Deep Learning based Generative Inpainting for VR

Tejas Munees Sivaramakrishnan B.Tech

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Intelligent Systems)

Supervisor: Aljosa Smolic

September 2020

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Tejas Munees Sivaramakrishnan

September 6, 2020

Permission to Lend and/or Copy

Acknowledgments

I would like to begin by thanking my professors, specially my dissertation supervisor, Prof. Aljosa Smolic for his constant support and guidance that he extended towards this project.

I would also like to use this opportunity to extend my gratitude to my dissertation co-supervisor, Dr. Cagri Ozcinar, for his constant support and valuable inputs throughout the project.

I would also like to thank my friends from different parts of the world, for their constant encouragement that, we are all in this together, during this difficult year.

Last but not the least, I am forever indebted to my loving family, as none of this would be possible without their support and encouragement.

Tejas Munees Sivaramakrishnan

University of Dublin, Trinity College
September 2020

Deep Learning based Generative Inpainting for VR

Tejas Munees Sivaramakrishnan, Master of Science in Computer Science

University of Dublin, Trinity College, 2020

Supervisor: Aljosa Smolic

The core objective of this study is to be able to successfully inpaint 360° images. The need to inpaint 360° images comes due to high occurrences of stitching artefacts and unwanted objects, while capturing the image. In this study, we begin by exploring existing state of the art inpainting techniques. It is found that, existing state of the art inpainting techniques focus on inpainting 2D images, and that, there are very few studies that discuss inpainting 360° images. We study in-depth about generative adversarial networks which is extensively used in inpainting tasks to produce novel image content. We also explore different projections of spherical image to better understand handling distortion. We propose a network that transforms a spherical image into different projections and try to utilise existing state of the art techniques to perform inpainting. Overall, the results from our model are at par or better than existing state of the art models, with scope for better performance based on better techniques to handle distortion. Therefore, we also propose future work that can potentially handle distortions within the inpainting network.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

From gaming and movie industries to tourism industry and to healthcare industry, the past decade has seen a tremendous growth in virtual reality (VR) utilisation. This trend has not reached its peak and continues to pick up momentum every day. The growth is attributed to one of VR's key deliverable, immersion. This has also motivated hardware manufacturers to produce more VR headsets and powerful graphics cards that enable everyone to consume VR content seamlessly. While VR contents can be artificially created, as it is in the case of gaming content, use-cases such as tourism and movies use images and videos of actual locations. Such content is generally Omni-directional images (ODIs) and videos respectively. In this research the focus is on ODIs.

Omni-directional images also known as 360° images are generally captured through a panoramic shot or by stitching multiple images captured by multiple cameras. However, such methods of capturing are not without issues that could impact immersion and privacy amongst others. Immersion is impacted when overlapping images are incorrectly stitched while obtaining a spherical panorama, creating a visible seam. The issue of privacy occurs when unwanted objects get included while capturing VR content and is often tackled in post-processing, like in Street View functionality of Google Maps, where sensitive data like people's faces or vehicle number plates are blurred. There are also instances where the image data is missing due to camera's limitation. These defeat the purpose of VR by providing the consumers an unrealistic experience.

Image inpainting is one of the common techniques used to correct errors such as correcting damaged or missing parts of an image, and could be extended to issues discussed earlier. Image inpainting has been a widely researched area and has been around for a while. However, the most commonly researched topics in the field of image inpainting has been about image restoration with respect to historical artefacts, image denoising, or even image restoration of physical photographs with cracks formed over time [12, 13]. Few of these studies have also researched deep learning methods to perform the same with the advent of better computational technologies. Researchers have been invested in novel ways of image inpainting even more so, ever since the advent of Generative Adversarial Networks (GANs) by Ian J. Goodfellow et al., 2014 [14].

In this research, the focus is on using such generative networks to inpaint ODIs successfully. We begin by studying generative adversarial networks, followed by different state of the art inpainting techniques. We also explore different 360° image representations to handle distortions. We then try to leverage existing state of the art techniques used for 2D image inpainting and implement it for inpainting 360° images with suggested changes to the image representation to handle distortion and discuss our findings along with future work and way forward.

1.2 Motivation

The motivation for this research is based on the increased VR consumption across various industries, including film, gaming, retail marketing, automotive, sports, amongst others.

While extensive literature is available for various inpainting techniques including those of generative methods, after a thorough literature review it was observed that very few focus on ODI images, and instead are aimed towards inpainting for regular 2D images whose field of view is very less compared to that of ODIs. This allows for a good opportunity to explore and record the findings for the same, whose solution would benefit extensively for VR content.

Chapter 2

Background and Related Work

2.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are very powerful among the generative model family that also includes models like Autoencoders. The very core of the GAN is in performing latent space interpolation to produce the desired output. Here, the output could be ranging from anything as simple as digits, to outputs that are as complex as facial features or indoor and outdoor scene creation.

While GANs can be designed using both Convolutional Neural Networks (CNNs) [15] and Recurrent Neural Networks (RNN) using Long Short-Term Memory (LSTM) [16], for image generation task we would be looking at Convolution based GANs in this research. There are various types of GANs, namely Deep-Convolutional GANs (DC-GANs) which are the basic form of GANs involving CNNs, whose output is random yet meaningful (desired latent space). To remove the randomness of the output, there are Conditional GANs (CGANs) which allows control over the output of GANs. Then, there are Wasserstein GANs (WGANs) which have a different loss function allowing it to be more stable than GANs during training [1]. We also have Least-square GANs (LSGANs) which aim at not only being stable, but also aims at producing high perceptive quality during output. We also have Disentangled representation of GANs like InfoGAN, and Cross-Domain GANs like CycleGAN. However in this study, we would be focussing more on the implementations of WGANs and Improved WGANs both of which have been proven to be more stable and effective at inpainting tasks [1, 6, 7].

2.1.1 GANs

Generative Adversarial Networks was proposed by Ian J. Goodfellow et al., [14]. GANs are broadly made up of two components, the generator and the discriminator. The generator tries to produce new outputs trying to pass it as a sample taken up from training data, where as the discriminator acts as a critic, trying to identify the fake (generator's output) from the real image taken from the training data. Both these networks together make up the adversarial part, and corresponds to a minimax two-player game [14]. The generator also takes noise input z, so as to represent the real data distribution, making the samples not only realistic, but also diverse.

One of the analogy the author provides in his paper to the GANs is that of a scenario between a police and counterfeiter[14], where the police tries to identify between real and fake currency and the counterfeiter tries to pass their counterfeit money as real, and as time progress the police gets better at finding the difference, and explains the counterfeiter why it was found as fake, and the counterfeiter uses this feedback to do better next time. Here the counterfeiter and police corresponds to the generator and discriminator model respectively. As seen in Figure 2.1, while training discriminators, the gradients are not passed on to generator, however, when the data generated by generator is passed on posing as data from training sample to the discriminator, the weights of discriminator are frozen and the updates are passed on to the generator for updates.

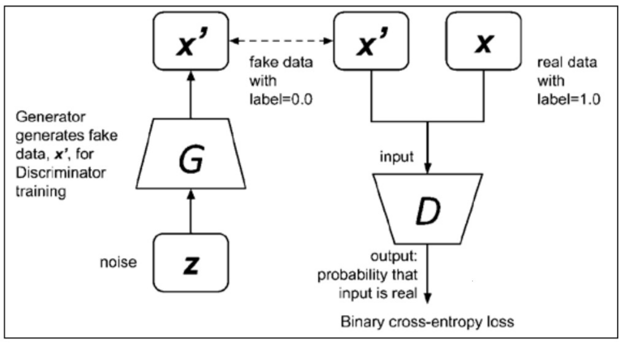
x'
x'
x
fake data with label=0.0
real data with label=1.0
Generator generates fake data, x', for Discriminator training
G
input
D
noise
z
output: probability that input is real
Binary cross-entropy loss

Figure 2.1: Training the Discriminator in a GAN [1]

The network architecture of a discriminator is similar to a CNN based classifier which comprises multiple sets of Convolution layer followed pooling layer and/or batch normalisation, flattened and connected to a fully connected neural network giving outputs as fake or real. The network architecture of a generator, is similar to a decoder network of the autoencoder. The generator takes noise vector as input and passes it through a fully connected neural network whose outputs are reshaped and passed through convolution layers with fractional strides, using which the final layer of an activation function generates an image.

The discriminator expects to maximise $\mathbb{E}_{x \sim p_{data}(x)} \log D(x)$ for real samples and maximise $\mathbb{E}_{z \sim p_z(z)} \log(1 - D(G(z)))$ for fake samples, where $D(x)$ and $G(z)$ are the outputs of discriminator and generator given input x and noise z respectively [1, 17]. The generator however, tries to fool the discriminator to predict its output as a sample taken from training data and therefore tries to minimise $\mathbb{E}_{z \sim p_z(z)} \log(1 - D(G(z)))$ [17]. Combining these, we get the loss function to be [1],

$$\min_G \max_D L(D, G) = \mathbb{E}_{x \sim p_{data}(x)} \log D(x) + \mathbb{E}_{x \sim p_g(x)} \log(1 - D(x)). \tag{2.1}$$

Equation 2.1 corresponds to binary cross-entropy loss.

At a certain point, the discriminator will no longer be able to identify the difference between generator's output and training data. This is a point where the probability of the image coming from the generator's distribution p_g, and the probability of the image coming from the training data distribution p_{data} are equal to $\frac{1}{2}$ respectively. It is at this point where p_g equals p_{data}, we achieve global optimum as $-\log 4$. On expanding Jensen Shannon (JS) Divergence between p_g and p_{data} [18, 19], we get,

$$L(D^+, G) = 2D_{JS}(p_{data}||p_g) - \log 4. \tag{2.2}$$

At the global optimum of $L(D^+, G^+)$, the divergence between two distributions become zero. This explains that the loss function, basically tries to calculate the JS Divergence between both distributions, p_g and p_{data}.

2.1.2 Challenges with GANs

GANs are extremely difficult to train. Achieving Nash equilibrium, avoiding vanishing gradient or mode collapse are just few of the major obstacles in training GANs [17].

Achieving Nash equilibrium is important to achieve the desired latent space to generate samples from. However, it is extremely challenging as both Generator and Discriminator aims to minimise their cost functions independently in a non-cooperating mini-max game [20]. This makes training highly unstable.

In Figure 2.2, we can see an analogous case of difficulty in achieving Nash equilibrium on a non-cooperating game, with players x and y trying to minimise xy and $-xy$ independently with learning rate η after n iterations [19]. We see overtime, the instability increases with cost function xy having high oscillations.

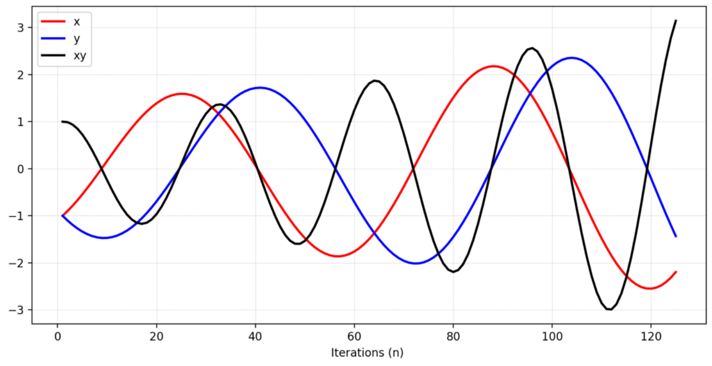

Figure 2.2: Cost Function of a two player non-cooperating minimax game

One of the other challenges is in handling vanishing gradient. This often occurs if the discriminator becomes robust very quickly, even before generator learns to produce any output closer to the distribution from the desired latent space. If the discriminator learns too quick, then the loss function eventually decreases enough that the generator does not receive any gradient [19]. On the contrary, if the discriminator is not capable of learning enough, the generator receives inaccurate gradients for update, and the output is not meaningful as they will not be from the desired latent space. The balance between training speed of discriminator and generator therefore becomes a challenge.

