APS360: Final Report

Image Inpainting

Dec 9, 2020

Guoli Yang
Wayne Chen
Priyanshi Patel
Barney Wei

Table of Contents

1.0 Introduction

Image inpainting is the process of reconstructing an incomplete image so that the result cannot be distinguished from a naturally complete image by a casual observer. This traditionally carries an important application for the restoration of priceless artworks in museums. However, with the digitalization of photos, a more popular use lies within photo editing that gifts the ability to remove any object from the photo and letting image inpainting to fill in the missing region. This means that anyone can easily remove photobombers, watermarks, timestamps, and even entire landmarks such as the Eiffel Tower (Figure 1) from their photos.

Figure 1: Eiffel tower removal using image inpainting with the original image (left), selection of pixels to be removed (center), and the visually coherent generated output image with the Eiffel tower removed (right). [1]

The goal of our project is to build a neural network model that when given images with their centers removed, will be able to regenerate the original image or at least the generated missing region will be visually coherent with the rest of the image (Figure 2). Machine learning (ML) is a reasonable approach because we will train on a large dataset of diverse images so the model could learn to extract and generate image features without being limited to only the surrounding pixels which traditional computer vision techniques suffer from which causes artifacts. Moreover, a successful ML model may be much less computationally expensive during deployment compared to classical techniques.

Figure 2: reconstruction of portraits using image inpainting with the original images (column a), center region removed (column b), and the visually coherent generated output images that are different from the original images (column c). [2]

2.0 Illustration/Figure

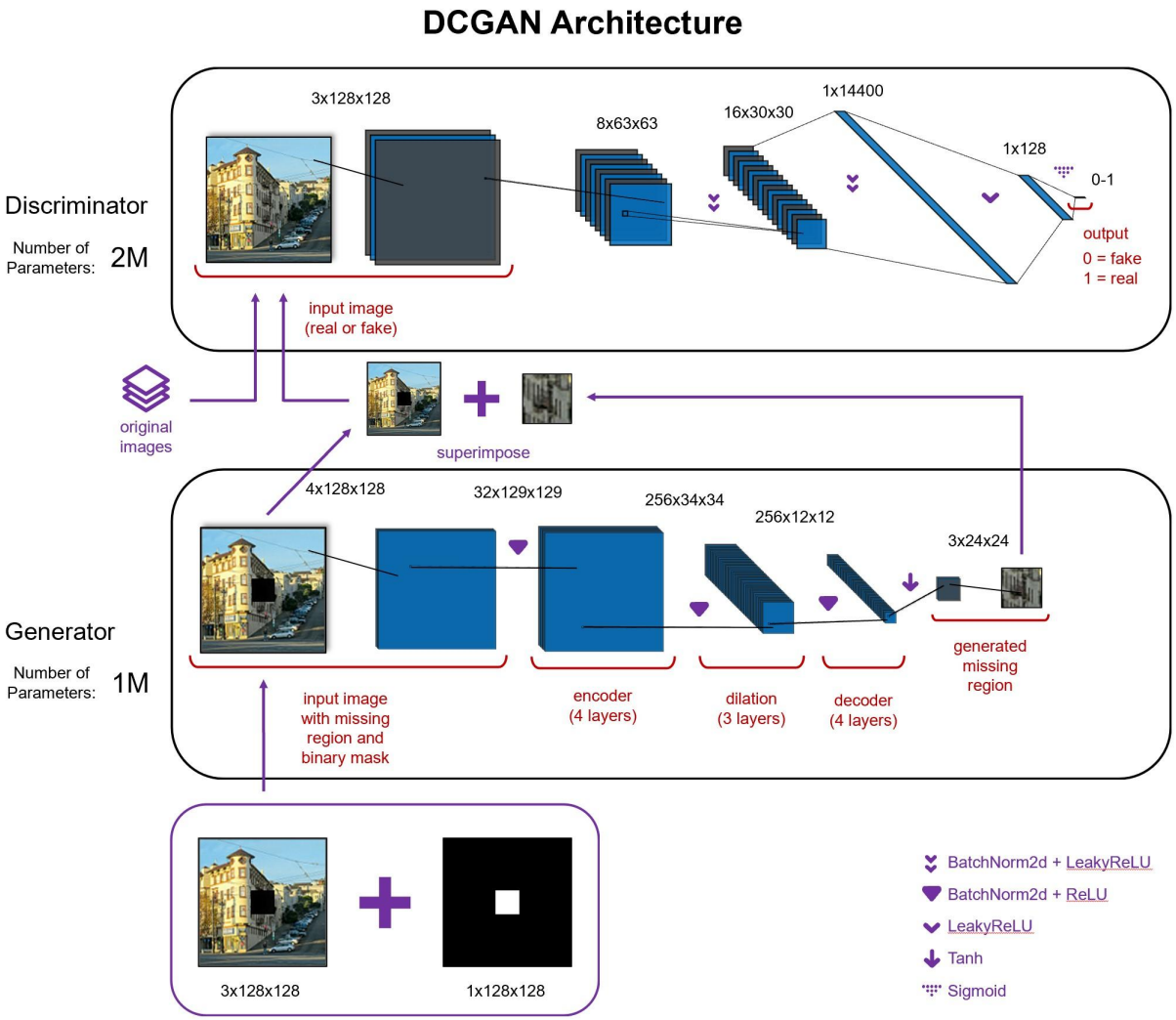

Figure 3: proposed DCGAN architecture

3.0 Background & Related Work

In recent years, deep learning and Generative adversarial network (GAN)-based approaches are widely used in image inpainting because of their powerful learning capabilities and the ability to extract high-level semantic features.

