COSC 4P80 Project

*Implementing and Analyzing a Deep Learning Network to Enhance the Resolution of an Image

Ameen Khawaja
Computer Science Department
Brock University
St. Catharines, Canada

***Abstract*—In this report, I take a deep dive into CNNs and how they can be used to upscale an image. I implemented and compared three different CNN architectures, which ranged from a basic SRCNN-inspired model to a more complex, deeper model with more layers. Every model was trained on the DIV2K dataset, which contains 800 high-quality images and another 800 corresponding low-quality images. The CNN was able to generate a super-resolution image, which is an upscaled image with significantly better quality. To effectively measure how much of an improvement the super-resolution image is, I evaluated it using Peak Signal-to-Noise Ratio (PSNR). While all three models generated a super-resolution image that had an improved image quality, the deeper network model showed that more layers does not necessarily mean a better result. Of the three models I tested, the second model performed the best, which struck a balance between the other two models. Furthermore, the model had several limitations due to a multitude of reasons such as overfitting, lack of compute power, and a lack of a larger dataset.**

I. Introduction

In today's world, the ability to capture high-quality photos is essential to many people and is most commonly done through smartphones or physical cameras. In many cases, when an image is captured in the moment, either the image taken looks high-quality or the image looks low-quality. If the image is low-quality, the result is a pixelated image that lacks detail, and there is a lot of noise within the image. This occurs for several reasons, such as poor lighting conditions, incorrect camera settings, or limitations within the camera hardware itself. The frustration of capturing a passing moment only to be left with a poor quality image is a common experience – especially when the opportunity to retake the shot is lost.

Fortunately, there is a solution to recovering a pixelated image with lots of noise: super-resolution. The key idea behind super-resolution is to enhance the resolution of low-resolution image to a high-resolution image. There is a wide variety of applications that super-resolution is used in, such as surveillance and television. Consider the example of a surveillance camera capturing a person's face from a far distance but not having the ability to identify the person due to low-resolution and lack of clarity. In this scenario, creating a super-resolution image of that instance can help transform that low quality footage to a higher-quality footage, increasing the chance of identifying the person. Super-resolution images are also used in televisions, making a lower resolution video appear sharper and more clear on a higher resolution television. As an example, watching a 240p quality video on an 8K tv will look distorted, but with super-resolution, it will upscale the lower-resolution image to fill the higher-resolution screen by reconstructing missing details and refining textures.

There are several different algorithms that can be used to construct a super-resolution image – a common and promising approach (discussed further in section II) to achieving an effective super-resolution image is through the use of convolutional neural networks (CNN). CNNs are a type of deep learning model that are able to effectively capture image features by leveraging several convolutional layers. In each layer, a CNN is able to identify complex features that are within the image, beginning with simple features such as edges or the texture of a photo, and progressing to more complex features such as shapes and patterns.

In order to evaluate the quality of an image, a peak-signal-to-noise ratio (PSNR) is used because it provides a numerical value of the ratio between the original and compressed image. PSNR compares the original image with the reconstructed image, and the result is a single value in decibels (dB). The higher the dB value, the more accurate the image reconstruction is and thus, a better super-resolution result. This will be discussed further in-depth section II.

Training the CNN model on a dataset of high and low quality images is computationally intensive and dedicated graphics cards are needed to effectively train the model to its maximum potential. When I initially began training my model, processing the images was not feasible on my computer as it took too much time. As a result, I created a work around of compressing the images and training the model on those compressed images. The specific details of this will be discussed in section II.

In this paper, I aim to show that my deep learning CNN model is able to effectively upscale images. The architecture of of the CNN is designed around extracting features of an image, followed by processing those features, then upscaling the features, and lastly, reconstructing the image, which transforms the processed data into a higher-resolution RGB image.

II. Background

Before diving right into the deep learning model and how it performed for constructing a super-resolution image, this section will aim to provide a detailed background explanation of choices that led to my final model design.

A. Understanding the Dataset

The dataset that the model was trained on is the DIV2K dataset. The DIV2K dataset consists of diverse RGB images, meaning that it's a dataset thats more than just photos, it contains various images of different scenes, textures, lighting conditions, and objects. This is crucial because it challenges the CNN model to generalize well. Within this dataset, there is a total of 800 unique high-resolution images, and another 800 corresponding images that are a downscaled version at different scaling factors. For this project, I used the high-quality image dataset paired with it's corresponding low-quality dataset that had a downscale factor of x2.