One the most common obstacle faced during training GANs are mode collapse. Mode collapse is a case when the generator always produces very similar output that are not in the desired latent space. Mode collapse is still an open ended problem with techniques to tackle it still being researched. A common reason of mode collapse is when gradients become zero. There may exist certain combination in latent space, where the generator is able to successfully fool discriminator without being able to achieve the desired real-world distribution. This latent space, may be small enough without much diversity, forcing the generator to produce same undesired outputs repetitively, ending up with mode collapse.

2.1.3 WGANs and Improved WGANs

Martin Arjovsky et al., [21] proposed an alternate to training of GAN known as Wasserstein GAN (WGAN). They proposed Wasserstein Distance, also known as Earth Mover's Distance, as an alternate metric to JS divergence. As the name suggests, fundamentally they are analogous to the mass $\gamma(x, y)$ required to be moved by distance d, for the probability distribution p_{data} to match p_g, where d is given by $||x-y||$ [1]. Here $\gamma(x, y)$ is one of the possible joint distributions of all possible distributions $\Pi(p_{data}, p_g)$. While there are different possible $\gamma(x, y)$ that can help match the distribution p_{data} and p_g, we are interested in the one that is least cost. Thus the Wasserstein distance is given by [21],

$$W(p_{data}, p_g) = \inf_{\gamma \sim \Pi(p_{data}, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||]. \tag{2.3}$$

To understand why Wasserstein metric provide more stability while training, it is useful to consider two non-overlapping distributions p_g and p_{data}. The KL divergence between such distributions, $D_{KL}(p_{g/data}||p_{data/g})$ is $+\infty$. While the JS divergence is a finite value, it is still a constant, allowing the GAN to have insufficient gradient to match the distributions [1]. However, with Wasserstein distance, $W(p_{data}, p_g)$ gives a smooth function [1, 21]. This is similar to the case when two distributions are parallel lines as mentioned in [21]. The authors also propose Kantorovich-Rubinstein Dual of equation 2.3 as given in Equation 2.4 [21], which transforms from finding the infimum as in Equation 2.3 to supremum as in Equation 2.4, over all K-Lipschitz functions f, where $K = 1$. This is done to avoid the difficult task of exhausting all possible joint distributions from $\Pi(p_{data}, p_g)$ [17].

$$W(p_{data}, p_g) = \frac{1}{K} \sup_{||f||_L \leq K} \mathbb{E}_{x\sim p_{data}}[f(x)] - \mathbb{E}_{x\sim p_g}[f(x)]. \tag{2.4}$$

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2|. \tag{2.5}$$

However, to enforce Lipschitz continuity during training as given in Equation 2.5, the authors proposed weight clipping with bounds (for example $[-0.01, 0.01]$). The study [21] also suggests that, weight clipping allows for smaller space for w, allowing it to be Lipschitz continuous. By finding f_w, one of the family of K-Lipschitz continuous function, we obtain equation 2.6 [1, 17, 21]. Therefore, equation 2.6 is the loss function that the generator and discriminator aims to minimise and maximise respectively.

$$W(p_{data}, p_g) = \max_{w \in W} \mathbb{E}_{x \sim p_{data}}[f_w(x)] - \mathbb{E}_z[f_w(G(z))] \tag{2.6}$$

However, WGAN still suffers from training instabilities due to vanishing gradients and gradient convergence because of weight clipping. If weight clipping is too small, the gradients back propagated are small leading to vanishing of gradients. If the weight clipping is too large, there could be exploding gradients as well [22]. Ishaan Gulrajani et al., [23] proposed an improved WGAN known as WGAN with gradient penalty (WGAN-GP) that uses gradient penalty instead of weight clipping to achieve Lipschitz continuity without WGAN's training instabilities. Using 1-Lipschitz's property, the authors suggest a gradient penalty as in equation 2.7 [23].

$$\lambda \mathbb{E}_{\hat{x} \sim P_{\hat{x}}}(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2. \tag{2.7}$$

In equation 2.7, $\hat{x}$ is sampled along straight lines between points sampled from distributions P_{data} and P_g [24].

2.2 State of the Art Inpainting Techniques

2.2.1 Traditional Inpainting

Inpainting is an old technique of restoring physical copies of painting, pictures or photos. Physical mode of inpainting using artists or a conservator-restorers is still very common today in museums across the world, where old paintings deteriorate over time. Before the advent of smart phone cameras, people who had physical copies of photos faced similar issues. They also faced more wear and tear with usage. These included scratches, colour deterioration, smudges amongst others. These were and still are repaired using inpainting techniques as seen in Figure 2.3, where the left image shows the damaged image with scratches, and the right image shows the inpainted output.

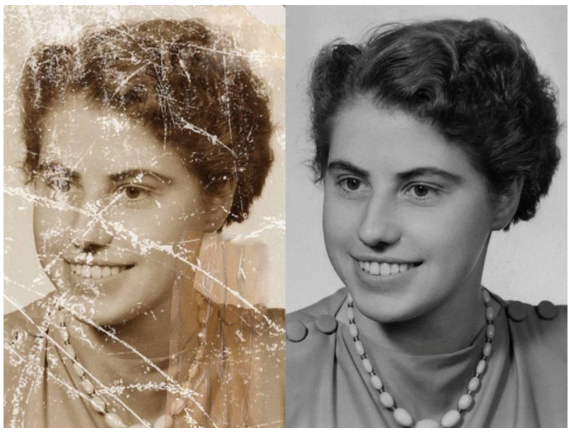

Figure 2.3: Image Inpainting of a women's photograph [2]

With recent advancements in technology, software based inpainting techniques has been extremely popular. Moreover, increased use of digital images and storage nowadays also bolsters the need for better software based inpainting techniques. Inpainting techniques can be broadly classified as shown in Figure 2.4. Traditional techniques have generally either been 'diffusion based' or 'exemplar based'. Diffusion-based techniques generally tend to establish a method of propagation like that of Bertalmio et al., [25], where the propagation is done from outside to inside of the area that need to be inpainted.

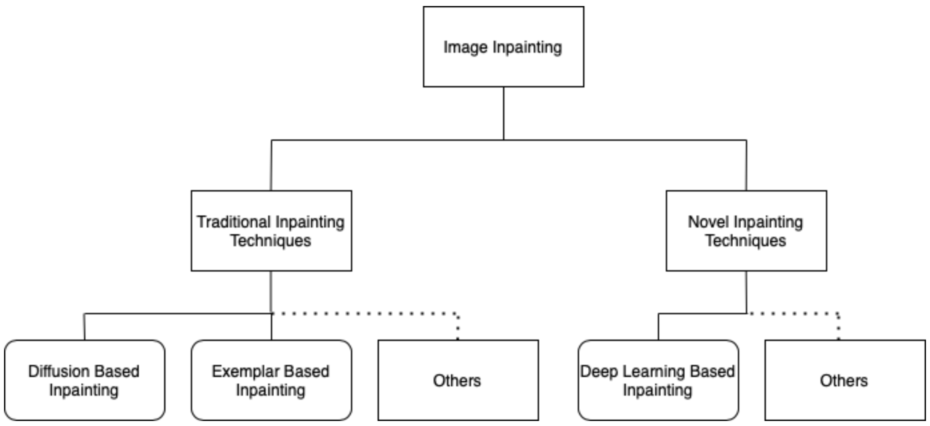
Image Inpainting
Traditional Inpainting Techniques
Novel Inpainting Techniques
Diffusion Based Inpainting
Exemplar Based Inpainting
Others
Deep Learning Based Inpainting
Others

Figure 2.4: Classification of Inpainting Techniques

In case of exemplar-based, the image inpainting is generally either done in a pixel level or patch level [3]. These generally involve in determining a priority for every point along the boundary, and a target pixel p or a small target patch centred at pixel p is determined with highest priority. To inpaint this target pixel p, a source pixel q or a small source patch centred at pixel q is determined [3, 26]. The target patch to be filled is determined using a priority function, which prioritizes the pixel that is to inpainted, assuming it has higher importance or information [3].

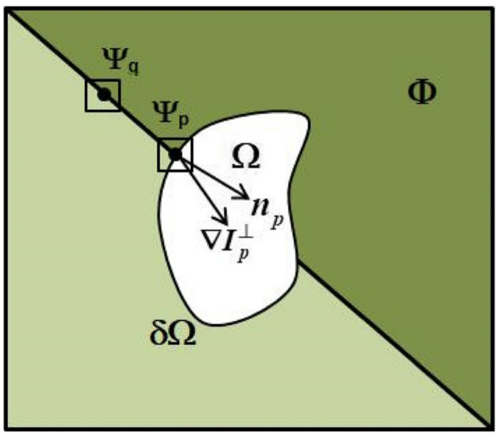

Figure 2.5: Generic exemplar-based inpainting [3]

This can be visualised in Figure 2.5, where Ω and $\delta\Omega$ signifies the missing region and the boundary line of the missing region respectively. Ψ_p and Ψ_q indicate the pixel centered at target patch and source patch respectively, where $p \in \Omega$ and $q \in \Phi$, where Φ includes all the known pixels and is given by the difference between the original Image (I) and the missing region Ω. The figure also shows n_p and $\nabla I_p^{\perp}$, which are the vector orthogonal to $\delta\Omega$ and the isophote vector to pixel p respectively, these are used in the calculation of the priority function $P(p)$ [3].

However, one of the biggest drawbacks with these traditional inpainting techniques is that they do not scale well. When the target area that needs to be inpainted gets too large, the output generated by these techniques are generally incoherent or unrealistic. One of the reasons for such a performance is that these methods sample data within the image to fill the target region. This strips the model the ability to hallucinate novel objects to inpaint target region. As the target region becomes larger, it becomes necessary for the inpainting model to be able to inpaint using samples from a sample space that is similar to the distribution of the original image, rather than just using the data from one image. This requirement combined with advancement in computational and graphical processing capabilities, made deep learning based inpainting techniques of much interest to researchers.

2.2.2 Deep Learning based Inpainting

Single Discriminator Encoder-Decoder Networks

As discussed in section 2.1, GANs have the ability to learn distribution through latent space interpolation, making it an ideal solution for inpainting large missing regions of an image. One of the early implementations of GANs for the purpose of inpainting include "Context Encoders" by Deepak Pathak et al., [4]. Similar to the implementations of autoencoders, the context encoder had two major components, the encoder and the decoder.

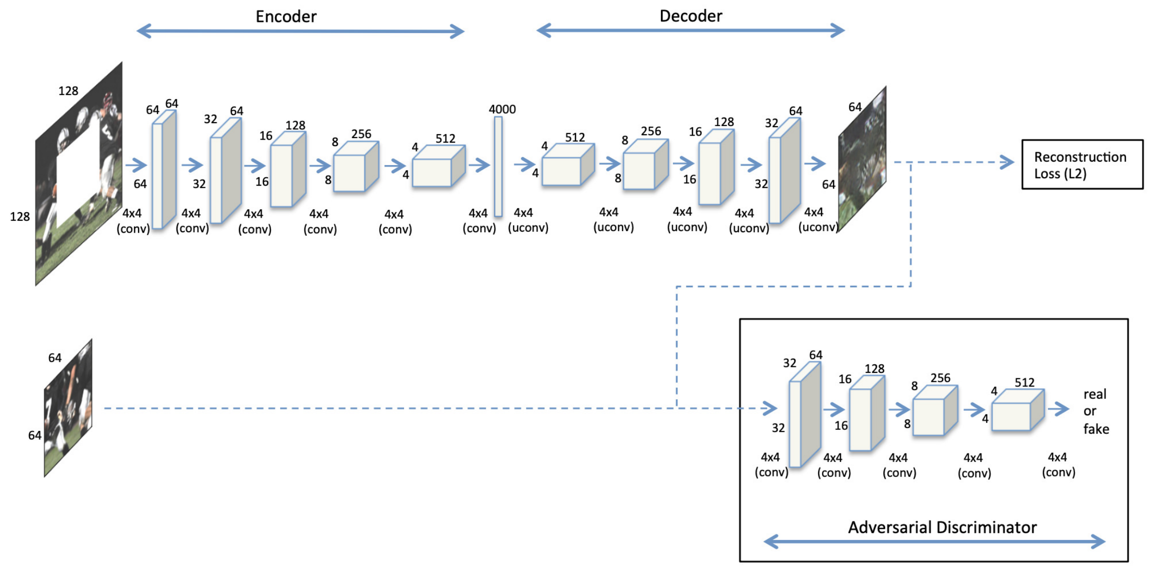

Figure 2.6: Context Encoder Architecture with Joint Loss[4]

Autoencoders are widely known for their ability in dimensionality reduction. However, they do not learn any meaningful information like context or semantics. So the authors [4] have modified their loss function so as to overcome this obstacle.