Pathak et al. used a deep Convolutional Neural Network (CNN) model with contest encoders to predict missing parts (center region) of a scene from their surroundings [1]. The context image is passed through the encoder to obtain features connected to the decoder using a channel-wise fully-connected layer. The encoder takes an input image with missing regions and produces a latent feature representation of the image. The decoder then takes the feature representation to produce the missing image content. The model used PASCAL VOC 2007 dataset and achieved a classification accuracy of 56.5%, detection accuracy of 44.5%, and segmentation accuracy of 29.7%. It outperformed a randomly initialized network and a simple autoencoder which is trained to reconstruct its full input. Figure 4 shows an overview of the architecture.

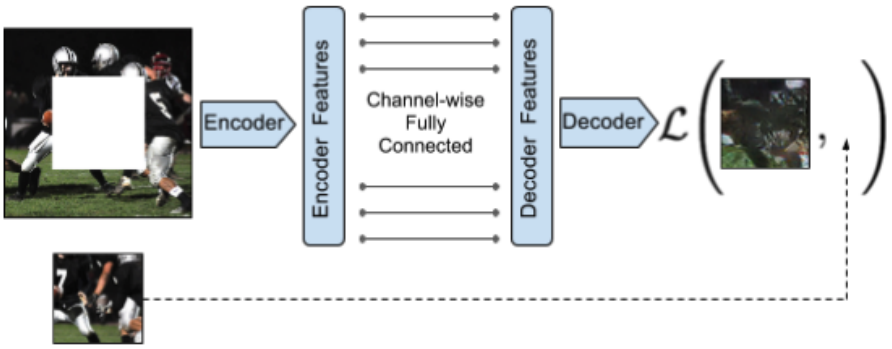

Figure 4: Overview of Architecture [3]

Iizuka et al. proposed a deep CNN model with global and local discriminators as adversarial losses for the image completion task [3]. The model uses a completion network for image completion, which follows an encoder-decoder structure. Additionally, it uses dilated convolutions in the completion network to replace the channel-wise fully-connected layer adopted in Context Encoders mentioned previously. The model also has two additional networks, the global and the local context discriminator networks, which are used to train this network to realistically complete images. The global discriminator assesses the entire image of the predicted output, while the local discriminator network takes only a small region centered at the generated area to enforce the local consistency. This model achieved an accuracy of 77% on the final result on CelebA dataset (dataset of celebrity pictures). Figure 5 shows an overview of the architecture.

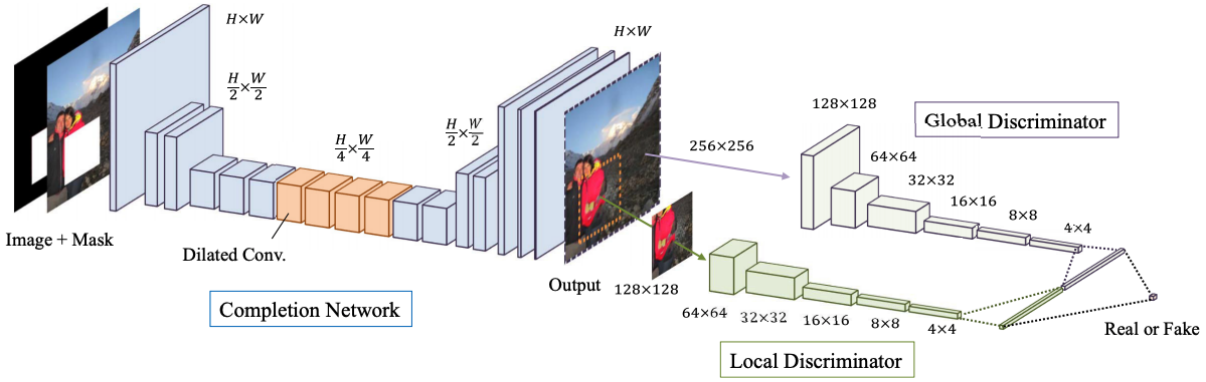

Figure 5: Overview of Architecture [4]

4.0 Data Processing

4.1 Dataset Description

The dataset used was sourced from Kaggle which comprises 25,000+ labelled images intended for computer vision projects. From this dataset, 2000 images belonging to the buildings and cityscape image class were selected to train the model on. The team limited the scope to this image class because they were predicted to be easier to inpaint due to their stylistic and geometric similarities. Please see Figure 6 for a few representative examples of the images within the dataset.

Figure 6. Sample Original Images from Dataset

4.2 Data Preprocessing

A simple data processing pipeline was developed to process images from the dataset before they were loaded into the model for training as seen in figure 7.

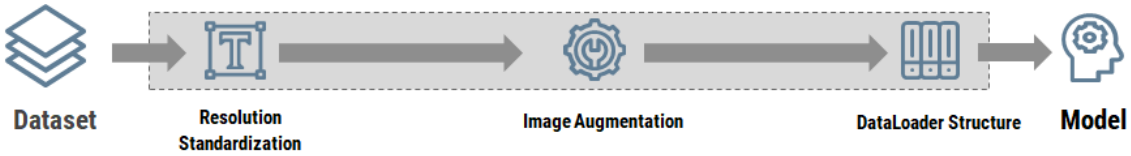

Figure 7: Data Processing Pipeline Overview

The first step is image standardization where each image is resized from an original size of 150x150 to a size of 128x128. Afterwards, two augmentation techniques are performed on each image which is mirroring the image on the y-axis and rotating the image 180°. The team believes these augmentations are sufficient in improving model robustness as the model will be able to learn and inpaint invariant representations of the original image. After augmentation, four new images will be generated for each input image. A diagram of the augmentation logic is found in figure 8.