TABLE I: High-Resolution Images and Corresponding Downscaled (x2) Images

Image	Resolution (High-Res)	Resolution (Low-Res x2)
image_0001.png	2040x1404	1020x702
image_0002.png	2040x1848	1020x924
image_0003.png	2040x1356	1020x678
image_0004.png	2040x1344	1020x672

In table I, it can be seen that every high-resolution image has a corresponding low-resolution image that is exactly half of the high-resolution image. I decided to take advantage of this, as it allowed us to process the entire dataset at a lower resolution. In my testing, I processed every high-quality image in the dataset with a dimension of 128x128, and every low-resolution image had a corresponding dimension of 64x64. This retains the same structure of the dataset, except, it also allows us to compute these values within a realistic time frame because prior to reducing the image dimensions, computing the images at full quality was not sufficient on my computer.

B. Super-Resolution Techniques

There are several different super-resolution methods and techniques I found for constructing super-resolution images while researching and reading academic papers – most notably, the use of generative models (SRGAN) [1], SRCNN [2], and Very Deep Super Resolution (VDSR) [3]. SRGAN uses a generative adversarial network where two neural networks apply deep learning on an image and compete against one another to see which network can have the more accurate prediction. It focuses on perceptual quality, so it generates images that look very realistic, however, the downside is the images contain artifacts. SRCNN was one of the first CNN models used in super-resolution. It takes the low-resolution image as an input and outputs the high-resolution image by mapping the low resolution image to the high-resolution image. Lastly, there is VDSR, which uses many layers with small filters and a very high learning rate. I based my implementation on the SRCNN algorithm while also taking inspiration from the VDSR implementation – in particular, the small filter sizes and high learning rate. So, my model is a CNN that learns to map low-resolution images to high-resolution images based on the DIV2K dataset. In section IV the architecture of my model will be discussed in-depth, alongside how it shares similarities from both SRCNN and VDSR.

C. CNNs for Image Processing

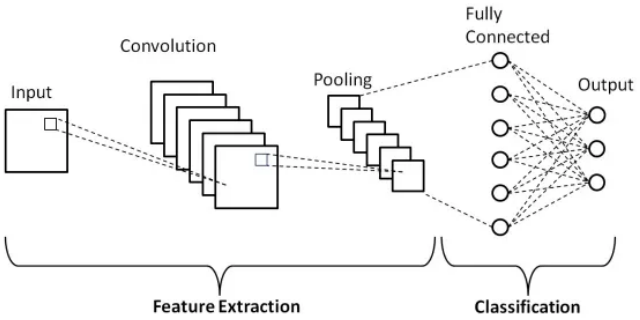

Fig. 1: CNN Architecture [4]

CNNs perform very well in image classification tasks, facial recognition, and object detection. This is mainly due to their unique architecture (as seen in Fig. 1) which consists of convolutional layers, Rectified Linear Units (ReLu) layers, pooling layers, and a fully connected layer which is the final layer. Each layer within the CNN can occur many times.

The convolutional layer takes an input and applies a filter, which then outputs a feature map. The feature map detects features and patterns from the input image. Specifically, features include any of the following seen in an image: edges, textures, shapes, or any other characteristic in the image.

The ReLu function introduces non-linearity (meaning it can learn more complex data) while the model is learning, as well, using a ReLu function helps reduce the vanishing gradient problem from happening during training. Next comes the the pooling layer, where it has a primary purpose of reducing the dimensions of the feature maps while retaining the most important information.

A common pooling technique is max pooling, which essentially slides a filter over the input feature map by selecting the highest value within that filter region. Eventually, the feature map size decreases and as it gets applied over several layers within the network, the dimensions of the image continue to shrink into a compact result. So, if I had an image of 128x128, if I apply pooling, it could shrink to 64x64, then 32x32, and so on, while every iteration retains the most crucial information. Lastly, the fully connected layer takes the output of the most recent pooling layer as an input and generates a classification based off that.

To construct a super-resolution image, I used several convolutional layers to extract image features from the low-resolution image, which is the input of the network. The feature maps that are produced are similar to the feature maps produced in an image classification task. In Fig. 1 there is an output layer for an image classification task. However, in the super-resolution image model, I are not outputting a class, but rather, I output a reconstructed super-resolution image.

D. Image Evaluation Metrics

To assess the quality of the super-resolution image that the CNN model produced, I was initially looking at the image the CNN would output and visually, I would be able to spot the improved image quality of the super-resolution image. However, I wanted a way to metrically prove that my model is able to output an improved image. So, I read a research paper on an SRCNN implementation [4]. In the report, the author's used PSNR as a metric which I found fascinating, so I decided to also implement it. I found an explanation of the PSNR equation [5] that made it simple to understand. I did not need to implement this math in my code as TensorFlow had a built in method to do it, but nonetheless, understanding the math behind it made the coding part more simple. PSNR is a common metric that compares a reconstructed image to the original high resolution image. The first step in calculating PSNR is to determine the Mean Squared Error (MSE):

$$\text{MSE} = \frac{\sum_{M,N} \left[I_1(m,n) - I_2(m,n) \right]^2}{M \cdot N} \tag{1}$$

where $\mathbf{M}$ is the number of rows, $\mathbf{N}$ is the number of columns, m is the row number of a pixel within the image, and n is the column number of that pixel within the image. Putting the equation altogether, $I_1(m,n)$ gives the color value of a single pixel in the original image, and $I_2(m,n)$ gives you the pixel value of the reconstructed image. Together, I take the difference and square it, then divide by the number of rows multiplied by the number of columns, giving the MSE. The MSE is computed as the average squared difference between every pixel within the original image and the corresponding pixel within the super-resolved image.