Type	**Kernel/Size**	**Stride (n)**	**Filters**	**Padding**	**Output Shape**
Input Image	$227 \times 227 \times 3$	-	-	-	-
Convolution	11×11	4	96	-	$55 \times 55 \times 96$
Maxpool	3×3	2	-	-	$27 \times 27 \times 96$
Convolution	5×5	1	256	2	$27 \times 27 \times 256$
Maxpool	3×3	2	-	-	$13 \times 13 \times 256$
Convolution	3×3	1	384	1	$13 \times 13 \times 384$
Convolution	3×3	1	384	1	$13 \times 13 \times 384$
Convolution	3×3	1	256	1	$13 \times 13 \times 256$
Maxpool	3×3	2	-	-	$6 \times 6 \times 256$

Table 2.1: AlexNet Architecture before fully connected layers

They device a joint loss that includes the $L2$ reconstruction loss as well as an adversarial loss from GANs [14]. As seen in Figure 2.6, during training, the encoder takes in the input image with the missing region, where the CNNs are used to generate the latent space of the image. The decoder uses this to inpaint missing content [4].

The encoder also uses the architecture from Alexnet [27], where the image input shape is $227 \times 227 \times 3$. The complete structure used for encoder is as in Table 2.1. However, the encoders are trained with random weights [4] and not the Alexnet's pretrained model which was trained using ImageNet [28] dataset. The output as seen in Table 2.1 is 9216 units, which is a large size compared to any generic autoencoder. As the aim is not dimensionality reduction, the size is allowed to be large. Both encoder and decoder are connected through channel-wise fully connected layer [4]. Stride (n) mentioned in Table 2.1 corresponds to a stride of ($n \times n$). The decoder is made up of five layers of transposed convolutions (also known as Convolution with fractional strides [29]), with rectified linear units (ReLU) as the activation function [4, 30].

The goal of inpainting is not to replicate the ground truth, but to inpaint the missing region that makes the image coherent and contextually meaningful [4]. This means that often there are more than one way to inpaint. The authors accounted this in loss function with two separate losses. The $L2$ reconstruction loss to handle the general structure and coherence of the inpainted output. However, reconstruction loss generally tends to average different potential outcomes in a distribution, also known as modes [4]. This often results in a washed out and blurry output [4]. To overcome this, the adversarial loss [14] is used in conjunction.

The $L2$ reconstruction loss (L_{rec}) used in context encoder [4] is as given in equation 2.8.

$$L_{rec}(x) = ||m \odot (x - C[(1-m) \odot x])||_2^2. \tag{2.8}$$

Here, m denotes the binary mask, where the region to be inpainted have the value 1 and other regions (known pixels) have the value 0 [4]. Also, x denotes the ground truth, and the function $C(k)$ denotes the encoder's output for input k. Moreover, $\odot$ represents pixel-wise product.

The adversarial loss (L_{adv}) used in context encoder is a modified loss function of equation 2.1. As discussed in section 2.1.2, training GANs can be unstable. This is more prominent in the case, when the input image has a patch in between, making it extremely easier for the discriminator to classify between real and generated outputs. This would lead to fast training of discriminator and the generator will end up with less gradients at an early stage, giving no meaningful outputs. To tackle this, the authors [4] use the context, only with generator, and only the patches for discriminator. The adversarial loss (L_{adv}) then is as given in equation 2.9 [4], where the function $D(k)$ denotes the decoder's output for input k. The joint loss is given by equation 2.10, where $\lambda_{rec} = 0.999$ and $\lambda_{adv} = 0.001$ [4].

$$L_{adv} = \max_{D} \mathbb{E}_{x \sim p_{data}(x)}[\log(D(x)) + \log(1 - D(C((1-m) \odot x)))]. \tag{2.9}$$

$$\mathcal{L} = \lambda_{rec} L_{rec} + \lambda_{adv} L_{adv}. \tag{2.10}$$

The study also compares three different masking techniques while training. One being a square patch, always in central region of the image, followed by random definite blocks within the image and final one being random patches without definite shapes. During all these, it was maintained such that the removed patches were up to $\frac{1}{4}^{\text{th}}$ of the input image. It was found that the latter two methods significantly outperformed the latter [4].

Multi-Discriminator Completion Networks

One of the studies that builds upon the context encoder as discussed earlier is "Globally and Locally Consistent Image Completion" by Satoshi Iizuka et al., [5]. Similar to context encoders [4], the Globally and Locally Consistent Image Completion (GLC) network has two major components, a completion network and the auxiliary network. The auxiliary network in GLC has two discriminators, the global discriminator and the local discriminator. The completion network as seen figure 2.7, is made up convolutional networks in an encoder-decoder structure. The global and local discriminators check for image consistency for the whole image (globally) and just the inpainted patch (locally) respectively [5].

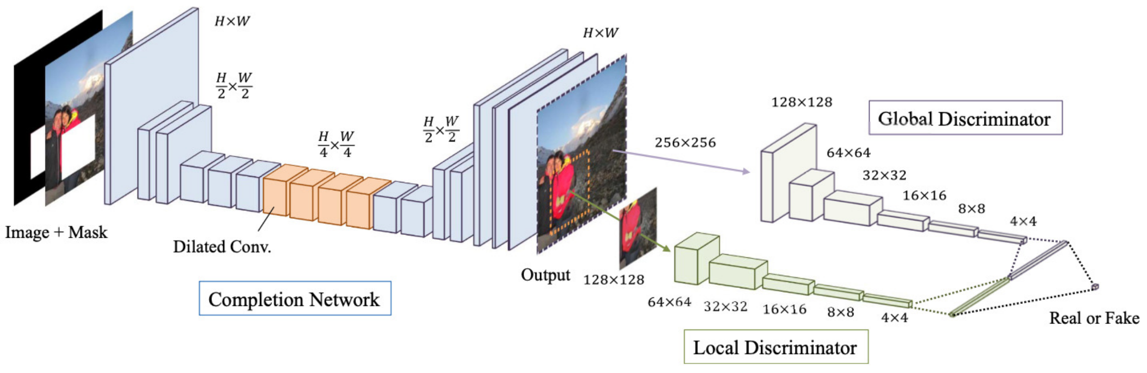

Figure 2.7: GLC Architecture [5]

Unlike Context encoder [4], which could only work on 128×128 image, the authors designed GLC to work with any random resolutions [5]. The completion network is as given in Table 2.2 [5]. The input includes a 3 channel image with the binary mask m, where the region to be inpainted have the value 1 and other regions (known pixels) have the value 0. The completion network outputs an entire image, unlike context encoder [4], where the output by the decoder was only the patch that needed to be inpainted. Using the mask m, only the region to be inpainted is extracted and used with the input image.

The authors also use dilated convolution [31], with dilation factor η as given in Table 2.2, so as to increase the receptive field of the convolution layer without any strain on computational resources.

Type	**Kernel**	**Dilation** (η)	**Stride** (n)	**Output**
Convolution	5×5	-	1	64
Convolution	3×3	-	2	128
Convolution	3×3	-	1	128
Convolution	3×3	-	2	256
Convolution	3×3	-	1	256
Convolution	3×3	-	1	256
Convolution	3×3	2	1	256
Convolution	3×3	4	1	256
Convolution	3×3	8	1	256
Convolution	3×3	16	1	256
Convolution	3×3	-	1	256
Convolution	3×3	-	1	256
Up-convolution	4×4	-	$\frac{1}{2}$	128
Convolution	3×3	-	1	128
Up-convolution	4×4	-	$\frac{1}{2}$	64
Convolution	3×3	-	1	32
Output	3×3	-	1	3

Table 2.2: Completion Network of GLC Network

In Table 2.2, where there is no dilation factor given, $\eta = 1$. When $\eta = 1$, the dilated convolution operation is the same as normal convolution. Stride (n) mentioned in Table 2.2 corresponds to a stride of ($n \times n$). The global and local discriminators updates first before the completion network during training. They have six and five convolution layers respectively, where each layer is a (5×5) and (2×2) strided convolution. Both provide a 1024 units concatenating to a fully connected layer with a total of 2048 units.

To give priority to the stability of the GAN, similar to context encoder [4] in equation 2.8, the GLC network [5] also use weighted mean squared error loss in conjunction with the adversarial loss [14]. While the former focuses on stability, the latter tries to improve the coherence and quality of output. The MSE loss function L_{MSE}, is as given in equation 2.11 [5].

$$L_{MSE} = ||m_i \odot (C(x, m_i) - x)||^2, \qquad (2.11)$$

where $C(x, m_i)$ represents the output of the completion network, for input image with patch x, and input mask m_i.

For the adversarial loss, the GLC does include the context to the global discriminator unlike the context encoder, as seen in equation 2.9. The adversarial optimization is as given in equation 2.12 [5].

$$\min_{C}\max_{D}\mathbb{E}[\log D(x,m_r)+\log(1-D(C(x,m_i),m_i))], \tag{2.12}$$

where $D(x, m_r)$ represents the output of the combined discriminators, for input image with patch x, and random mask m_i. The final optimization combining the MSE loss and adversarial optimization then becomes [5],

$$\min_{C}\max_{D}\mathbb{E}[L_{MSE}+\alpha\log D(x,m_r)+\alpha\log(1-D(C(x,m_i),m_i))], \tag{2.13}$$

where the hyper-parameter $\alpha = 0.004$. To overcome instability issues that occur in achieving the nash equilibrium, the authors implement a three stage learning process, where initially the completion network is trained with loss from equation 2.11 for T_C iterations followed by, discriminators with optimization from equation 2.12 for T_D iterations [5]. At the end, both are trained together. The input image is resized to 256×256 resolution for training. Also, the patches and masks are randomly generated, where the height and width of the patch range from $[96, 128]$ pixels, while training. To improve coherency and output quality, the GLC also implements a post processing technique to the final output of the completion network, using 'fast marching method' along with 'Poisson image blending' [5].

Like most of the inpainting state of the art, the GLC was trained on *places2* dataset [32] for scene completion with around 8,097,967 training images [5] out of 10,624,928 available from the dataset. GLC also uses *CelebFaces Attributes (CelebA)* [32] and *CMP Facade* [33] datasets, for face and facade training. It uses $200,000$ and 550 out of $202,599$ and 606 available images from the datasets respectively. The latter two datasets, are used by resuming training on a completion network earlier trained on *places2* dataset, with $T_C = 90,000$, $T_D = 10,000$ and $T_{total} = 500,000$ iterations [5]. One of the limitations of this study was in training time. According to the authors,

> *"The entire training procedure takes roughly 2 months on a single machine equipped with four K80 GPUs."* [5].

Contextual Attention based Networks

One of the studies that uses the Globally and Locally Consistent Image Completion network as a baseline and significantly improves on training time is "Generative Image Inpainting with Contextual Attention" by Jiahui Yu et al,. [6]. Along with GANs, the network also includes contextual attention to use features from distant location [6].