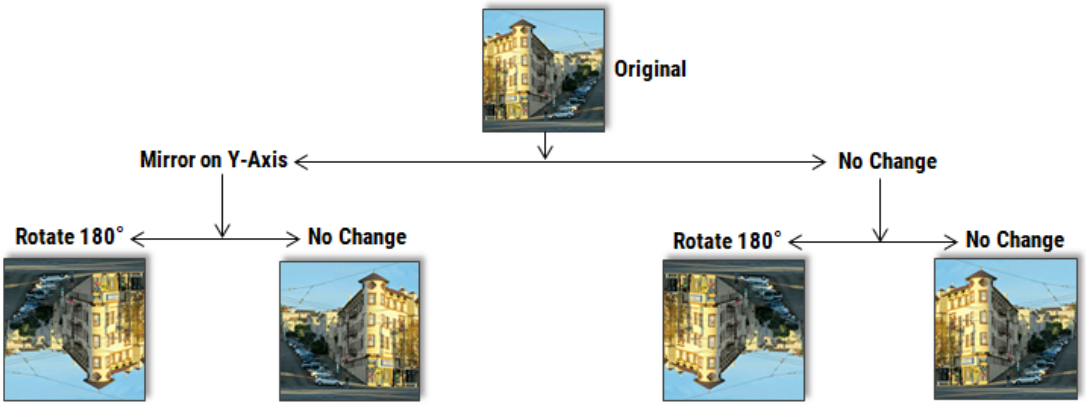

Figure 8: Image Augmentation Logic

As the last step, the correct data structure must be loaded into the Pytorch DataLoader tool before being ingested by the GAN neural network. The first of two components in the data structure is a tensor of an incoming 3-channel image. The other component is a tensor of the same image with a missing region in the middle of size 24x24 concatenated with a binary mask that highlights the area that the network should inpaint. This will create an image with 4-channels with the binary mask occupying the fourth channel; see figure 9 for a visual example. Additionally, key statistics of the finalized dataset after passing through the data processing pipeline is summarized in Table 1.

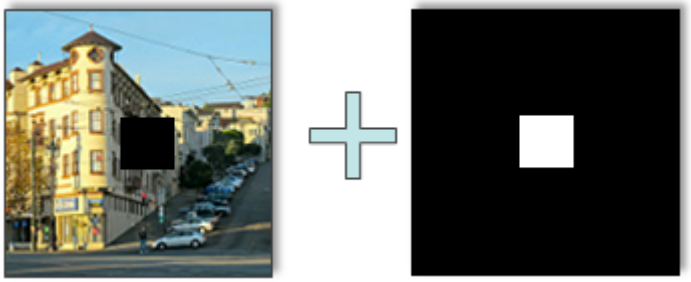

Figure 9: Composition of 4-Channel Image: Left Image and Right Image Concatenated, (Left) Original Image with Missing Region, (Right) Binary Mask

Image Size	128 x 128 Pixels
Number of Images	8,000
Augmentations Performed	Mirror on Y-axis, Rotation of 180°
Missing Region Size, Positioning	24 x 24, Image Center
DataLoader Structure	Input 1 - Type: Tensor, Size: (3, 128, 128) Input 2 - Type: Tensor, Size: (4, 128, 128)

Table 1: Statistics of Processed Dataset

5.0 Architecture

Our deep convolutional generative adversarial network (DCGAN) architecture consists of two networks: generator and discriminator. 2-D batch normalization is added after each layer in both networks so that they can stabilize the learning process and dramatically reduce the number of training epochs required [5].

5.1 Generator

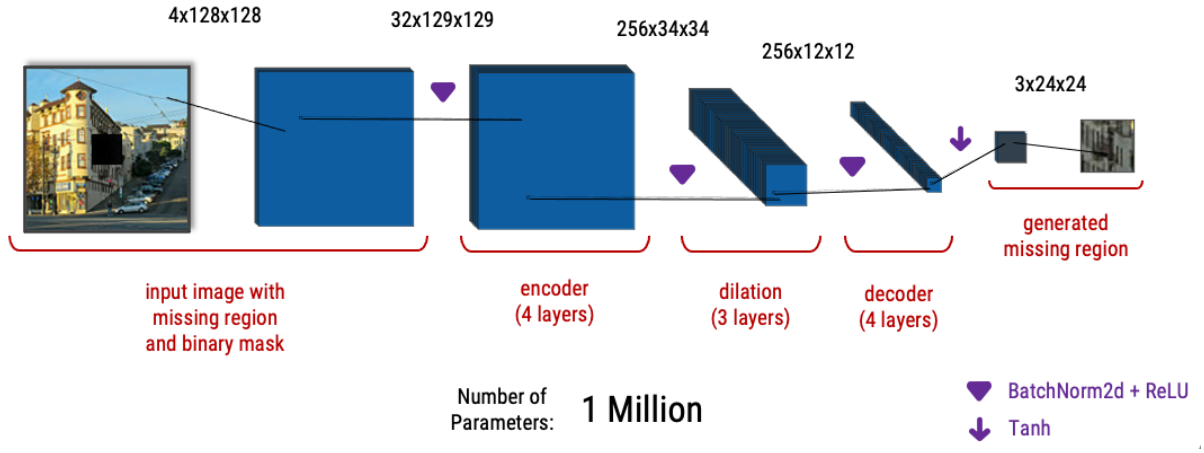

Figure 10: Generator architecture

The generator is a generative convolutional autoencoder model, which is shown in Figure 10. The input to the generator model is a RGB image with a binary channel that indicates the image completion mask, and it also can be represented by a tensor that contains a batch of 4-channel images (3 RGB channels plus 1 channel for binary mask) of size 128x128. The output of the generator is a reconstructed image for the missing region, which can be represented by a tensor that contains a batch of 3-channel RGB images. The generator consists of 4 layers of strided convolutional layers, 3 dilation layers, and 4 fractionally-strided convolutional layers. Dilated convolutional layers are used at lower resolutions, to detect the large missing area in the input image and provide spatial support to the large missing area so that the model can include pixels outside of the missing area while computing output pixels [6]. Finally, the output of the generator is fed through a tanh function to return it to the input data range of [−1,1]. The detailed architecture of the generator model can be seen in Table 2.

