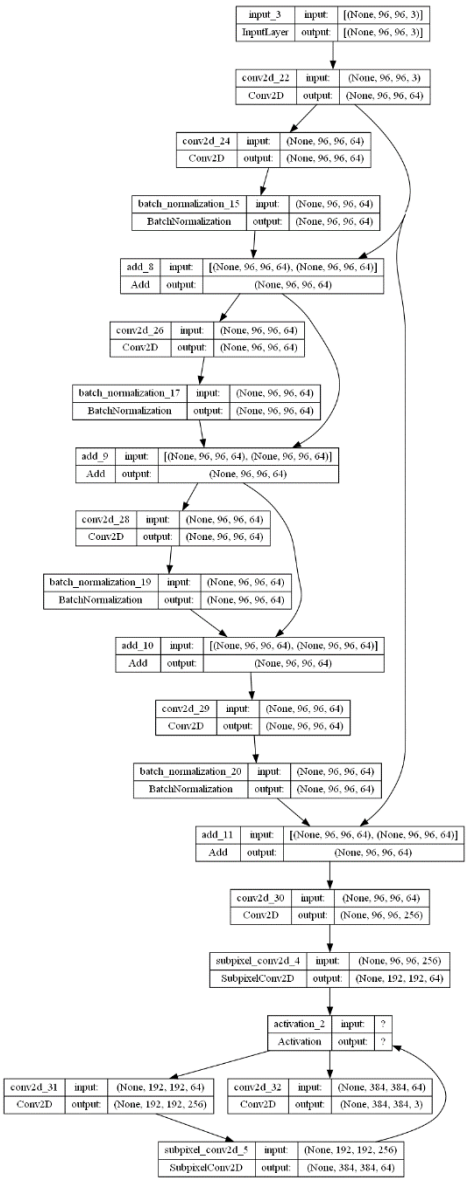
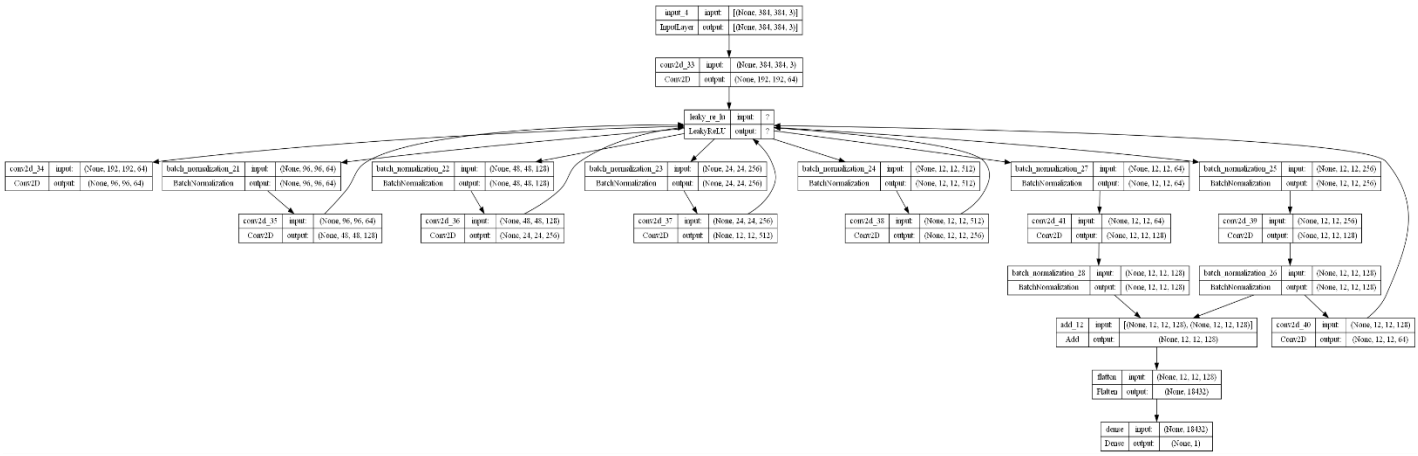


Fig 9. Discriminator Model Architecture



Upon creating the Generator model, we can proceed to define the Discriminator model.

The model starts by taking an input image and passes it through a convolutional layer with 64 neurons and applies a LeakyReLU function. This layer is the first feature extractor of the model. Next, four convolutional layers, each increasing in number of neurons are implemented to try to obtain a higher level of features. A stride of 2 is used to reduce the spatial dimensions of the input by half.

These layers are also fitted with BatchNormalizations and LeakyReLU activations. Another four more convolutional layers, with varying neurons and kernel sizes are added, with a stride of 1.