Once the MSE value is obtained, I can calculate the PSNR value as such:

$$PSNR = 10 \log_{10} \left(\frac{R^2}{MSE} \right) \tag{2}$$

where R represents the maximum possible pixel value that an image can have (in this case, 255). Ultimately, I take the maximum pixel value and divide it by the MSE, and multiply it by $\mathbf{10 \log_{10}}$, which compresses the ratio and converts the result to a decibel value. The higher the decibel value the PSNR outputs, the better the image quality is (this will be demonstrated in section V).

III. Preprocessing the Data

Converting the images from the DIV2K dataset into meaningful results was trickier then I anticipated. To load the images, I sorted the images by name so every high-resolution image corresponds to it's low-resolution image. Then, for each pair of images, I process the image pair by resizing the high-resolution image to 128x128 and the corresponding low-resolution pair to 64x64. Once the images were resized, then I convert the images into arrays of pixel values and normalize it. Lastly, I convert the image arrays to NumPy arrays which prepares the CNN to begin training on. So, the algorithm for loading the dataset into values can be described as such:

Algorithm 1: Load and Process Image Dataset

Input: high_res_images, low_res_images
Output: high_res_array, low_res_array
Initialize empty arrays high_res_images and low_res_images
Sort image names from both directories to ensure correct image pairing
foreach *(high_res_file, low_res_file) in image pairs* **do**
    Process high resolution image and resize to 128x128
    Process low resolution image and resize to 64x64
    Append processed images to array's
Convert image arrays to NumPy arrays
return *high_res_array, low_res_array*

During the process of resizing images to 128x128 and 64x64, a major problem I had was the images were very blurry after the downsampling was complete. However, I discovered that there was a resampling algorithm called LANCZOS that is built into Python's Pillow library which I were already using for handling the image data. Upon applying the LANCZOS resampling algorithm, the images instantly became much more clear which helped with training the model. Without LANCZOS, the training data would have been too blurry, making it difficult for the model to learn the proper mapping between low and high-resolution features. Additionally, I normalized the image data in a range of [0,1] by dividing the pixel values by 255 as it helped the network train faster and reach an optimal solution.

IV. Implementation

I implemented the CNN model using the TensorFlow deep learning framework and Python. TensorFlow simplified the process significantly, as it has an environment with built in methods such as optimizers, pre-built layers, along with useful utility methods that assisted with processing images and training the model. A few of the TensorFlow methods I used were:

- **Conv2D**: Layers extract features from the input images through applying filters that detect any features or patterns
- **ReLU / PReLU**: An activation function that introduces non-linearity, allowing the model to learn complex patterns and reducing the vanishing gradient issue.
- **BatchNormalization**: Adds stability while the model is training by normalizing layer input values, which ultimately means the model generalizes better and reaches a faster convergence.
- **UpSampling2D**: Increases the image resolution, so in this case, when the model needs to create the super-resolution image, it transforms the image from a 64x64 dimension to 128x128.

There are three different models I built using TensorFlow to gather data and to observe the performance the neural network.

The result of all three models will be discussed in section V and the architecture of each model will be discussed here.

Starting with the first model I built, it consists of a total of 15 layers and is broken down in Table II:

Layer Type	Number of Filters	Kernel Size
Conv2D	32	3×3
PReLU	-	-
BatchNormalization	-	-
Conv2D	16	1×1
PReLU	-	-
Conv2D	32	3×3
BatchNormalization	-	-
Conv2D	32	3×3
Conv2D	24	3×3
PReLU	-	-
UpSampling2D	-	2×2
PReLU	-	-
Conv2D	16	1×1
PReLU	-	-
Conv2D (output)	3	3×3

TABLE II: Layer Architecture of the First Model

As seen in Table II, the first model's network architecture contains 15 total layers – including 7 convolutional layers that use a combination of 3x3 and 1x1 kernels, with filter sizes being 16, 24, or 32. The reason I put a number of different filters is to allow for an effective feature extraction, while BatchNormalization layers and PReLU activations are there to stabilize the network. The architecture processes input images from the dataset of size 64x64 and produces an output that is a super-resolution image of size 128x128, through an UpSampling2D layer that doubles the dimensions.