Type	Kernel	Dilation	Stride	Output
Conv.	2 x 2	1	1 x 1	32
Conv.	2 x 2	1	1 x 1	64
Conv.	2 x 2	1	2 x 2	128
Conv.	2 x 2	1	2 x 2	256
Dilated Conv.	2 x 2	4	1 x 1	256
Dilated Conv.	2 x 2	8	1 x 1	256
Dilated Conv.	2 x 2	16	1 x 1	256
Transposed Conv.	1 x 1	1	1 x 1	128
Transposed Conv.	1 x 1	1	1 x 1	64
Transposed Conv.	1 x 1	1	1 x 1	32
Transposed Conv.	2 x 2	1	2 x 2	3

Table 2: Detailed Architecture of Generator

5.2 Discriminator

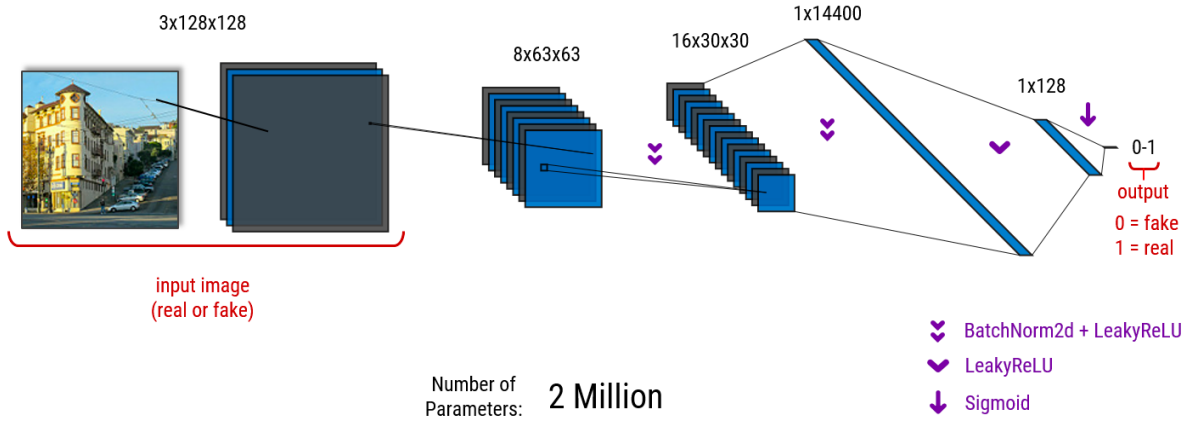

Figure 11: Discriminator architecture

The input to the discriminator model is a tensor representing a batch of 3-channel images of size 128x128. The discriminator consists of 2 convolutional layers followed by 2 linear layers as outlined in Figure 11. Following each convolutional layer, batch normalization and the leaky ReLU activation function with the slope of 0.01 are used. Leaky ReLU is also used between the linear layers. While leaky ReLU activation ensures that the neurons do not get stuck in a state where they continuously output 0 for all inputs by allowing small negative values to pass through [7]. Finally, a sigmoid activation is used to ensure the result is between 0 and 1. A result less than 0.5 indicates the discriminator finds the image to be fake, while a result greater than equal to 0.5 indicates the image is real to the discriminator. The detailed architecture for each layer in the discriminator can be seen in Table 3.

Type	Kernel	Padding	Stride	Output
Conv.	5 x 5	1	2 x 2	8
Conv.	5 x 5	0	2 x 2	16
Linear	N/A	N/A	N/A	128
Linear	N/A	N/A	N/A	1

Table 3: Detailed Architecture of Discriminator

6.0 Baseline Model

The baseline model that will be used is an image inpainting algorithm based on the Fast Marching Method (FMM) [8]. The FMM mathematical algorithm uses an extrapolation technique based on pixel information surrounding the missing region such as gray value (brightness of a pixel) and gray gradient. The FMM model is simple to implement as it is accessible through a method in the OpenCV computer vision library and only requires two image inputs [9].

The team will use the results from the FMM model and compare it with the final iteration of our ML algorithm. To do so, a pixel-wise accuracy metric will be utilized that calculates the difference between each pixel value in the inpainted region. Additionally, an extra visual comparison step will be conducted to evaluate the visual consistency of the inpainted region. It was found that the baseline model performed well when the surrounding contextual information is similar to the inpainted region. Conversely, inpainted regions with high

geometric complexity did poorly. Examples of the good and poor use-cases of FMM is seen below.

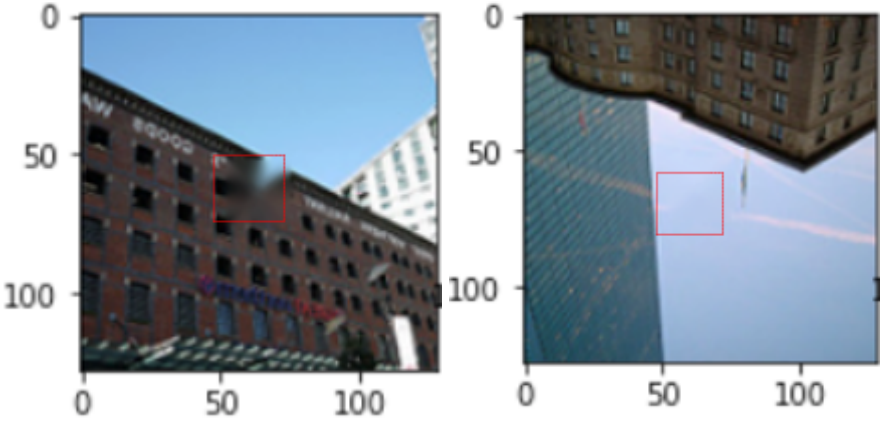

Figure 12: showcase of inpainted results from baseline model, (left) bad result, (right) good result