Finally, the model takes the output of the last convolutional layer and adds it to the output of the previous convolutional layer. This output is then passed through a Flatten layer to convert the multi-dimensional vector into a 1D vector, which is passed through the final Dense layer with a single neuron, representing our output.

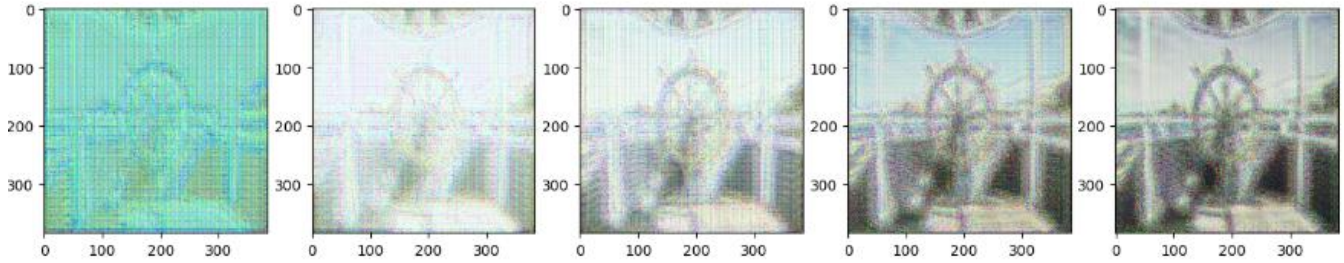
The Dense layer uses a sigmoid activation function because our output is binary in nature, representing real and fake images.

Fig 10. VGG19 Model Architecture

Before fitting the model, we have to define a third model, the VGG19 model. This model is used to extract features from the High-Resolution images, which are then used as a loss function to train the generator. By training the generator to produce images that are both High-Resolution and possesses similar feature maps to the High-Resolution images, we are able to not only create high quality images, but also ensure that they are semantically consistent.

The VGG19 model is a pre-trained model imported from the keras library. We define a input shape of 384 by 384, set the include_top parameter to ensure the fully connected layers of the VGG are not included, and initialize the weights pretrained on the ImageNet dataset.

Fig 11. Initialize Generator Learning



With our 3 network model architectures created, we can move on to initialize Generator learning. The initialization of generator learning refers to the process of training the generator before it s used in the full architecture of the GAN. Typically, it is done with a simple loss function such as Mean Squared Error (MSE). The MSE loss function compared the generated high-resolution images with ground truth images and is then used to update the generator's weights. The main objective of this

process is to provide a good starting point for the generator in order to produce realistic resolution images before using the full architecture, where the adversarial loss function is used to fine tune the performance of the generator.

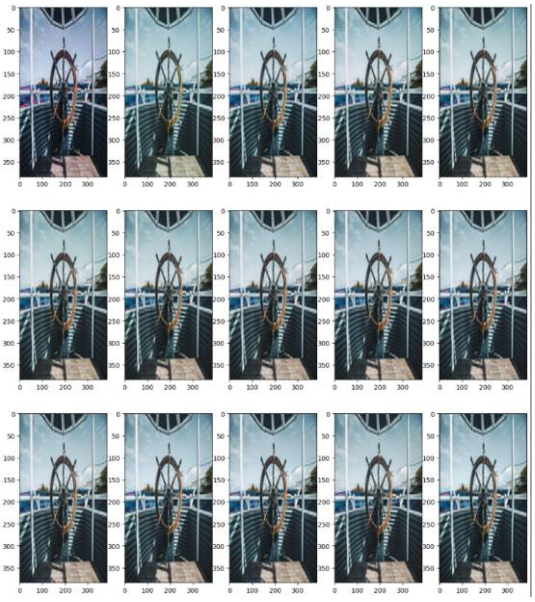
Fig 12. Training The SRGAN

Now that our Generator, Discriminator and VGG19 networks have been constructed and we have initialized the generator's learning, we can proceed to fit the data into the full SRGAN model architeecture.

To train the network, we use tensorflow's GradientTape over a for loop. There are multiple reasons for doing so: Firstly, it uses a computation graph to record operations, which allows for efficient computation of gradients in comparison to for loops. Its automatic differentiation also allows for the computation of gradients for any operation, regardless of complexity. It general, GradientTape offers a more efficient and versatile method of gradient computation in complex machine learning and deep learning models.