The second model I created has a different architecture than the first model, and is broken down in Table III:

Layer Type	Number of Filters	Kernel Size
Conv2D	64	3×3
BatchNormalization	-	-
Conv2D	64	3×3
BatchNormalization	-	-
Conv2D	128	3×3
BatchNormalization	-	-
Conv2D	128	3×3
BatchNormalization	-	-
UpSampling2D	-	2×2
Conv2D	64	3×3
Conv2D (output)	3	3×3

TABLE III: Layer Architecture of the Second Model

The second model has a total of 11 layers, with 6 of those being convolutional layers with significantly larger filter sizes (64 and 128) in comparison to the first model. Another key difference of the second model is that it uses the standard ReLU function rather than PReLU, and it has double the BatchNormalization layers at a 4 of four versus 2. While this model has fewer total layers than the first model, it processes much more features per layer due to the larger filter sizes, which in theory should allow for more complex feature extractaion at each step, ultimately, generating a better image. The training time of this model is around three times longer than the first model in terms of testing.

The third and last model I created was inspired by VDSR [3], where I created a very deep model with a total of 26 layers. The breakdown is in table IV

Layer Type	Number of Filters	Kernel Size
Conv2D	64	3×3
BatchNormalization	-	-
Conv2D	64	3×3
BatchNormalization	-	-
Conv2D	96	3×3
BatchNormalization	-	-
Conv2D	96	3×3
BatchNormalization	-	-
Conv2D	128	3×3
BatchNormalization	-	-
Conv2D	128	3×3
BatchNormalization	-	-
Conv2D	256	3×3
BatchNormalization	-	-
Conv2D	256	3×3
BatchNormalization	-	-
Conv2D	128	3×3
BatchNormalization	-	-
Conv2D	128	3×3
BatchNormalization	-	-
UpSampling2D	-	2×2
Conv2D	96	3×3
BatchNormalization	-	-
Conv2D	64	3×3
BatchNormalization	-	-
Conv2D (output)	3	3×3

TABLE IV: Layer Architecture of the Third Model

As seen in Table IV, the third model is very deep with a total of 26 layers, including 13 convolutional layers and 12 BatchNormalization layers. The filter's also progress in an increasing manner and then gradually decrease for compression – starting at 64, 128, 256 and then decreasing to 96, 64, 3. In comparison to the first model which used PReLU and only 15 layers, and the second model which had 11 layers, this architecture is significantly more complex but comes at the cost of taking ten times longer to train (30 epochs takes around 50 minutes).

With the layer-by-layer architecture of each of the three CNN models explained, in the next section, I will discuss the training process and the results obtained from each of these three models.

V. Training & Results

In this section, all three models discussed in section IV will be tested against images that belong in the DIV2K dataset and images that do not belong in the DIV2K dataset. This will allow us to see whether or not the models are successfully able to enhance a lower resolution image. Unless explicitly mentioned, the parameters for the deep learning network for all experiments use a learning rate of 0.0001, a batch size of 16, and a validation split of 0.1. Also, the same images will be used across all three models for experiments to compare the differences. Afterwards, model 1 will showcase its results across several different images in the dataset and not in the dataset.

A. Model 1

The training and evaluation of model 1 has shown very promising results in terms of enhancing the image and creating a super-resolution image.

Fig. 2: Model 1 - Low Res Photo

In figure 2, I can see a low resolution photo that is very pixelated and noisy.

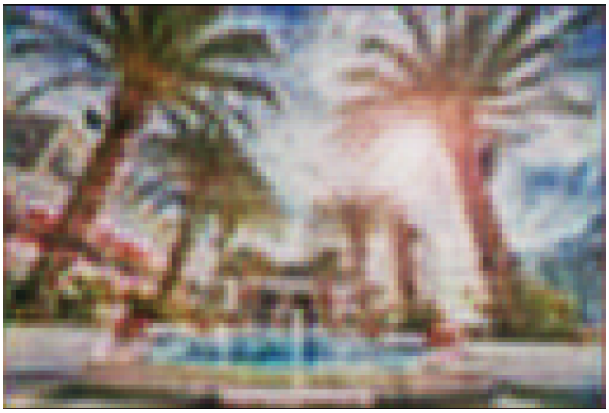

Fig. 3: Model 1 - Super Res Photo

In figure 3, I see that the quality has significantly improved over the low resolution photo in figure 2. This is the super-resolution image that the deep learning model predicted after being trained on the dataset. If I closely analyze it, there is much less noise and it's not as pixelated as the low-resolution image. Specifically, the super-resolution image successfully enhanced the detail and clarity of the pool and the palm trees. The network's prediction reconstructed a fairly accurate image that seems to maintain all the correct RGB pixels in place.

Fig. 4: Model 1 - High Res Photo

In figure 4, it can be seen that the high resolution image looks the best with very little noise and no pixelated appearance. The purpose of comparing the low-resolution and high-resolution image is to see how well the network was able to train.