7.0 Quantitative Results

7.1 Loss

Unlike other architectures, the loss curves for GANs do not directly correlate with the model's performance. For a good model, instead of the loss curves converging to a lower value than their initial, loss curves of GANs fluctuate: a higher loss for the generator would correspond to a lower loss for the discriminator and vice versa. Figure 13 shows the loss curves for the generator and discriminator of our model. The large difference between the training and validation curves in both figures in the latter half of training shows possible overfitting on the training dataset.

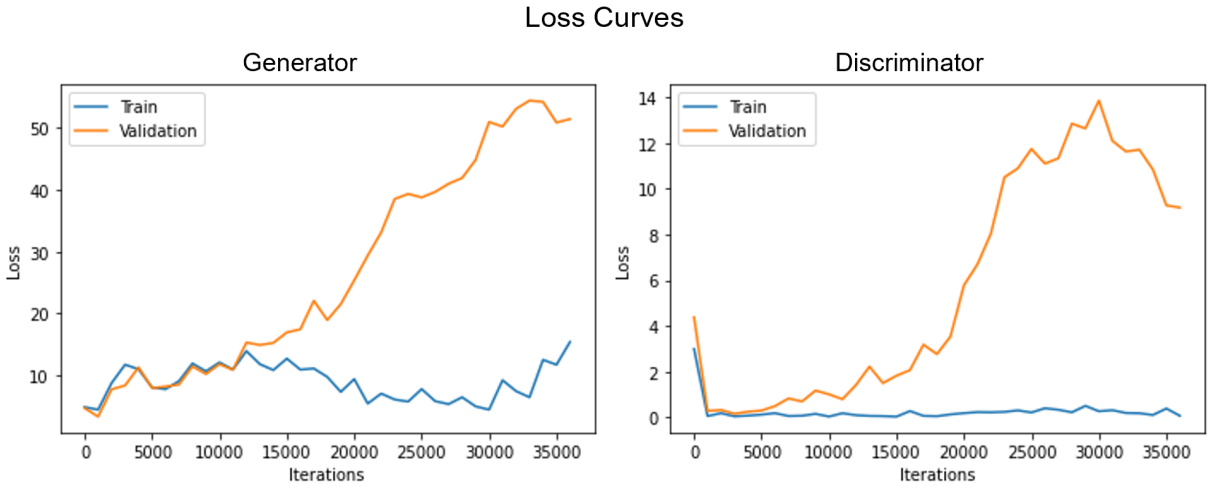

Figure 13: training loss curves for both generator (left) and discriminator (right).

7.2 Pixel-wise Accuracy

$$A_{pixel\ wise} = 1 - \frac{\sum_{i=1}^{N_i}\sum_{p=1}^{N_{sp}}|p_t - p_g|}{N_i \times N_{sp} \times 255}$$

N_i: number of images

N_{sp}: number of subpixels per image. There are three subpixels (RGB) per pixel.

p_t: value of target subpixel (range: 0-255)

p_g: value of generated subpixel (range: 0-255)

Figure 14: pixel-wise accuracy equation.

While the goal of the generator is to generate visually consistent images that need not be identical to the original images which are framed for qualitative analysis, our quantitative approach is to measure pixel-wise accuracy. Figure 14 defines pixel-wise accuracy and is used to measure the similarity between the generator output and the corresponding region from the original image (target). Figure 15 illustrates the pixel-wise accuracy curve of our model during training with the training and validation dataset converging to 60% and 43%, respectively. On the validation data, compared to the 60.8% achieved by the baseline model, our model performs worse. The difference between the training and validation curves in the latter half of training also shows possible overfitting on the training dataset.

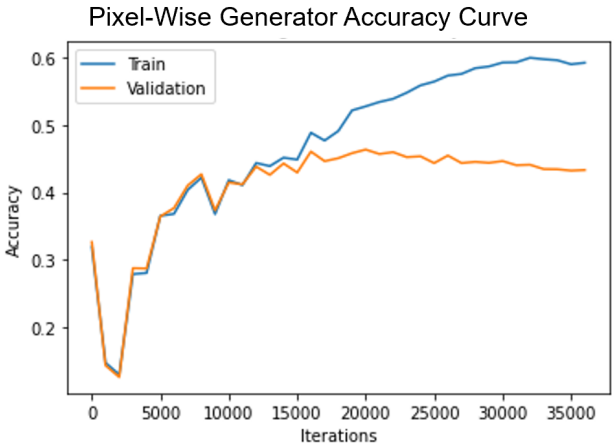

Figure 15: training pixel-wise generator accuracy curve.

7.3 Pixel-wise Accuracy on Test Dataset

Our model achieved 42.3% pixel-wise accuracy on our test dataset of 200 images which is worse than the 60.5% achieved by the baseline model. Figure 16 illustrates images that our model performed better or around the same as the baseline model while Figure 17 are images that our images performed much worse.