The network is designed as two stages, a course network and a refinement network [6]. Similar to earlier studies, the contextual attention based model also involves two losses, a modified version of reconstruction loss known as *'spatially discounted reconstruction loss'* along with adversarial loss using global and local discriminators as in the GLC network [5, 6]. The network architecture is as seen in Figure 2.8. The course network is given inputs similar to that in the GLC architecture [5], an input image with its binary mask (m), trained with just the reconstruction loss to produce a course output [6]. This output is then fed into the refinement network which uses both the reconstruction loss along with adversarial loss. Both the stages use dilated convolution [31] to increase the receptive field without significant impact on computational resource requirements [6].

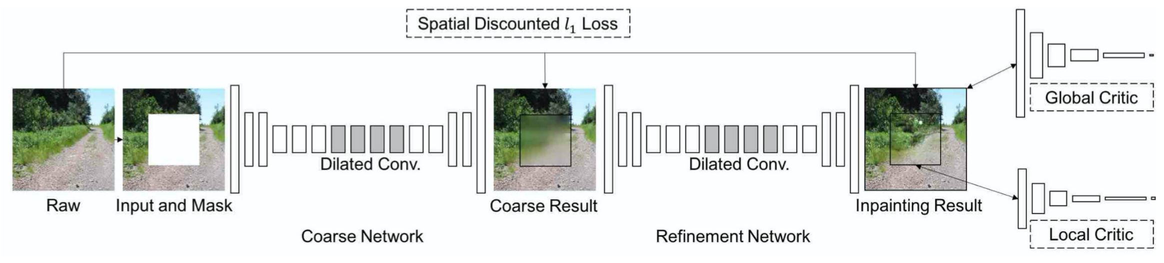

Figure 2.8: Generative Inpainting Network Architecture [6]

Moreover, unlike context encoders [4] or the GLC [5] which use deep-convolutional GANs (DCGANs), this Contextual Attention based network uses Wasserstein GAN with Gradient Penalty (WGAN-GP). As discussed in section 2.1.3, WGAN-GP is more stable and robust in reaching the required latent space distribution than DCGAN. Since the network only needs gradients for the region that needs to be inpainted, the authors modify the gradient penalty from equation 2.7, as in equation 2.14 [6].

$$\lambda\, \mathbb{E}_{\hat{x}\sim P_{\hat{x}}}(||\nabla_{\hat{x}} D(\hat{x}) \odot (1-m)||_2 - 1)^2. \tag{2.14}$$

The spatially reconstructed loss (l_1) is calculated for each pixel is as given in equation 2.15. A weighted sum of these losses are calculated.

$$l_1 = \gamma^l \tag{2.15}$$

Here, l is the distance between the unknown pixel ($p_u \in I_r$) in the region that needs to be inpainted (I_r) and the nearest known pixel (p_k). Also, γ is a constant whose value is 0.99. The notion behind such as loss function is that, in the region (I_r), the unknown pixels (p_u) near the boundary are closer to pixels with known values (p_k). Therefore, these unknown pixels (p_u), would be more similar in distribution to that of its nearby pixels with known values (p_k). This makes equation 2.15 to produce a higher loss to drive p_u and p_k to be similar. However, any unknown pixel (p_u) in a region that is closer to the centre of the region (I_r), need not necessarily be the same as its nearest known pixel (p_k). If the region (I_r) is large enough, a novel hallucinated object may make the inpainting more coherent and meaningful, than being similar to its nearest known pixel. In this case, equation 2.15 produces a lower loss.

The authors also suggest that the use of spatially discounted reconstruction loss with WGAN has significantly improved the training speed to a week [6] instead of the two months as mentioned in [5].

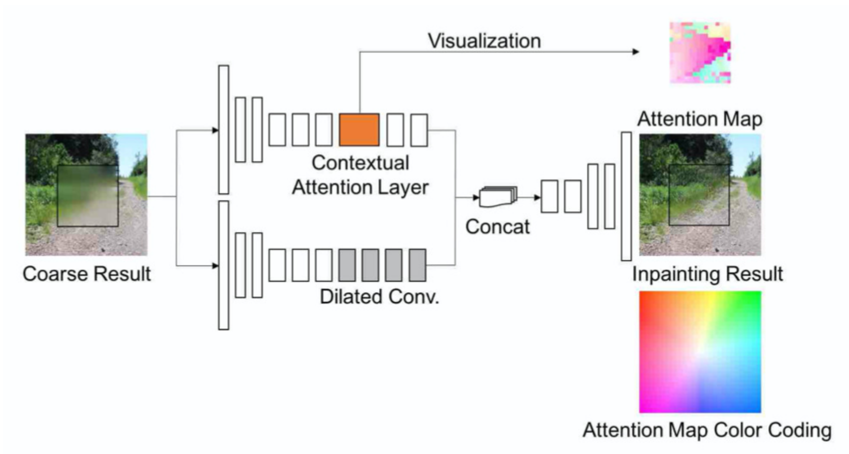

Figure 2.9: Refinement Network [6]

The refinement network is made up of two parallel encoders as seen in Figure 2.9. The layer at the top is contextual attention layer and the layer below is the dilated convolution layer [6]. The contextual attention layer helps learning the background feature (or known pixels) that may be required to inpaint the region I_r [6].

The attention map colour coding is just a visualisation tool used by the authors, that follow an optical flow colour coding model [34]. The colour of attention visualisation represent the localisation of the attention of the pixel corresponding to the location of that colour in the attention map color coding [6]. Moreover while the contextual attention layer focuses on background features, the parallel dilated convolution layer is used to hallucinate novel objects [6]. The output from both these parallel encoders are concatenated and given as input to the decoder.

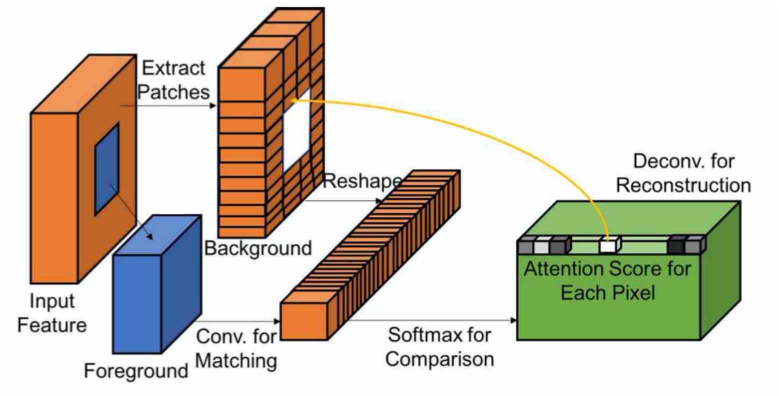

Figure 2.10: Contextual Attention Layer [6]

Figure 2.10 shows the contextual attention layer. The authors follow a *'match and attend'* [6] approach. To match unknown pixels p_u, (foreground) to pixels in known pixels (background), the authors propose [6] to reshape the background patches into 3×3 convolutional filters $\{b_{x',y'}\}$ and use cosine similarity with the foreground patch $\{f_{x,y}\}$, as in equation 2.16 [6].

$$c_{x,y,x',y'} = [\frac{f_{x,y}}{||f_{x,y}||}, \frac{b_{x',y'}}{||b_{x',y'}||}], \tag{2.16}$$

where $c_{x,y,x',y'}$ is the cosine similarity between the foreground and background patch centred in (x,y) and (x',y') respectively [6].

We then get the attention score $\widetilde{c}_{x,y,x',y'}$ after applying softmax function, as seen in equation 2.17 [6].

$$\widetilde{c}_{x,y,x',y'} = sf_{x',y'}(\lambda c_{x,y,x',y'}), \tag{2.17}$$

where the function *sf(k)* is softmax function for input k and λ is a constant.

Once the attention scores are obtained, the same convolutional filters, $\{b_{x',y'}\}$ are used in inpainting by up-convolution. As an attempt to improve coherency, the authors also propose *'attention propagation'* implementation [6]. It involves in propagation of attention scores as seen in equations 2.18 and 2.19 for left-right and top-down propagation respectively [6], implemented in that order.

$$\hat{c}_{x,y,x',y'} = \sum_{i\in\{-k,\ldots,k\}} \widetilde{c}_{x+i,y,x'+i,y'}. \tag{2.18}$$

$$\hat{c}_{x,y,x',y'} = \sum_{j\in\{-k,\ldots,k\}} \widetilde{c}_{x,y+j,x',y'+j}. \tag{2.19}$$

Here, k represents the kernel size. The model was trained on 256×256 image with largest hole size as 128×128, which is $\frac{1}{4}^{\text{th}}$ of the image, on the datasets *CelebA*, *DTD textures* and *ImageNet* [28, 35, 36].

The network architectures are as given in Appendix A.1.

High Resolution Image inpainting

Recently, image consumption across industries involve very high-resolution content. However, the earlier discussed methods do not scale well, as they would require very high computing power therefore high cost associated with it. A recent study that focuses on high-resolution image inpainting is "Contextual Residual Aggregation for Ultra High-Resolution Image Inpainting" by Zili Yi et al,. [7]. The authors state that the model would be able to inpaint image with resolution as high as 7680×4320 (8K) [7]. To tackle quality issues with large holes and to make the training feasible the authors propose contextual residual aggregation (CRA) mechanism [7]. The network architecture is as given in Figure 2.11.

As seen in seen in Figure 2.11, the network down-samples the input image to a lower resolution before feeding it to the generator. Later, this lower resolution image is up-sampled to the original input size and then compared to the original image to obtain contextual residual (high frequency components) for the region with known pixels [7]. The inpainted output of the generator is of the lower resolution, which is also up-sampled to the resolution of the original image. The attention scores are then used to obtain high frequency components for the region that need to be inpainted. The high frequency components are then added to the inpainted output to obtain a high-resolution image. Similar to [6], this study also proposes a two stage network, course and refinement network for the generator.

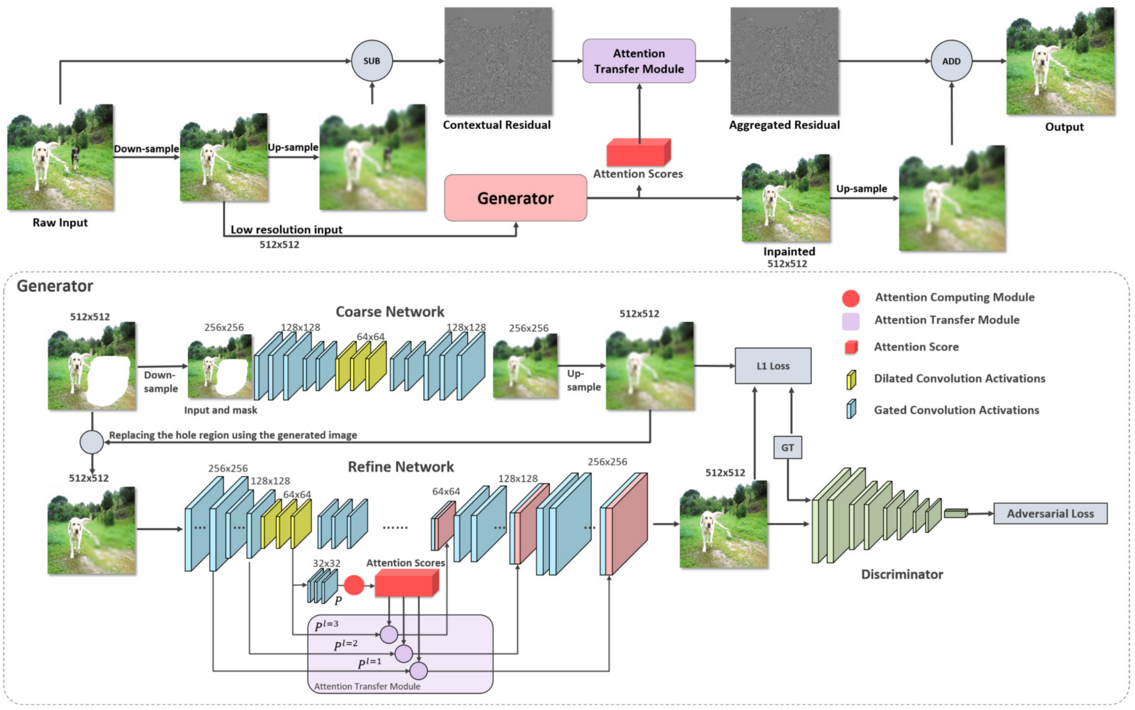

Figure 2.11: High Resolution Inpainting Architecture [7]

To account for irregular hole-filling, the authors propose the use of modified gated convolution known as Light weight gated convolution (LWGC), given by equation 2.20 [7],

$$O = \sigma(conv(W_g, I)) \odot \psi(conv(W_f, I)), \tag{2.20}$$

where σ and ψ are given by sigmoid and ELU activations respectively, W_g and W_f are given by convolutional filters used for gates and features calculations respectively. For the generator, the down-sampled (512×512) input image is further down-sampled to (256×256) to make the course network lighter. Both the stages use dilated convolution along with modified gated convolutions as discussed above [7].