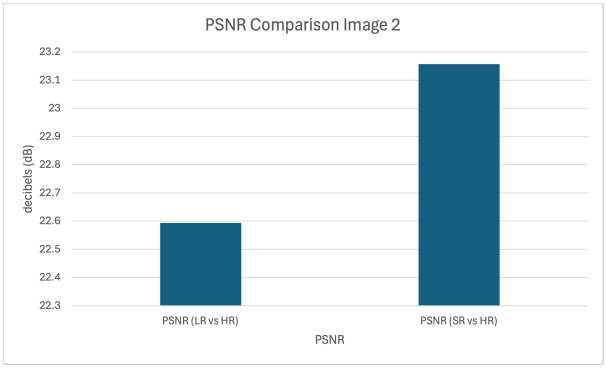

Fig. 5: Model 1 - PSNR

Figure 5 confirms the visual improvements from the low-resolution image to the super-resolution image by analyzing the PSNR values. The PSNR value (higher PSNR is better) of the low-resolution image vs the high-resolution image resulted in approximately 22.6 dB, whereas the PSNR value of the super-resolution image vs the high-resolution image resulted in a value of 23.15 db. The improvement of nearly 0.6 dB suggests that the model has learned to construct images that match the high-resolution photo closer than the low-resolution photo.

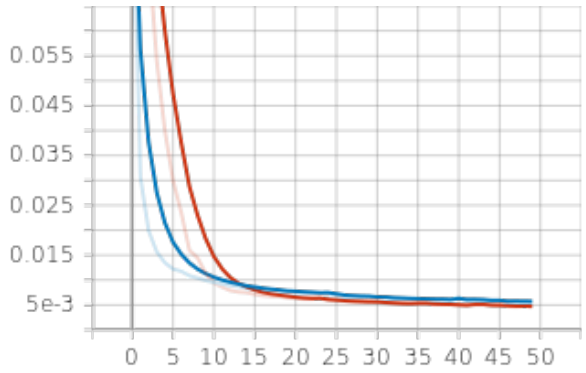

Fig. 6: Model 1 - Epoch Loss

Figure 6 showcases the epoch loss during learning. The epoch loss curve shows a fairly quick initial convergence at around 13 epochs for both the training (blue line) and validation (red line) sets. Past 13 epochs, the loss stabilizes at around 0.01, which means that the model has reached it's optimal state. By looking at both the validation and training loss curves, I believe that because they are so close to one another, that indicates the model generalized well and is not overfitting too much.

Now, lets analyze and see how model 1 performs on two low-resolution images that do not belong in the DIV2K dataset.

Fig. 7: Model 1 - Low Res Image not in DIV2K

Figure 7 was a low resolution image found on Google Images. Once the model completed its training, I passed this image into the model to generate a super-resolution image.

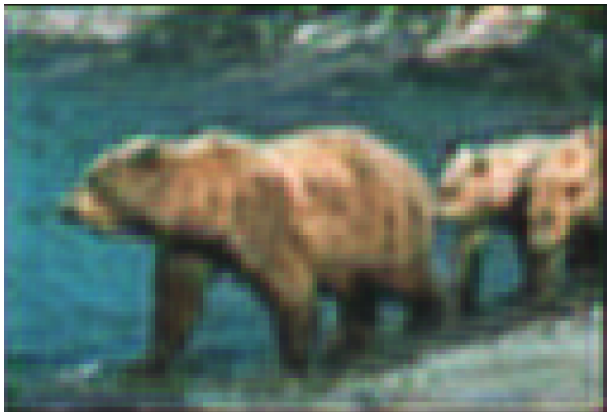

Fig. 8: Model 1 - Super Res Image

Figure 8 is a super-resolution image generated from figure 7. Immediately, several improvements can be noted – specifically, less noise in the image, less pixels (look at the two bear cubs in figure 7 and compare it to the two bear cubs in figure 8), and the image feels smoothened out overall. However, the image produced is far from perfect and obvious flaws can also be seen. For example, there is a green tint covering the entire image which is altering the RGB colours.

Fig. 9: Model 1 - Low Res Image not in DIV2K

Figure 9 was another low resolution image found on Google Images that appears very pixelated. Once the model completed its training, I passed this image into the model to generate a super-resolution image.

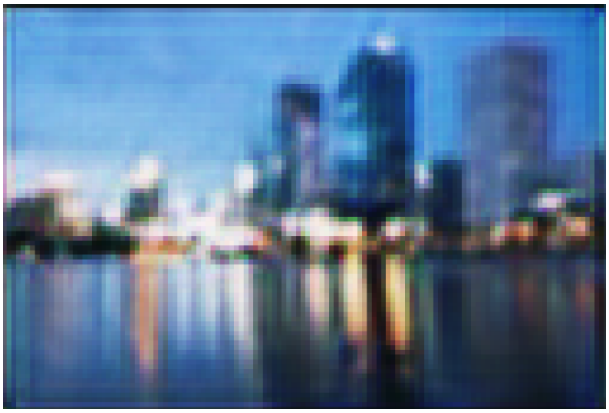

Fig. 10: Model 1 - Super Res Image

Figure 10 is a super-resolution image generated from figure 9. The result is a smoothened out and less pixelated image, however, there is still lots of blurriness to the image. Aside from the blurriness, the green tint appears on the super-resolution image once again, similar to figure 8 While the result is not phenomenal, a minor improvement can still be seen.