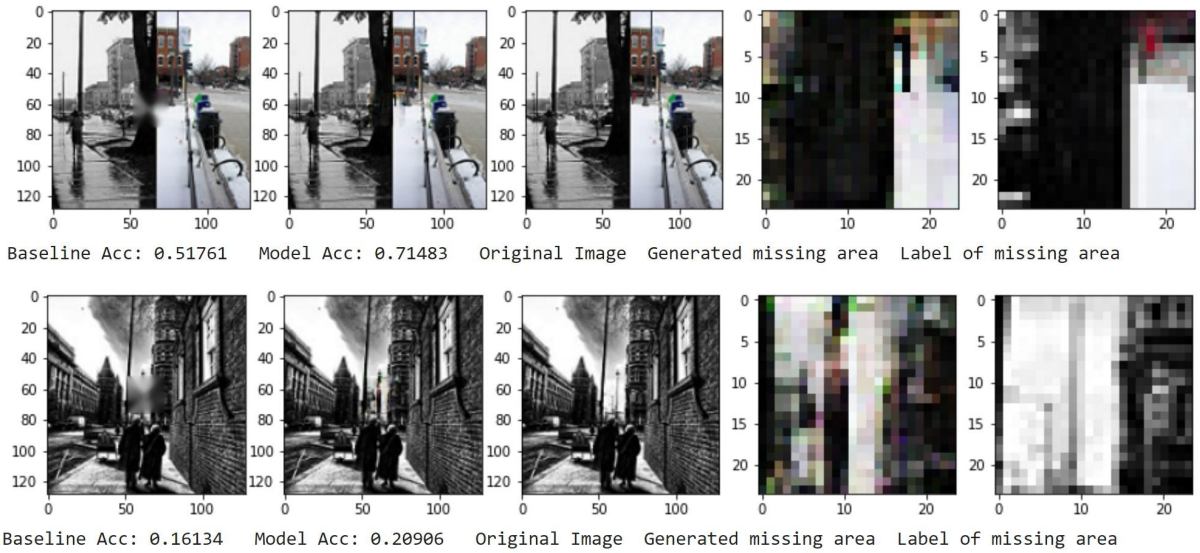

Figure 16: two selected images from the test dataset on which our model performed well. For each row, from left to right: baseline model output superimposed onto the original image, generator output superimposed onto the original image, original image, generator output, ground truth.

Figure 17: two selected images from the test dataset on which our model performed poorly. For each row, from left to right: baseline model output superimposed onto the original image, generator output superimposed onto the original image, original image, generator output, ground truth.

8.0 Qualitative Results

Figure 18 illustrates an image on which our model's output is more visually consistent with the rest of the image while the accuracy is lower than the baseline model's output. Figure 19 illustrates an image on which our model performed poorly on the generated missing region because the output is strongly influenced by the poster on the wall to the bottom right of the missing region. Figure 20 illustrates that our model performed worse on the image with the missing region filling in the sky than the baseline model.

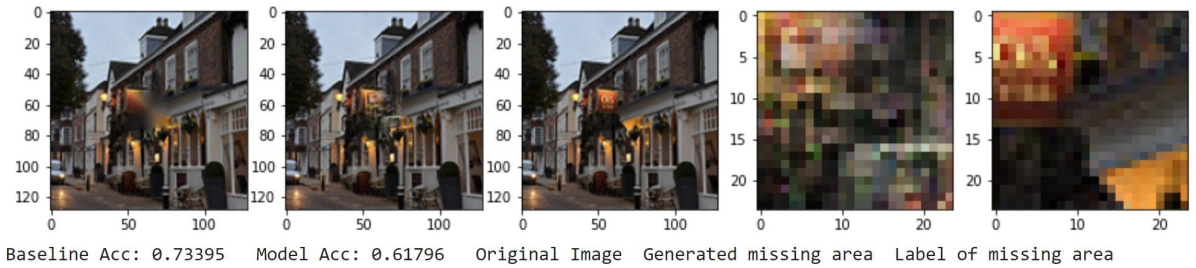

Figure 18: an image from the test dataset shows that our model's output is more visually consistent than the baseline model's output. For each row, from left to right: baseline model output superimposed onto the original image, generator output superimposed onto the original image, original image, generator output, ground truth.

Figure 19: an image from the test dataset on which our model's output is strongly influenced by the poster on the wall to the bottom right of the missing region. For each row, from left to right: baseline model output superimposed onto the original image, generator output superimposed onto the original image, original image, generator output, ground truth.

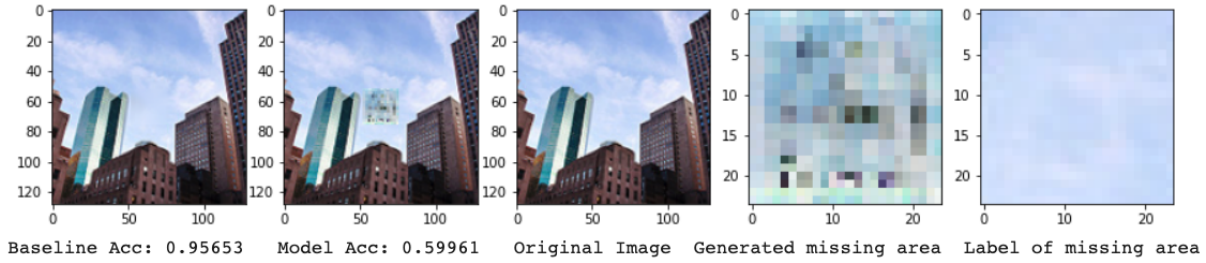

Figure 20: an image from the test dataset on which our model performed poorly because our dataset is focused on buildings. For each row, from left to right: baseline model output superimposed onto the original image, generator output superimposed onto the original image, original image, generator output, ground truth.

9.0 Model Performance on New Data

Pictures of UofT Campus buildings were collected to test out model performance on new data that has never been seen before. The images selected were chosen due to their diversity in architectural styles that are representative of the types of buildings images found in the training dataset. Good and bad results can be found in Figure 21 and Figure 22 respectively along with a summary of the accuracy scores found below.