This model also calculates attention scores similar to [6] as in equation 2.16 and equation 2.17. However, in addition to the attention scores, the authors also include an attention transfer module (ATM) [7].

To better understand attention transfer module, let us assume the region to be inpainted as re_{inp} and region outside re_{inp} as re_{out}. As seen in equation 2.21, the j^{th} patch to be filled in re_{inp}, (p_j^l) is given by i^{th} patch (p_i^l) taken from re_{out} from its corresponding layer l weighted by attention score $\widetilde{c}_{i,j}$ [7].

$$p_j^l = \sum_{i=1}^{N} \widetilde{c}_{i,j} p_i^l, \tag{2.21}$$

where $l \in \{1, 2, 3\}$ and N is the number patches in re_{out} of size (3×3) [7]. This can also be seen in Figure 2.11 in the generator's architecture. Since the attention scores are obtained from 32×32 patches from re_{out} and p_i^l is coming from different levels, the size of the patch p_i^l, must vary accordingly for complete coverage [7]. For $l = 1, 2, 3$, the size of the patch p_i^l must be greater than or equal to (8×8), (4×4) or (2×2) respectively.

The process of obtaining high frequency components for re_{inp} is similar to obtaining contextual image data from equation 2.21. It is given by equation 2.22.

$$R_j = \sum_{i=1}^{N} \widetilde{c}_{i,j} R_i, \tag{2.22}$$

where R_i is the i^{th} patch in re_{out} from Contextual Residual image obtained by subtracting input image with down and up-sampled image as seen in Figure 2.11, and R_j is the j^{th} patch filled in re_{inp}.

Similar to [6], the study also uses WGAN-GP adversarial loss along with reconstruction loss. The model's adversarial loss objective that the generator and discriminator aim to minimise and maximise respectively is the same as equation 2.6, the gradient penalty is given by equation 2.7.

The $L1$ reconstruction loss is lighter than the loss used in [6]. It is given by equation 2.23 [7].

$$L_{rec} = \alpha_1 L_{in-hole} + \alpha_2 L_{context}, \tag{2.23}$$

where $\alpha_1 = 1, \alpha_2 = 1.2$ are coefficients and $L_{in-hole}$ and $L_{context}$ are given by equation 2.24 and 2.25 respectively [7].

$$L_{in-hole} = |G(x, m)| \odot m, \tag{2.24}$$

$$L_{context} = |G(x, m)| \odot (1 - m), \tag{2.25}$$

where $G(x, m)$ denotes the generator's output for input image x and mask m. The combined loss is given by equation 2.26 where $\lambda = 10^{-4}$ is the coefficient for adversarial loss (L_{adv}) [7].

$$L = L_{rec} + \lambda L_{adv} \tag{2.26}$$

The study also diversified masking techniques by both simulating random masks, scratches amongst others [7]. The training was done on datasets like *Places2* [32], *CelebA-HQ* [37] and *DIV2K* [38, 39].

2.3 Omnidirectional Image Representation

There are various representations of 360° images (also known as omni-directional image (ODI) or spherical panoramas). It is vital to understand different representations of 360° images so as to take factors like distortion into account. All of the earlier studies discussed for inpainting were designed to work on traditional 2D image. This often means that the networks are optimized for the same. Therefore, to implement similar methodologies for 360° images, we need to either change the network so as to work with 360° images or work with different representations to be able to extend existing techniques as a solution to our inpainting problem for 360° images.

A camera's field of view denotes the angle upto which the camera is able to capture the image in both vertical and horizontal directions. On a typical smartphone camera[1], the images are taken with a field of view (also known as angle of view) of around 60° horizontal [41]. In comparison, omni-directional images have a field of view of 360° horizontal. Having a spherical field of view like that of $360° \times 180°$ and projecting them onto a 2D plane would have loss of information in the form of distortions. Some of the frequently used projections of 360° image or maps thereof include [8, 10, 42, 43],

- Equirectangular Projection
- Cubemap Projection
- Mercator Projection
- Lambert Projection

Other less frequently used projections include Truncated Square Pyramid Projection (TSP), Craster Parabolic Projection (CPP) [42]. Maps are similar to 360° images in the sense that a spherical earth map (3D-plane) is projected onto a 2D map, involving similar distortions challenges. Since issues with creation of map are not new, map projections serve useful to understand spherical distortions. General map projections can be classified into [11, 10],

1. Equal Area Projections
2. Conformal Projections
3. Equidistant Projections

Each of these projections aim to preserve a quality. Equal area projections like that of Lambert Equal Area Projection, preserve the relative area on the map [10]. Conformal Projections like that of Mercator Projection maintain the relative shape in the map [10]. Equidistant projections like that of Equirectangular projection, as the name suggests, preserve relative distance in the map with respect to locations on 3D plane. In this study, we shall see Equirectangular, Cube Map and Tangent Image representations more closely.

[1]iPhone 6 [40] taken for reference.

The study, also suggests the category of conventional projections, which are application based designs and do not come under either of the above categories. Examples include gnomonic and miller projections amongst others [8]. Cube map and tangent images are an example of conventional projection as they are obtained via gnomonic projection [10].

2.3.1 Tissot's Indicatrix

Introduced in 1930 [44] by Nicolas Auguste Tissot, Tissot's Indicatrix is used to visualise the spherical distortion. The indicatrix overlays ellipses at different locations of the map. The shape and area of the ellipse changes with respect to the level of distortion and is generally compared with an ellipse (generally a circle) from a non-distorted location (generally the centre) of the map. Figures 2.12 and 2.13 represent the tissot's indicatrix overlay on Mercator and Lambert equal-area projections respectively.

In Figure 2.12, which is a conformal projection, we can see how all the ellipses are circles, signifying preservation of relative shape, however, we see that the area of ellipses are different meaning this projection does not preserve area. However, in Figure 2.13, we see the opposite. Being an equal-area projection, we see while the shapes of ellipses are different, the area of all ellipses are equal.

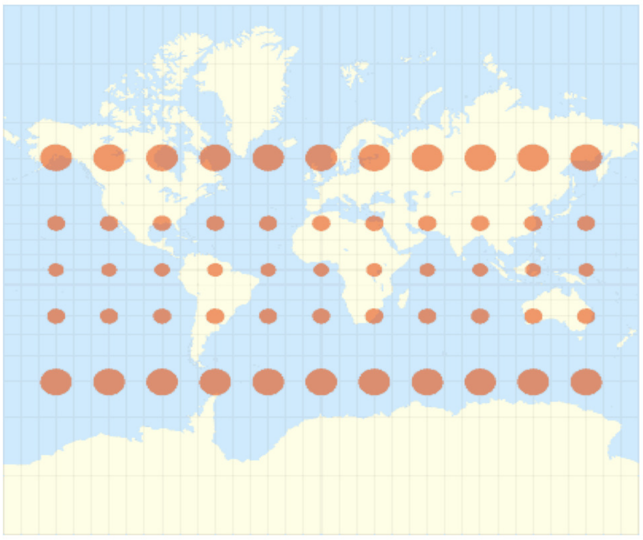

Figure 2.12: Tissot's Indicatrix on Mercator Projection Map [8]

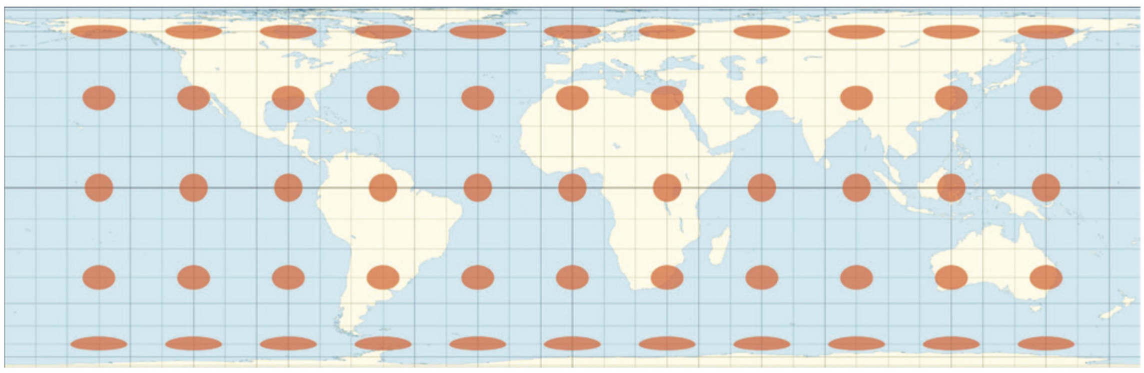

Figure 2.13: Tissot's Indicatrix on Lambert-equal area Projection Map [8]

2.3.2 Equirectangular Image

Equirectangular projections are one of the most commonly used projection in 360° computer vision problems. The rectangular image is obtained via cylindrical projection, where a open-ended cylinder enclosing the sphere is unfolded into rectangle after the projection, as seen in Figure 2.14.

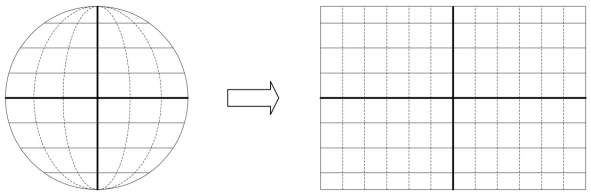

Figure 2.14: Equirectangular Projection [9]

2.3.3 Cube Map Representation

Cube map representations are obtained through rectilinear projection, where each face of the cube is a projection made by a plane tangential to the respective position of the sphere as seen in Figure 2.15. This way we end up with six tangential projections corresponding to six faces of the cube. Each cube face can be categorised as front, back, up, down, left and right. While the distortion is relatively less within each face of the cube, one of the drawbacks is when trying to use the projection completely for image processing or computer vision problems. Especially, when trying to conform it a rectangular 2D image, we end up with lot of empty space as seen in Figure 2.16, which most of the time prove detrimental to the task at hand.

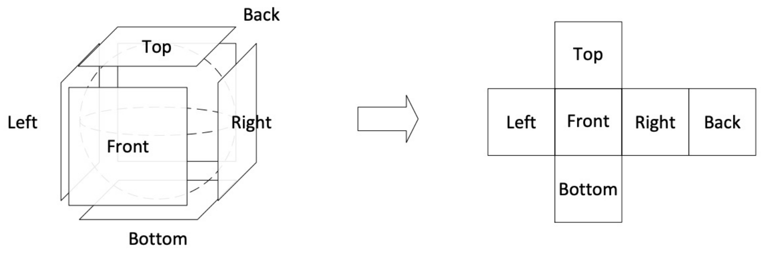

Figure 2.15: Cube-Map Projection [9]

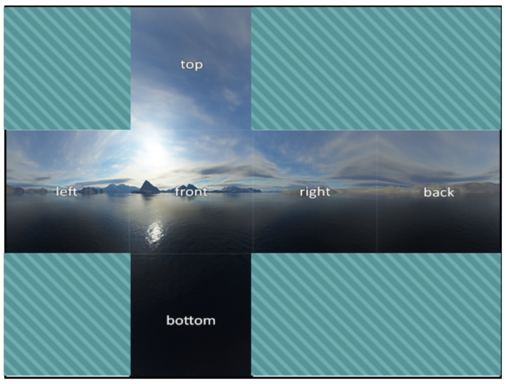

Figure 2.16: Cube-Map Projection visualised in rectangular image

2.3.4 Tangent Images

One of the studies that emphasised on the application of computer vision techniques that are used in regular 2D images to 360° images with minimal changes required in the techniques, was done by Marc Eder et al., [11]. This study proposed a new projection technique known as 'Tangent Images', that involved minimal distortion while allowing the retention of existing computer vision techniques without much modifications.