Here, we train the model for 50000 epochs. In every epoch, we calculate the Discriminator losses for real and fake images, Generator loss, VGG19 loss, GAN loss as well as the Mean Squared Error loss function. Every 20 epochs, the Generator predicts on a Low-Resolution image from the training set. And saves the image in a directory. While such an operation may be more computationally expensive, it is extremely useful in helping us monitor the performance of the model and its outputs.

Fig 13. SRGAN Training In Progress

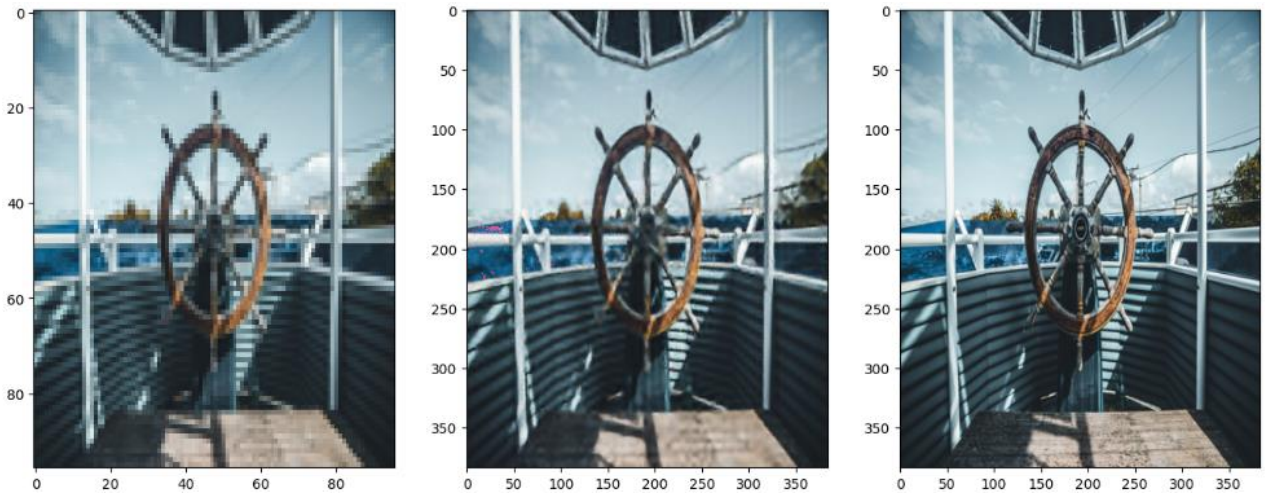


The image above, shows the images generated over the span of the 50000 epochs while training. As shown, it is clear that the images generated vastly increase in image quality as it trains. This is a good sign that the model is performing as expected.

V. Model Evaluation

Now that the model has completed training, we will evaluate how the model has performed after running for 50000 epochs.

Fig 14. Low-Resolution, High-Resolution & Generated Images

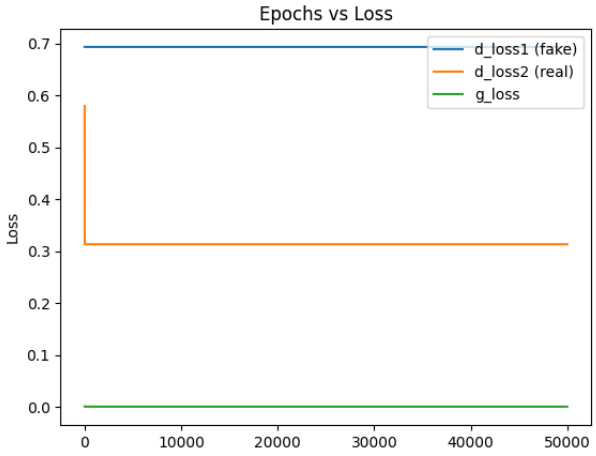


Low-Resolution *SRGAN Generated* *High-Resolution*

As shown from the Low-Resolution, High-Resolution and SRGAN generated images, we can see that the model has performed up to our expectations. Although the image generated by the SRGAN is slightly less high quality than the High-Resolution training sample, we can see that its pretty close. In comparison with the Low-Resolution training sample that the model predicted on, we can see that the image quality is very signifantly better.