B. Model 2

Model 1 showed it had promising results when it came to generating a super-resolution image. Now, let's test model 2, where the architecture is more complex and as there are more features being processed in every layer. I will be testing it against the exact same images to compare the performance between the two.

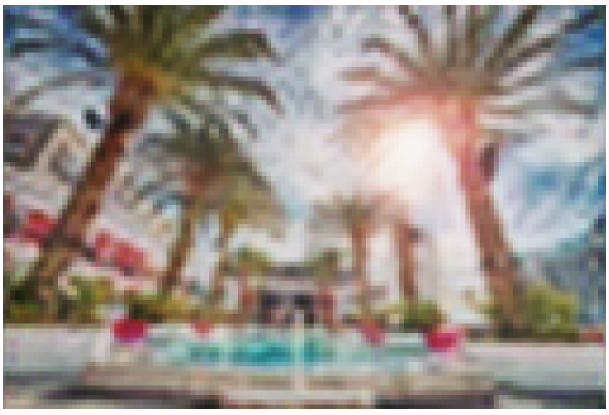

Fig. 11: Model 2 - Super Res Photo

In figure 11, a super resolution-image is generated using a more complex architecture as discussed in section IV. By examining this image, it appears to recover more fine details compared to model 1's results. To be specific, the enhancement is most notable in the palm tree leaves, the architectural details of the building, the clarity of the water, and the overall sharpness is improved with brighter colours. In figure 3, the image appears significantly more dull than 11.

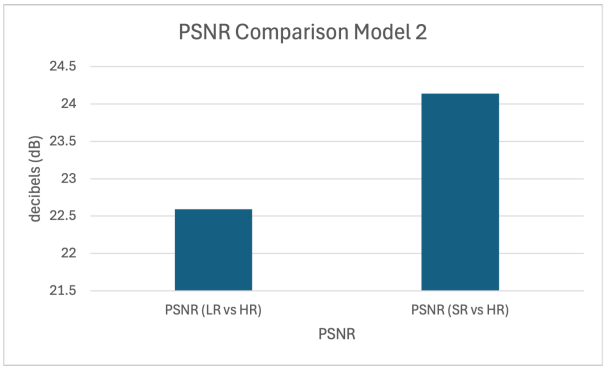

Fig. 12: Model 2 - PSNR

The PSNR scores achieved by the second model is much better compared to model 1. Although model 1 showed a PSNR improvement of nearly 0.6 dB (from 22.6 to 23.15 dB), model 2 is able to have an improvement of 1.4 dB (from 22.6 to 24 dB). The increase in PSNR indicates that the second model was able to reconstruct the image more closer to the high-resolution image and it definitely generated an super-resolution image that is better than the low-resolution variant.

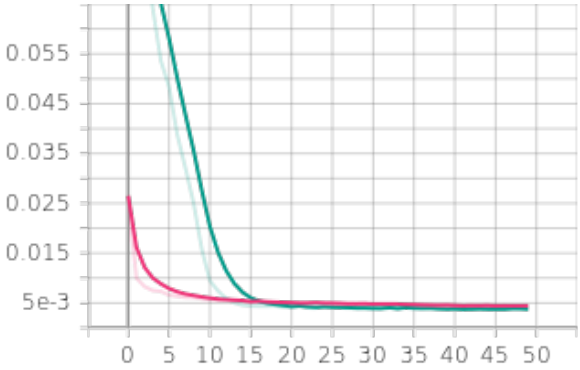

Fig. 13: Model 2 - Epoch Loss

The epoch loss curves in figure 13 are rather interesting, as there is noticeable differences compared to model 1. The second model shows a faster convergence in the training set (pink line), whereas the validation set (green line) takes around 15 epochs to start stabilizing itself. Additionally, the final loss values in model 1 (figure 6) were around 0.001, whereas with model 2, it converges at around 0.005. The gap between training and validation loss is also smaller, meaning this model generalized better.

Fig. 14: Model 2 - Super Res Image

Figure 14 is a super-resolution image that was generated by using model 2's architecture. The difference is subtle, yet noticeable if you look closely. Figure 14 is less pixelated and you can see the face of the bear cubs a bit more clear in comparison to model 1's result. Also, there appears to be less of a green tint in figure 14 compared to figure 8.