The projection of the spherical image is done on a twenty face polyhedron known as icosahedron [11]. Icosahedron is a polyhedron with the most number of faces, with each of its face having equal area and equal number of faces at every vertex. As seen in Figure 2.17, each face can further be divided into four equal faces. This when done infinitely, we get a sphere. The equation for the number of faces is as given in 2.27 [10].

$$F = 20 \times (4^k), \tag{2.27}$$

where F denotes the number faces and k denotes the number of division on a face [11].

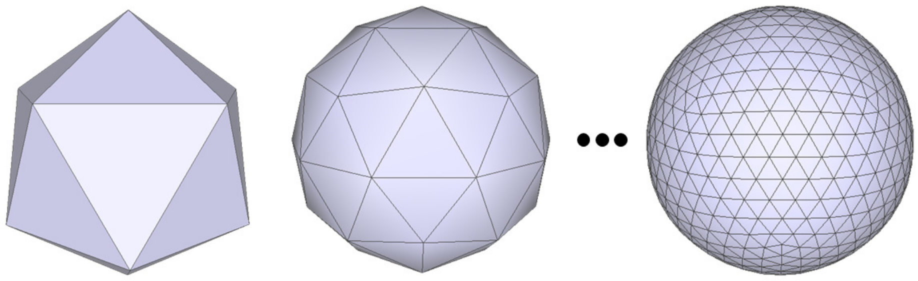

Figure 2.17: Face division in Icosahedron [10]

According to the authors, tangent images are defined as,

> *"Tangent images are the gnomonic projection of the spherical data onto oriented, square planes centered at each face"* [11].

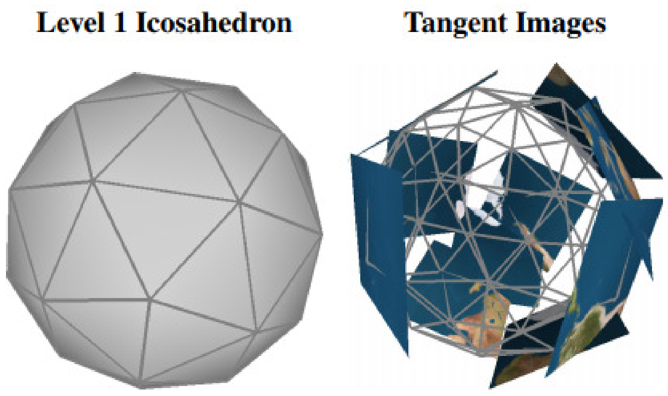

Figure 2.18: Tangent Image Projection [11]

As seen in Figure 2.18, each of the face has an image which is tangential to that icosahedral face [10]. The dimension d, of such overlapping tangential images are given by equation 2.28 [11].

$$d = 2^{s-b}, \tag{2.28}$$

where s denotes the level of icosahedron. For example, taking a 256×512 spherical image which is approximately equal to a level 7 icosahedron (327,680 faces) ($s = 7$), when taken without any further subdivision ($b = 0$), yields 20 tangent images of the resolution 128×128. However, in the same case if subdivisions were set to one, ($b = 1$), it will yield us 80 tangent images of the resolution 64×64.

Chapter 3

Inpainting for Omnidirectional Images

3.1 Overview

Our initial approach to inpaint 360° images was to utilise existing networks and solve distortion issues outside the network. Therefore, as discussed in section 2.3, we used projections like cube-map, tangent and equirectangular to record performance on existing state of the arts. We have used 'Generative Image Inpainting with Contextual Attention' by Jiahui Yu et al., as discussed in section 2.2.2 as our baseline [6]. Our modified network architecture is as shown in Figure 3.1. The basic structure of the two-stage GAN network follows that of our baseline model [6], therefore the loss function follows that of equation 2.6 for GAN losses and equation 2.14 for gradient penalty.

3.2 Training Environment

The training for different models generated were performed on cloud with Google Colab Pro [45]. Google Colab is a free jupyter notebook based environment on frontend and linux on the backend. It is a product of Google to encourage Machine Learning projects and research on its platform. However, being free, there are lot of constraints on the product, such as 'idle timeouts' of the notebooks, '12 hours run-time limit', with GPUs of Nvidia K80 and lower RAM. Due to the requirements of our study, we went for the paid version, Google Colab Pro, which gave access to higher number of concurrent sessions, with run-time limits increased to 24 hours, GPUs upto Nvidia Tesla P100 and higher RAM instances.

Our project used instances with Tesla P100 which has 16GB of HBM2 VRAM and compute capability of 6.0, 24GB RAM, and Intel Xeon 2.3GHz processor. However, it is to be noted that Google allocates instances shared between various users, this meant that our instances, for example, were allocated with less than 24GB RAM most of the time.

3.3 Training Details

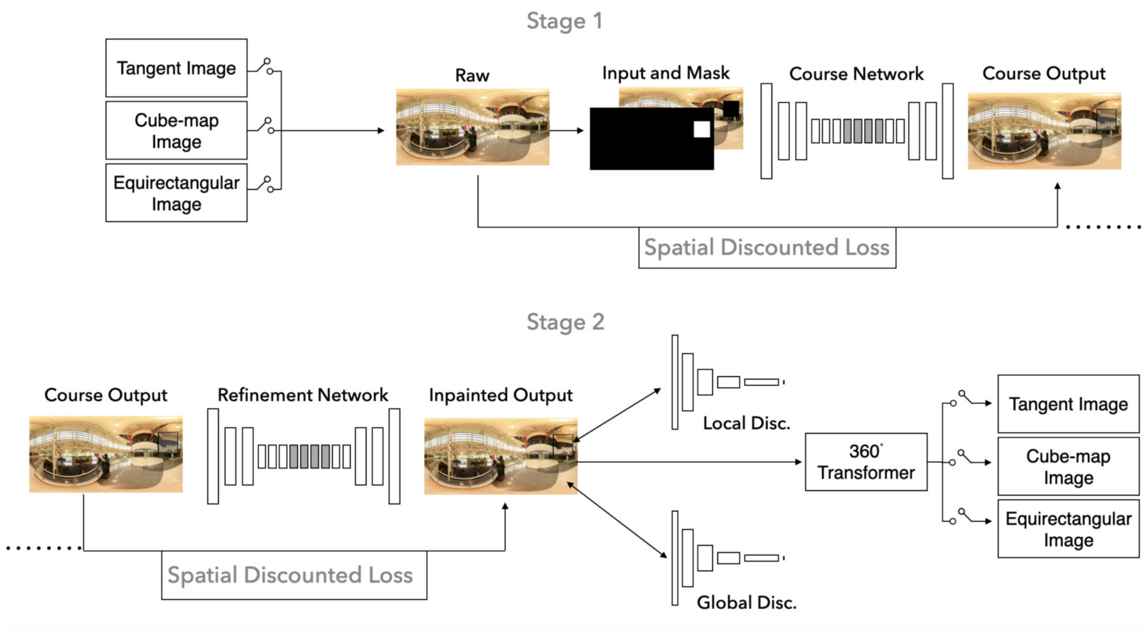

Figure 3.1: Proposed Architecture

The network generates a random patch and its respective mask during training using the input image (*Raw* as seen in Figure 3.1). The network architecture is as given in Appendix A.1. For equirectangular projection, we had trained three different models using the same architecture as in Figure 3.1 to conduct various experiments as listed below.

1. Data Augmentation with just Lateral Shifting (*S1*)
2. Data Augmentation with Rotation and Flipping (*S2*)
3. Patch Resolution Comparison

For *S1* and *S2* the training images were of the resolution 384×192 with random patches of the size 50×50. The image resolution was selected based on hardware capabilities. For patch resolution comparison, the model was trained on 384×192 images with random patches of the size 32×32. Due to the limitations in hardware mentioned in section 3.2, specifically due to run-time limits, we had limited each our training run to a maximum iterations of 50,000 with each epoch having 400 iterations amounting to 125 epochs. After each such run, the model was saved and the instance was reset to continue training. The training was totally done for 1000 epochs for *S1*, and 250 epochs for *S2* and patch resolution comparison with ADAM optimizer, where $\beta_1 = 0.5$, $\beta_2 = 0.9$ and learning rate as 10^{-4}, with weights restored from the baseline model [6]. For all training instances, the batch size was kept to 6. $\lambda = 10$ for the gradient penalty in WGAN loss. The $\alpha_{L1} = 1.2$ and $\alpha_{WGAN} = 5 \times 10^{-4}$ for L1 and WGAN losses respectively. $\alpha_{GlobalWGAN} = 1$, for WGAN loss, making both Global and local WGAN losses weighted equally.

3.4 Dataset

In this study, we use the SUN360 [46] dataset, for training. We used 3003 images from the dataset, each with a resolution of 1920×960. Out of these 3003 images we removed 12 images for final testing, giving us 2991 images to work with. An additional 34 images were taken from "HDRI Haven" collection, a CC0 image set for testing [47].

For data augmentation with just lateral shifting (*S1*), we augmented 5 images (inclusive of the input image) from one input image. Out of 960 pixels wide, we shifted the last set of 192 pixels ($\frac{1}{5} \times 960$), a part of image of size, 1920×192, and is appended in front of the beginning of the image as shown in Figure 3.2. This expands the dataset from 2991 to 14,995 images. These 14,995 images were split into 12,715 ($\approx 84.7\%$) and 2,240 ($\approx 15.3\%$) images for training and validation set respectively. Both experiments *S1* and Patch Resolution Comparison use this dataset for training.