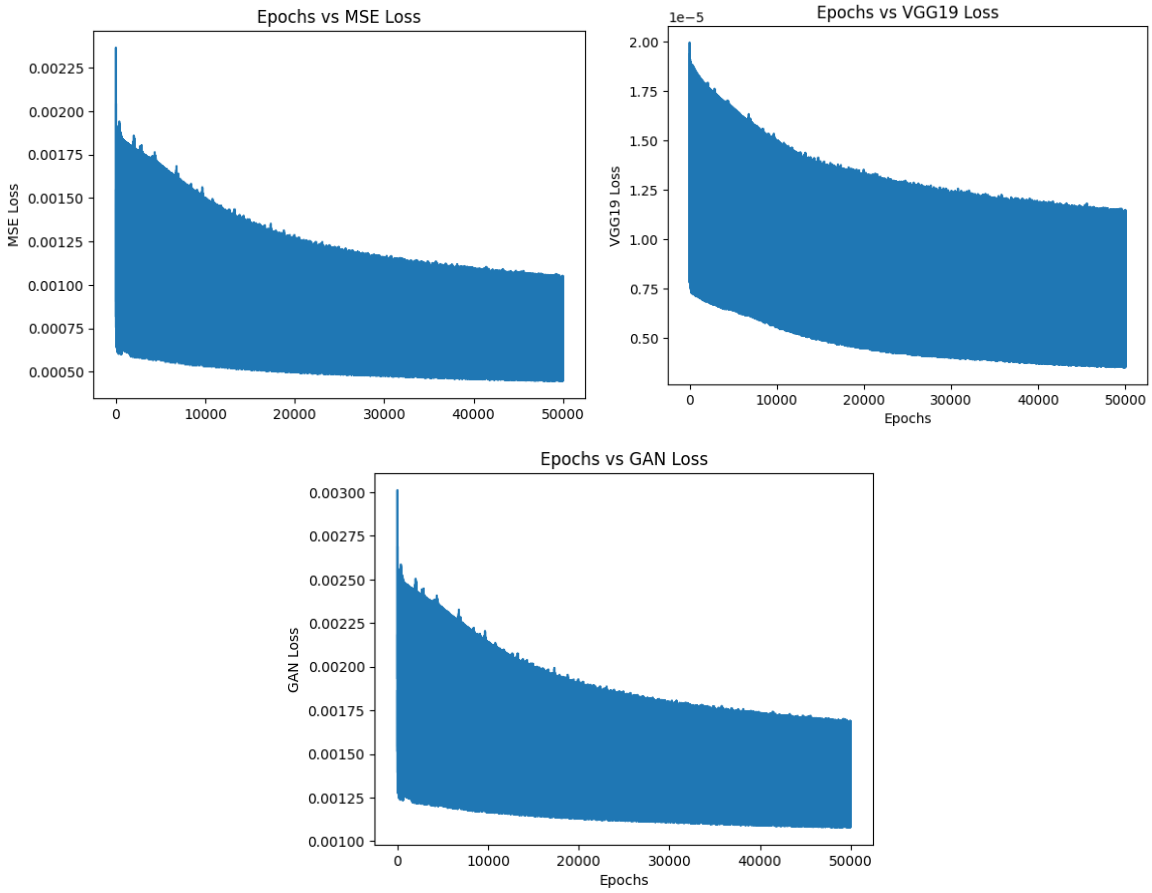
To observe how our model has done statistically and mathematically, we can visualize and analyze the metrics we have collected earlier during training by plotting some graphs.

Fig 15. Discriminator & Generator Loss



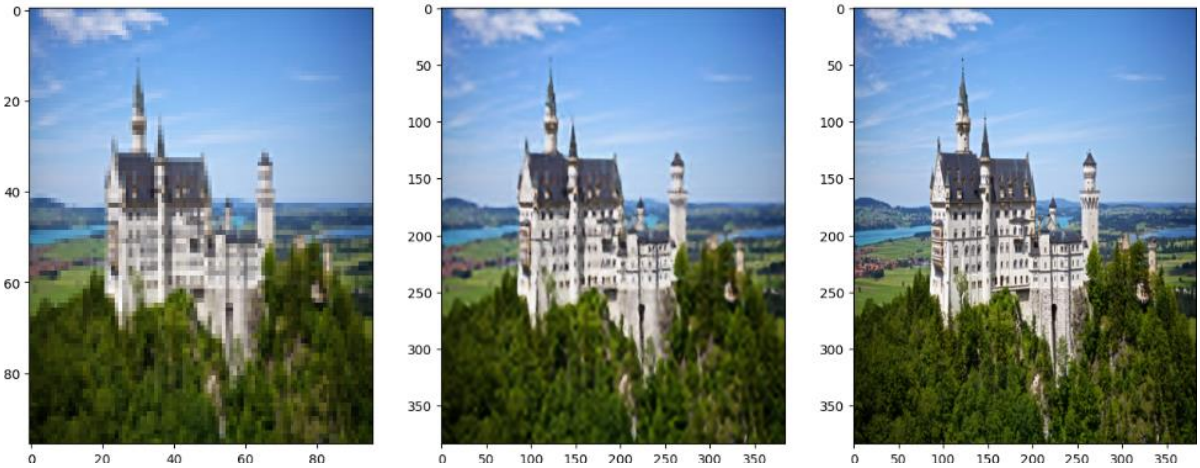
The above graph shows the plot of real and fake image discriminator loss, and generator loss. From the grah we can see that the three losses vary greatly. All three losses are effectively stagnant throughout training, apart from a sharp decrease in real image loss at the start of training. The fake loss is nearly twice as high as the real loss, meaning that the model is able to easily classify real images, but has a harder time classifying fake images. This means that the generator is able to produce images of quality high enough that the discriminator has a tougher time classifying between generated images and the grouth truths. In addition, our generator boasts an extremely low loss, which shows our model is performing well.