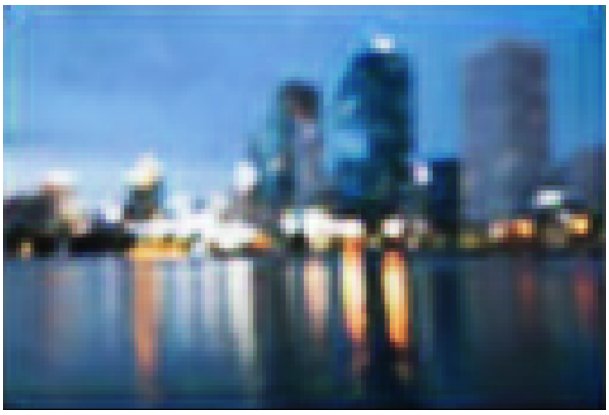

Fig. 15: Model 2 - Super Res Image

By using model 2, it was able to generate figure 15, which is significantly better in terms of quality than model 1 (figure 10). The reflection of colours on the lake are smoothened out and appear less pixelated, and the overall image has less of a green tint.

The second model was able to show that it can produce a better visual result. I believe the reasoning is that because the layers had larger number of filters within it, the model was able to find a better balance between feature extraction and reconstructing the image.

C. Model 3

Model 3 is significantly deeper than model 1 and 2 and takes an average of 1 hour and 30 minutes to train 50 epochs. The performance of it reveals that more layers does not necessarily mean better results, as will be shown below.

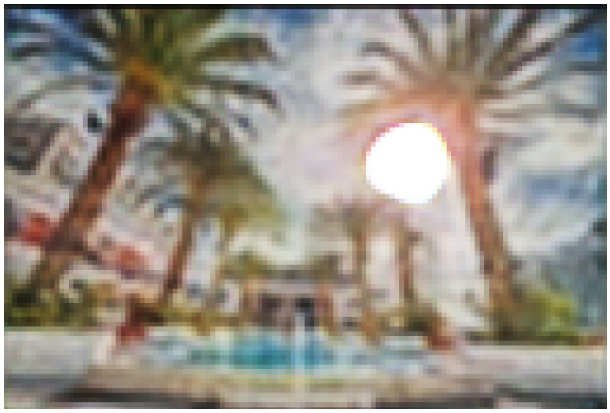

Fig. 16: Model 3 - Super Res Photo

In figure 16, there is a noticeable error that completely invalidates the image. On the top right of the image near the palm tree leaves, an artifact appears as a bright white circular region. This means that the model incorrectly reconstructed the super-resolution image and the training/learning process was problematic. Since the top right of the image has an artifact and the rest of the image looks relatively improved compared to the original low-resolution image, it indicates that instability occurred in the network.

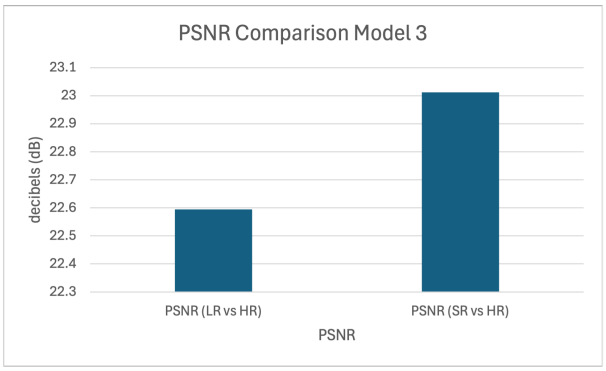

Fig. 17: Model 3 - PSNR

The PSNR improvement as seen in figure 17 for the third model is an improvement of 0.5 dB (from 22.5 to 23.0 dB), which is worse than both model 1 (0.6 dB improvement) and model 2 (1.4 dB improvement). This makes sense because when analyzing figure 16, I can see there is improvement in the image except for when it comes to the palm tree leaves area. Hence, the small dB improvement that is less than model 1 and model 2 makes sense.

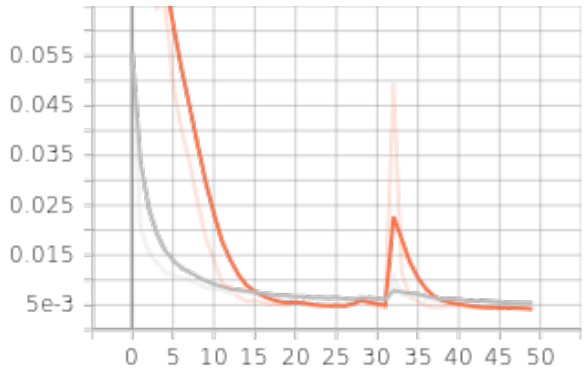

Fig. 18: Model 3 - Epoch Loss

The epoch loss for both the training (gray line) and validation (orange line) is different than both model 1 and model 2. The training loss converges quicker than the validation loss and the validation loss has a lot of instability. If I analyze the validation loss at around 30 to 36 epochs, there is a spike where the error increases, yet, the training has a very small spike that is not significant. The large gap indicates that overfitting happens with this model's architecture and the model is unable to generalize well. I believe that the spike in the graph correlates directly to the artifact in figure 16.