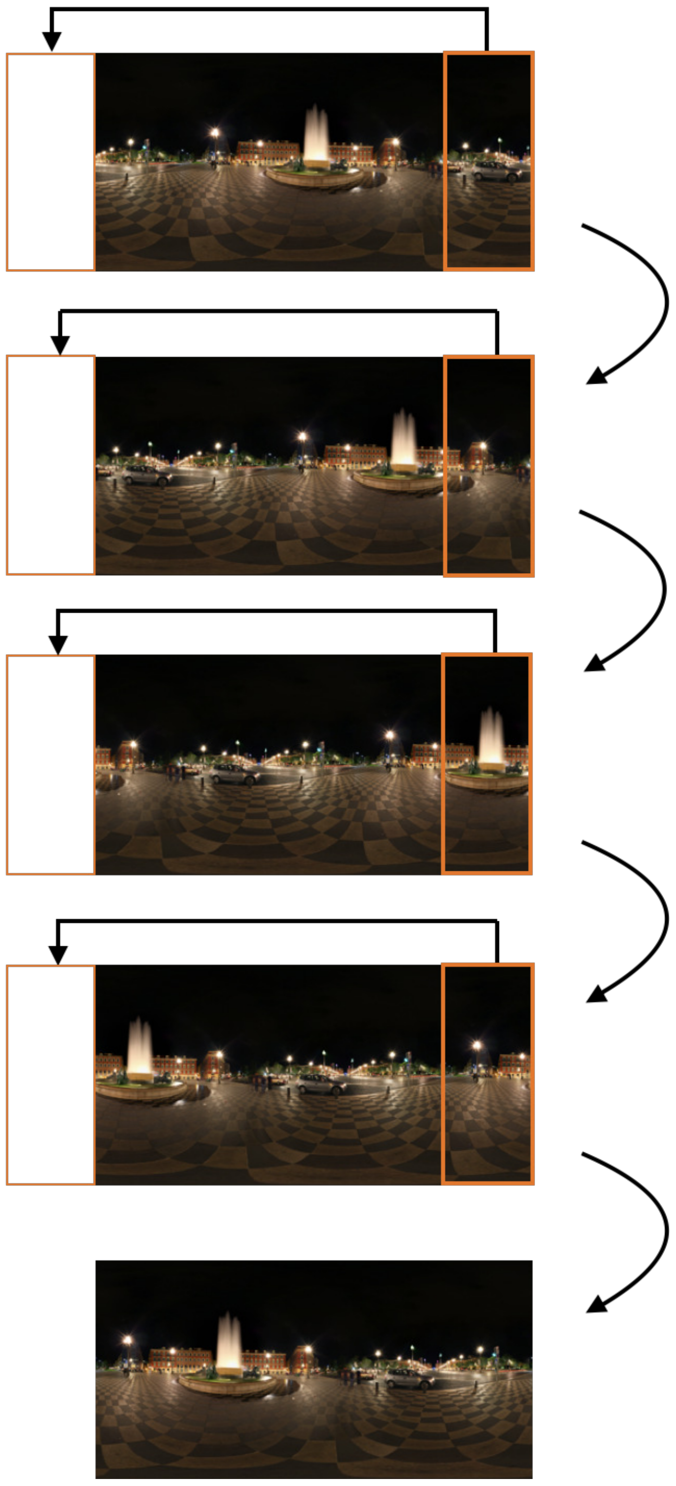

Figure 3.2: Lateral Shifting- Data Augmentation

For data augmentation with rotation and flipping (*S2*), we augmented 4 images (inclusive of the input image) from one input image. Each input image was flipped (mirror image) and rotated (180°). Then the rotated image was also flipped. This gives all possible orientation of the image. This is visualised in Figure 3.3. This expands the dataset from 2991 to 11,964 images, for *S2* experiment. These 11,964 images were split into 10,112 ($\approx 84.5\%$) and 1,852 ($\approx 15.5\%$) images for training and validation set respectively. For both augmentation methods, the images are downsized according to the experiments as mentioned in section 3.3.

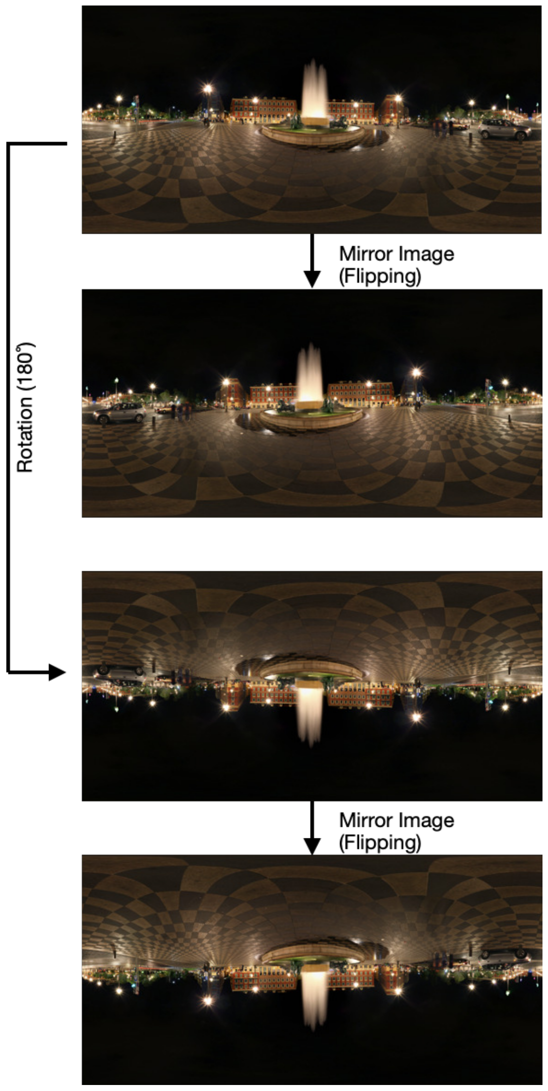

Figure 3.3: Rotation and Flipping- Data Augmentation

Extending experiment *S1*, we used our test set images and created three versions of it, all with same patch size 50×50. The three versions varied in image resolution as categorised as large, medium and small, being 768×384, 576×288 and 384×192. This was done to see the effects of change in available image content for the network to inpaint from.

For the test set images we create patches at random location of various sizes according to the experiment as mentioned before. The input image with patch and their respective mask that indicates the region that needs to be inpainted in white, are as shown in Figure 3.4.

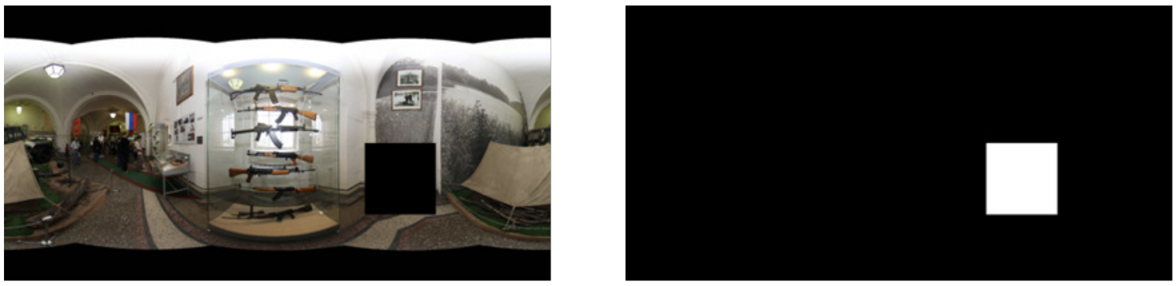

Figure 3.4: Input Image with Patch and Mask

3.5 Evaluation

3.5.1 Cube-Map and Tangent Images

We do several experiments to evaluate the performance of our model against other state of the art models, and evaluate the performance of different image projection techniques. For cubemap projection we did two experiments, one where the cube map image is taken as a whole rectangular image with the extra unwanted region (R_u) (as mentioned in section 2.3.3) filled in as black. This is seen in Figure 3.5 where the input image with patch and its mask from Figure 3.4 are converted into the cube map equivalent (Yellow Border with text just shown for clarity). We expected this model to perform poorly, due to R_u filled with black colour making the model inpaint patches near the boundary with black as well. The result shown in Figure 3.6 is the inpainted output, where red and green ellipses mark the respective patch splits in different faces. We are able to see that the output is incoherent.

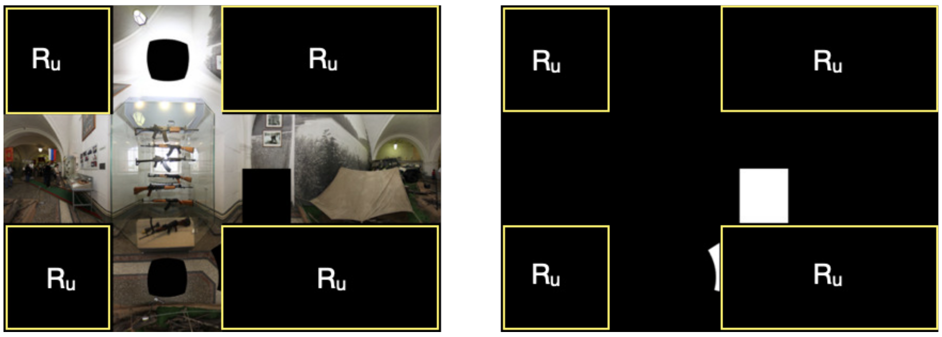

Figure 3.5: Cube Map input and mask conversion from Equirectangular projection

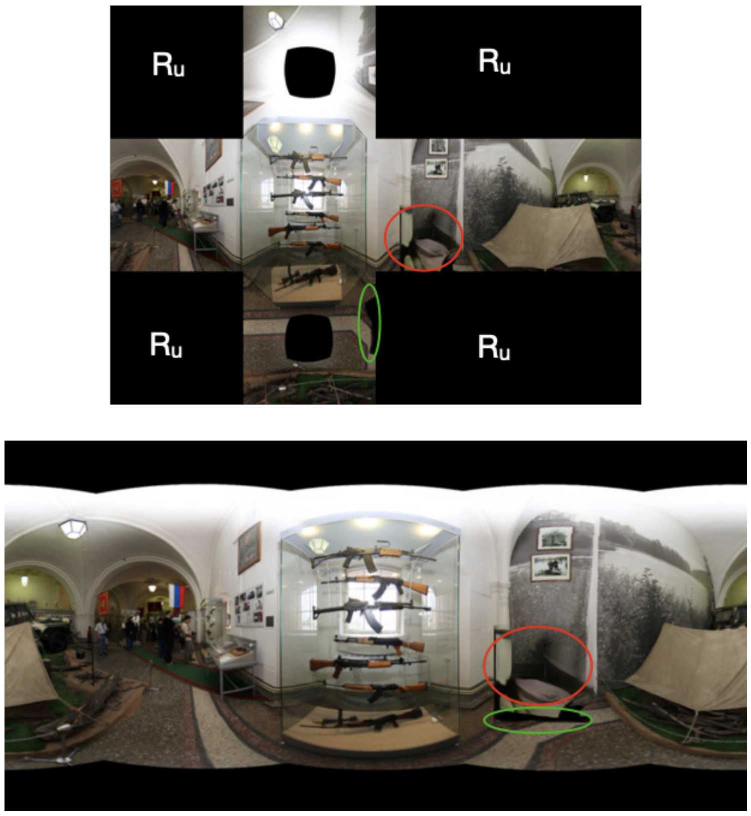

Figure 3.6: Cube Map inpainted image

For the second experiment for cubemap, we split both the input image with patch and their masks into their respective faces of its cube map representation, and fed the network each of the face set (image face + mask face) separately to inpaint them and convert them back into equirectangular projection after all the faces are inpainted independently. This is shown in Figure 3.8 for input image as shown in Figure 3.7. For this we took input images of the resolution 960×480, 0.25 times of the size of the original input. We took higher resolution in this case, as each face is split into smaller resolution of 240×240. For both the cube-map tests we used the baseline model [6] to inpaint the patches.

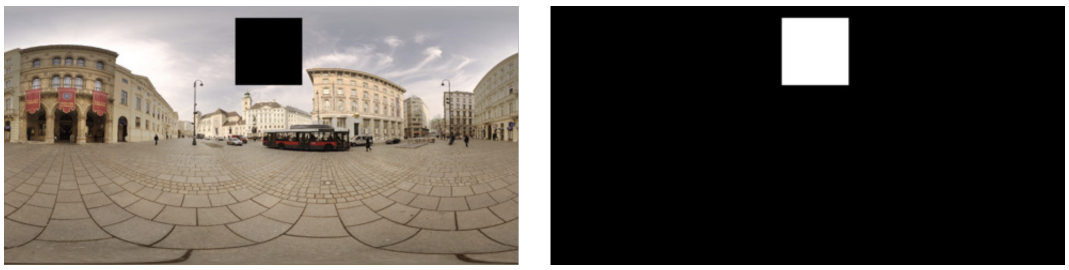

Figure 3.7: Equirectangular Input Images

We can see that the difference between ground truth and the inpainted output after stitching back together are completely different and the output is incoherent as well. On observing the masks, we find that there are only two faces with the entire region that needs to be inpainted. This means that the generator of our network has to inpaint such a large area using very less information (only one face). We hypothesise this as the reason for the model, failing to inpaint. Moreover, even after assuming that the patch (unknown pixels) gets divided equally between all faces or that the patch is very small, the model gets to see only one face at a time. This means that the contextual information from other faces is not available for the generator to inpaint from. Based on the training methods, this method would always produce less meaningful inpainted outputs in comparison to a scenario in which the generator has the whole image available to gather features and information from. Our hypothesis is bolstered by the results of tangent images and equirectangular images, as discussed later in this section. Further results of cube-map projection using the above technique is presented in Appendix A.2.1.