	Baseline Accuracy	Model Accuracy	% Difference (Baseline - Model)
Good Results	73.0%	60.4%	12.6%
	72.9%	61.9%	11.0%
Bad Results	71.8%	49.8%	22.0%
	84.4%	52.5%	31.9%

Table 4. Accuracy Scores of Performance on New Data

9.1 Good Results

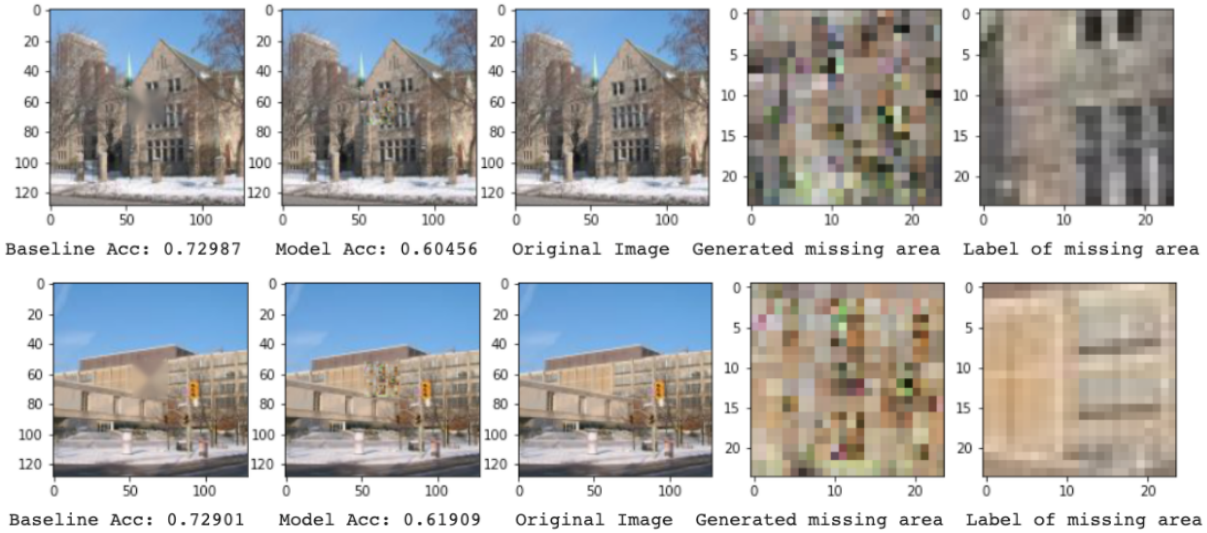

Figure 21: Relatively Good Results from Tests on New Data

9.2 Poor Results

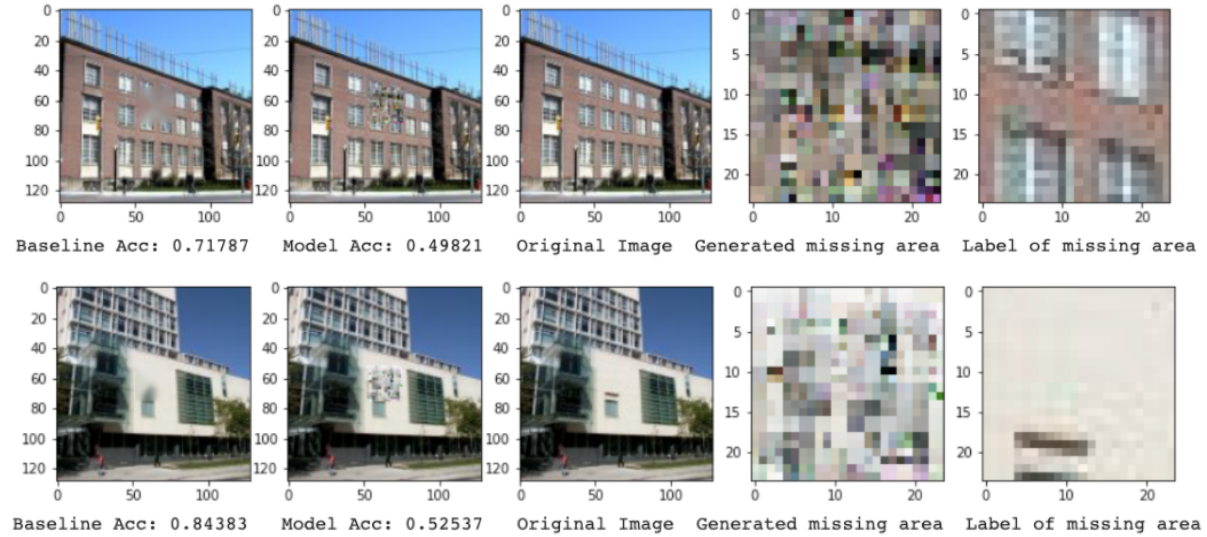

Figure 22: Relatively Poor Results from Tests on New Data

10.0 Discussion

In terms of our results, we noticed that for images with building-like elements such as straight lines and rectangular shapes as in Figure 16, our model can accurately inpaint the image. Whereas with significantly simpler images, such as the sky in Figure 20, our model struggles to inpaint. Similarly, inpainting repetitive patterns such as windows, as in Figure 17, the model once again struggles. While these generated outputs are not visually consistent, our model can still generate colours consistent to the original image. The occasional visual inconsistency may be due to our choice to limit the missing region to the 24x24 center region. As for most of the images in our dataset, this center region captures the edge of buildings. So when this region contains outliers such as the plain surface or repetitive patterns, our model struggles. This behaviour could be mitigated if the location of the 24x24 region was randomized for each image in the dataset. This would allow the model to encounter more variety and it would make the model more applicable to real world uses.