Fig 16. MSE, VGG19 & GAN Loss

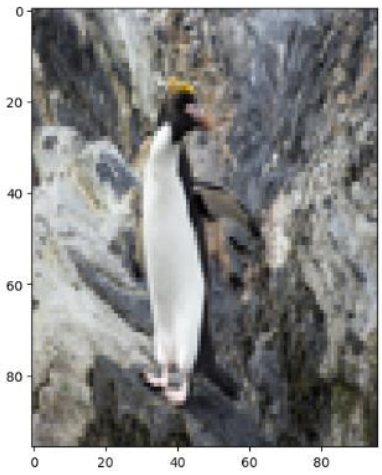


By plotting the MSE loss, VGG19 model loss, and GAN loss, we can observe that although seems very volatile, has a very small margin of error as shown in the scale of the y-axis. The range of error seems to be within 0.002, with the lowest error being nearly 0.001 and the highest being around 0.003. It is important to note that while there are many metrics to help with the evaluation of the model, statistical results vary greatly from what may be considered good images, as it is very subjective to human standards, and hence should be taken with a grain of salt, only to be used as mathematical guidelines.

Fig 17. Low-Resolution, High-Resolution & Generated Images 2

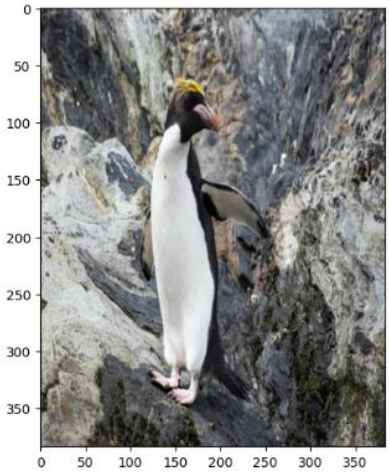


Low-Resolution *SRGAN Generated* *High-Resolution*



Low-Resolution

SRGAN Generated



High-Resolution

Based on the samples of images generated by the SRGAN, as compared to the Low-Resolution and High-Resolution training samples, it seems that the model has performed up to standard, by upscaling the images drastically from its poor-quality counterparts. This aligns with our metrics analysis earlier, showing that the model has been performing at a high-level of quality.

VI. CONCLUSION

In this paper, we have discussed how to analyze and forecast the electricity generation capacity of Singapore along with various deep learning techniques. These include how to perform exploratory data analysis on time series data, preprocess and prepare data for time series analysis using LSTMs, checking for stationarity, how to create LSTM model architectures, as well as how to perform hyperparameter tuning on our model.

In this paper, we have discussed how Generative Adversarial Networks (GAN) and Super Resolution Generative Adversarial Networks (SRGAN) work, their model architectures, real-world implementations and applications, and how to make use of them along with various deep learning techniques to upscale image quality and resolution. Furthermore, we delved deeper into how to evaluate GAN models and their performance.

VII. REFERENCES

https://medium.com/sciforce/whats-next-for-gans-latest-techniques-and-applications-3be06a7e5ab9
https://medium.com/analytics-vidhya/gans-a-brief-introduction-to-generative-adversarial-networks-f06216c7200e

https://towardsdatascience.com/generative-adversarial-networks-explained-34472718707a

https://arxiv.org/abs/1406.2661

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

https://sthalles.github.io/intro-to-gans/

https://blog.paperspace.com/super-resolution-generative-adversarial-networks/

https://jonathan-hui.medium.com/gan-super-resolution-gan-srgan-b471da7270ec#:~:text=Super%2Dresolution%20GAN%20applies%20a,design%20without%20GAN%20(SRResNet).