Fig. 19: Model 3 - Super Res Image

By analyzing figure 19, I were unable to spot the difference in image quality compared to figure 14. To us, it appears the model has begun overfitting with too many layers, and as a result, an improvement does not exist.

Fig. 20: Model 3 - Super Res Image

In figure 20, the differences compared to figure 15 are not visually noticeable. The two images appear identical visually, and the model 3 failed to produce a noticeable improvement – unlike the significant difference shown between Model 1 and Model 2.

VI. Conclusions

In this paper, the goal was to demonstrate how effective CNNs are with respect to enhancing the resolution of images. I explored this through the implementation of three different CNN architectures for the purpose of creating a super-resolution image. Initially, I began with using an SRCNN architecture as a baseline (model 1), then I moved to a more complex model (model 2) which used significantly more filters, and to end it off, I used a very deep network architecture (model 3) which took inspiration from VDSR. After closely analyzing and comparing the results of all three models, it is evident that the most complex model with the most layers was not particularly the best, and that there is a middle ground between model complexity and results. The most ideal results were yielded by the second model, which struck a balance between model 1 and model 3 in terms of its architecture.

I began by processing the DIV2K dataset, which contains 800 high-resolution images and another 800 corresponding low-resolution images. The DIV2K dataset was important as all three of the models were trained on it. I constructed all three of the models using TensorFlow and the models themselves were inspired by the SRCNN paper that I read. Once I created the first model that was similar to an SRCNN model, I then adjusted and created two more models that had different architectures. All models included Conv2D, PReLU/ReLU activation functions, BatchNormalization, and UpSampling2D – however, the difference in architecture came down to how they were ordered and the difference in terms of the amount of layers that was used. I analyzed the difference performance for each module and evaluated it in two different ways, the first way was visually evaluating it and observing the differences in terms of the low-resolution image and the super-resolution image. The second way of evaluation was using PSNR, which calculates the ratio between a reconstructed image and the

original image. The higher the PSNR value, the less noise and error there is in the reconstructed image.

While I believe my implementation was successful, I also noticed that there are several limitations for this model. The initial plan was to process every single image at the full 2K resolution and to allow the model to output a super-resolution image that also has near a quality of 2K. However, computing power restricted me from doing so, and as a result, I had to compress the spatial dimensions of the images to 64x64 and 128x128. The model's performance is largely limited by the quality of the training dataset. I believe that if I had an even larger dataset that is even more diverse, I are confident that the network can generate better super-resolution images so long as I also have more compute power. Another limitation I found with the model was that no matter how many iterations it was trained for, a green tint would appear within the images. Nonetheless, implementing this network was a great learning experience for us as a group and after completing the implementation, I are curious how the results would have looked if I used another model such as SRGAN.

REFERENCES

[1] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi, "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network," https://arxiv.org/abs/1609.04802

[2] Chao Dong, Chen Change Loy, Kaiming He, Xiaoou Tang, "Image Super-Resolution Using Deep Convolutional Networks" https://arxiv.org/abs/1501.00092

[3] Jiwon Kim, Jung Kwon Lee, Kyoung Mu Lee, "Accurate Image Super-Resolution Using Very Deep Convolutional Networks" https://arxiv.org/abs/1511.04587

[4] Ganesh Bajaj, "My Goto Notes on Convolutional Neural Networks for any Deep Learning Interview" https://ai.plainenglish.io/my-goto-notes-on-convolutional-neural-networks-for-any-interview-38b7da0735c8

[5] MathWorks, "Peak Signal-to-Noise Ratio (PSNR)," https://www.mathworks.com/help/vision/ref/psnr.html

[6] Stack Overflow, "How to read HDF5 files in Python," https://stackoverflow.com/questions/28170623/how-to-read-hdf5-files-in-python

[7] Stack Overflow, "What is the use of expand_dims in image processing," https://stackoverflow.com/questions/66426381/what-is-the-use-of-expand-dims-in-image-processing

[8] TensorFlow, "Get started with TensorBoard," https://www.tensorflow.org/tensorboard/get_started

[9] TensorFlow, "tf.keras.layers.PReLU," https://www.tensorflow.org/api_docs/python/tf/keras/layers/PReLU

[10] StackOverflow, "How do I resize an image using PIL and maintain its aspect ratio," https://stackoverflow.com/questions/273946/how-do-i-resize-an-image-using-pil-and-maintain-its-aspect-ratio

[11] note.nkmk.me, "Python, Pillow: Add padding (margin, border) to image with expand canvas," https://note.nkmk.me/en/python-pillow-add-margin-expand-canvas/