An interesting observation was that earlier in the training process, before the generated output will resemble the 24x24 center region of the original image, it will first resemble the entire original image downscaled to a 24x24 image. This observation is consistent with the behaviour of autoencoders, which is the architecture of our generator model albeit being unsymmetrical. We initially had experimented with other generator architectures such as an encoder. However, we found that our model overfitted more (Figure 23). Ultimately, the autoencoder architecture produced the best results.

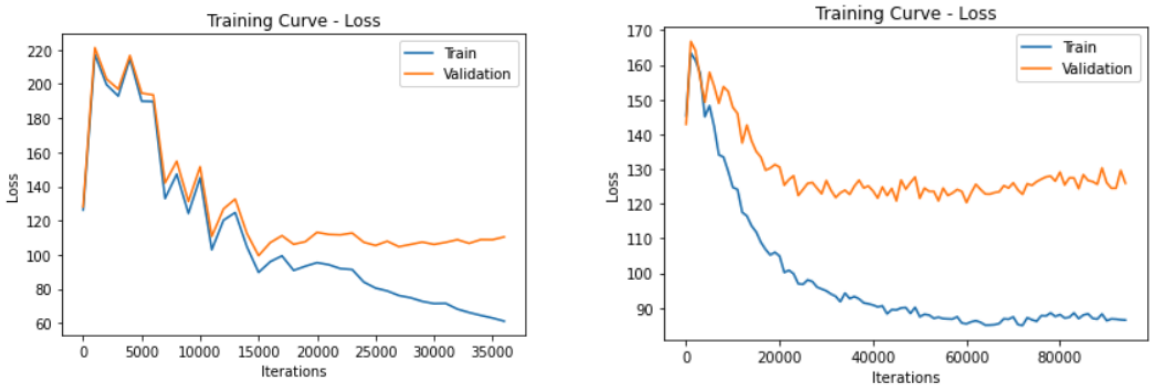

Figure 23: Training loss for autoencoder with 1500 epochs (left) and training loss for encoder with 2000 epochs (right).

Additionally, we found that it was difficult to quantitatively gauge the accuracy of our model and consequently do quantitative comparisons between our model and the baseline model. The goal of the DCGAN model was not to reconstruct the original image but instead to inpaint the image with visual consistency. This means that there is no ground truth on which we can evaluate our model nor using pixel-wise accuracy. Thus, model evaluations often had to be done through our visual observations.

11.0 Ethical Considerations

Given the scope of our model being limited to inpainting buildings and cityscape images, there are few ethical concerns associated with its use-case. The major consideration is around collecting images that infringe on private property. If the existing dataset for training were to

be expanded in the future, the team must ensure collected images abide by the aforementioned consideration.

In the context of expanding our model capabilities to inpaint other image classes, more ethical considerations arise. In the application of using the model to remove unwanted artifacts, an end-user can easily distort the contents of an image in malicious ways with the intention to deceive viewers [10]. For instance, certain individuals could be removed by inpainting where they once were using the surrounding contextual information. This manipulated image and its resulting consequences could then be falsely propagated by social media users and news outlets to damaging effect.

Model Limitations:

- The model is only trained to inpaint images of buildings and cityscapes. To expand the capability and scope of image classes that the model can inpaint, the model must be re-trained on a separate dataset with the desired image class.
- The in-painted region must be square in geometry as the model will be trained to complete such missing portions.
- The current model is trained to inpaint 128x128 images. If the desired resolution of images to inpaint were to increase, computation times would drastically increase.

Training Data Limitations:

- Obtaining images is not a concern as there are many datasets available, and removing portions of the images for training can be automated.

12.0 References

[1] “Recently Uploaded Images View All,” Inpaint. [Online]. Available: https://theinpaint.com/. [Accessed: 05-Dec-2020].

[2] F. Altinel, M. Ozay, and T. Okatani, “Deep Structured Energy-Based Image Inpainting,” 2018 24th International Conference on Pattern Recognition (ICPR), 2018.

[3] "Feature Learning by Inpainting", People.eecs.berkeley.edu, 2020. [Online]. Available: https://people.eecs.berkeley.edu/~pathak/papers/cvpr16.pdf. [Accessed: 17- Oct- 2020].

[4] "Globally and Locally Consistent Image Completion", Iizuka.cs.tsukuba.ac.jp, 2020. [Online]. Available: http://iizuka.cs.tsukuba.ac.jp/projects/completion/data/completion_sig2017.pdf. [Accessed: 17- Oct- 2020].

[5] "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", 2016. [Online]. Available: https://arxiv.org/pdf/1511.06434.pdf. [Accessed: 16-Oct-2020].

[6] “DilatedNet - Dilated Convolution (Semantic Segmentation)”, 2019. [Online]. Available: https://towardsdatascience.com/review-dilated-convolution-semantic-segmentation-9d5a5bd768f5 [Accessed: 9-Dec-2020].

[7] "A Short Introduction to Generative Adversarial Networks", 2018. [Online]. Available: https://sthalles.github.io/intro-to-gans [Accessed: 16-Oct-2020].

[8] "An Image Inpainting Technique Based on FMM", 2020. [Online]. Available: https://www.researchgate.net/publication/238183352_An_Image_Inpainting_Technique_Based_on_the_Fast_Marching_Method. [Accessed: 17- Oct- 2020].

[9] "OpenCV: Image Inpainting", Docs.opencv.org, 2020. [Online]. Available: https://docs.opencv.org/master/df/d3d/tutorial_py_inpainting.html. [Accessed: 17- Oct- 2020].

[10] "Home Page", Cs.stanford.edu, 2020. [Online]. Available: https://cs.stanford.edu/people/eroberts/cs181/projects/photoethics/default.htm. [Accessed: 17- Oct- 2020].