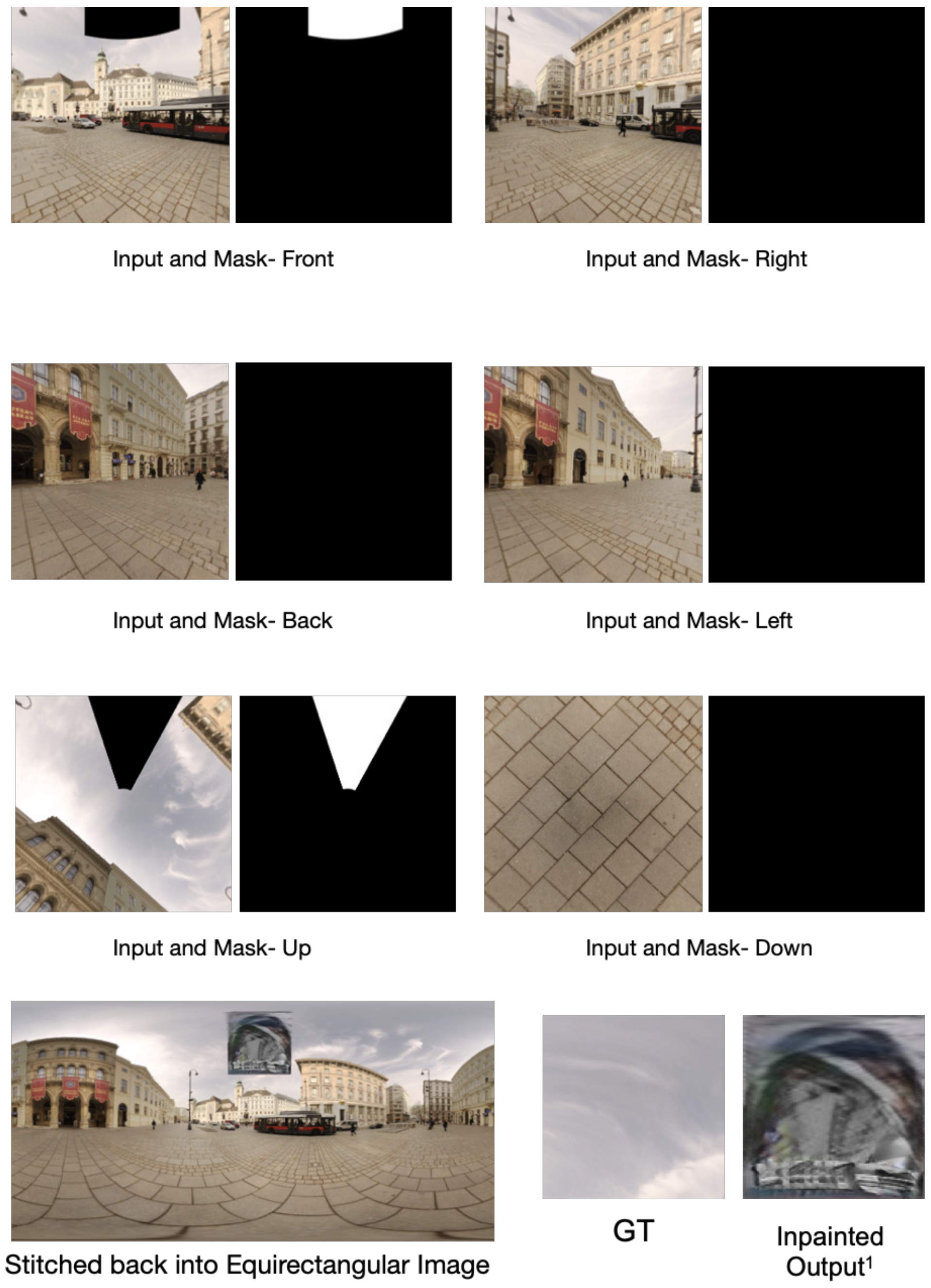

Figure 3.8: Cubemap- Faces as input to network and stitched back to equirectangular image and compared with ground truth

[1]BL- Baseline Model- [6]

In case of testing the tangent images, based on cube-map's output we tested it against 20 subdivision, i.e. $k = 0$ where k is the level, from equation 2.27, instead of $k = 1$ with 80 subdivision. The latter would have created 80 subdivided tangent images, meaning the generators had way less information while inpainting if that tangent image had a patch. We have shown few tangent image and masks for $k = 1$ in Appendix A.2.2 to understand how less information would have been available to the generator to inpaint from. Our input image was approximately level 10 icosahedron based on pixel-face analogy i.e. $s = 0$ and $b = 0$, giving us tangent images of resolution 1024×1024. Below in Figure 3.9, we show some of the tangent images and their masks from the total 20, where there exist region that needed to be inpainted for input image as in Figure 3.7. The results does correlate with our hypothesis and are poorer than cube-map's output in terms of coherence.

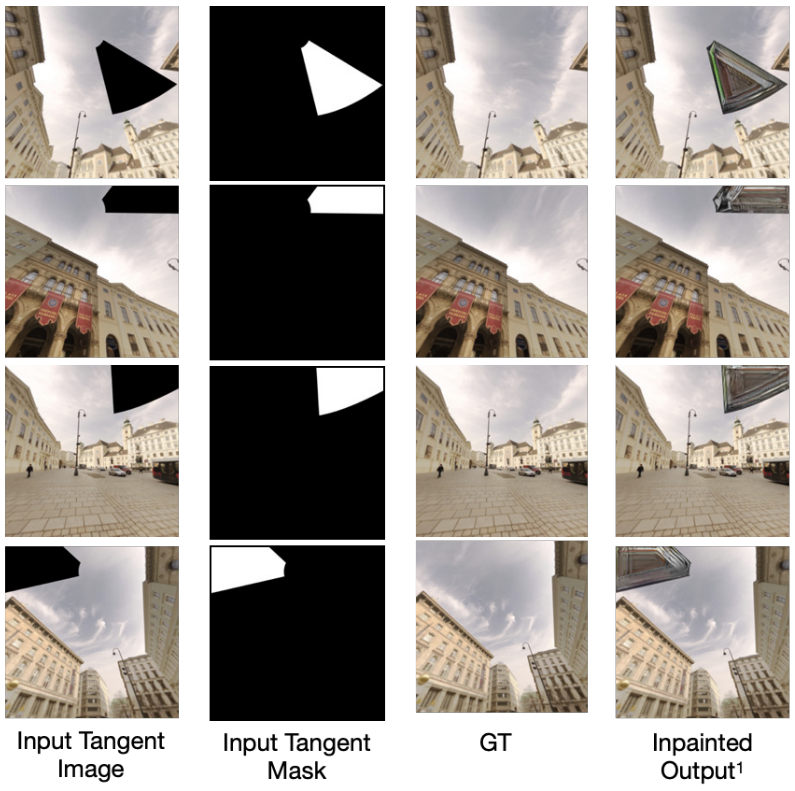

Figure 3.9: Tangent Images- Ground Truth and Inpainted Output Comparison

[1]BL- Baseline Model- [6]

As mentioned in section 3.3, our tests for equirectangular are of three categories, *S1*, *S2* and Patch Resolution Comparison. For these experiments we also use PSNR and SSIM, for quantitative metrics.

3.5.2 PSNR

Peak Signal to Noise Ratio (PSNR) is an indicator that is measured in decibels, measures the quality of reconstructed image to that of the original image. As the name suggests, PSNR is calculated by the ratio between the maximum possible power of the signal to that of peak noise that affects the quality of the signal. PSNR is directly proportional to the quality of reconstruction and therefore indirectly proportional to the error. It is given by Equation 3.1,

$$PSNR = 10 \log_{10} \frac{P_{MAX}}{MSE}, \tag{3.1}$$

where P_{MAX} denotes the maximum possible power of the signal, in our case, the image, where P_{MAX} denotes the highest value an 8-bit image can have, that is 255. Also, MSE denotes the peak noise and is given by equation 3.2, where M denotes all the pixels in the concerned patch P, and P_{GT} and P_{MG} corresponds to the ground truth patch and model generated patch respectively.

$$MSE = \frac{1}{M} \sum_{k}^{M} (P_{GT}^{k} - P_{MG}^{k})^2. \tag{3.2}$$

3.5.3 SSIM

Proposed by Zhou Wang et al.[48], Structural Similarity Index (SSIM) as the name suggests, measures the similarity in the structure between the images. Similar to PSNR, SSIM is directly proportional to the quality of the reconstruction and indirectly proportional to the error. It takes the effects of luminance (l), contrast (c) and structure (s) into account, and therefore SSIM is generally considered to calculate from the human's perceived quality [49] . It is given by equation 3.3 [48, 49].

$$SSIM(P_{GT}, P_{MG}) = [l(P_{GT}, P_{MG})]^{\alpha} \cdot [c(P_{GT}, P_{MG})]^{\beta} \cdot [s(P_{GT}, P_{MG})]^{\gamma}. \tag{3.3}$$

Here, the luminance (l), contrast (c) and structure (s) are given by equations 3.4, 3.5 and 3.6.

$$l(P_{GT}, P_{MG}) = \frac{2\mu_{P_{GT}}\mu_{P_{MG}} + C_1}{\mu^2_{P_{GT}} + \mu^2_{P_{MG}} + C_1}. \tag{3.4}$$

$$c(P_{GT}, P_{MG}) = \frac{2\sigma_{P_{GT}}\sigma_{P_{MG}} + C_2}{\sigma^2_{P_{GT}} + \sigma^2_{P_{MG}} + C_2}. \tag{3.5}$$

$$s(P_{GT}, P_{MG}) = \frac{\sigma_{P_{GT}P_{MG}} + C_3}{\sigma_{P_{GT}}\sigma_{P_{MG}} + C_3}. \tag{3.6}$$

Here, $\mu_{P_{GT}}$ and $\mu_{P_{MG}}$ are local means and $\sigma_{P_{GT}}$ and $\sigma_{P_{MG}}$ are local standard deviations for patches P_{GT} and P_{MG}. Similarly, $\sigma_{P_{GT}P_{MG}}$ denotes the co-variance between patches P_{GT} and P_{MG}. The constants C_1 and C_2 are given by $(K_1L)^2$ and $(K_2L)^2$, where the authors chose $K_1 = 0.01$ and $K_2 = 0.03$ [48, 49]. Here, L is similar to P_{MAX} in equation 3.1, denoting the highest value an 8-bit image can have, that is 255.

Generally, α,β, and γ equal to 1, and $C_3 = C_2/2$ [48], this simplifies equation 3.3 as equation 3.7

$$SSIM(P_{GT}, P_{MG}) = \frac{(2\mu_{P_{GT}}\mu_{P_{MG}} + C_1)(2\sigma_{P_{GT}P_{MG}} + C_2)}{(\mu^2_{P_{GT}} + \mu^2_{P_{MG}} + C_1)(\sigma^2_{P_{GT}} + \sigma^2_{P_{MG}} + C_2)} \tag{3.7}$$

Both PSNR and SSIM are not the right metrics for our study here, as our task to inpaint comes under no-reference metric. The goal of inpainting to remove unwanted objects or other artefacts, inherently makes the generated image/patch (P_{MG}) different from original image/patch (P_{GT}), therefore making both PSNR and SSIM a poor metric. Despite being aware, state of the art inpainting techniques and model continue to report PSNR and SSIM score as a quantitative metric for their study. Therefore we would report the scores in our study as well.

3.5.4 Equirectangular Images

In Figure 3.10 and 3.11, we see the inpainted outputs for our models *S1* and *S2*, compared with other state of the art models and ground truth. More results of entire images of the inpainted output from our models are in Appendix A.2.3.

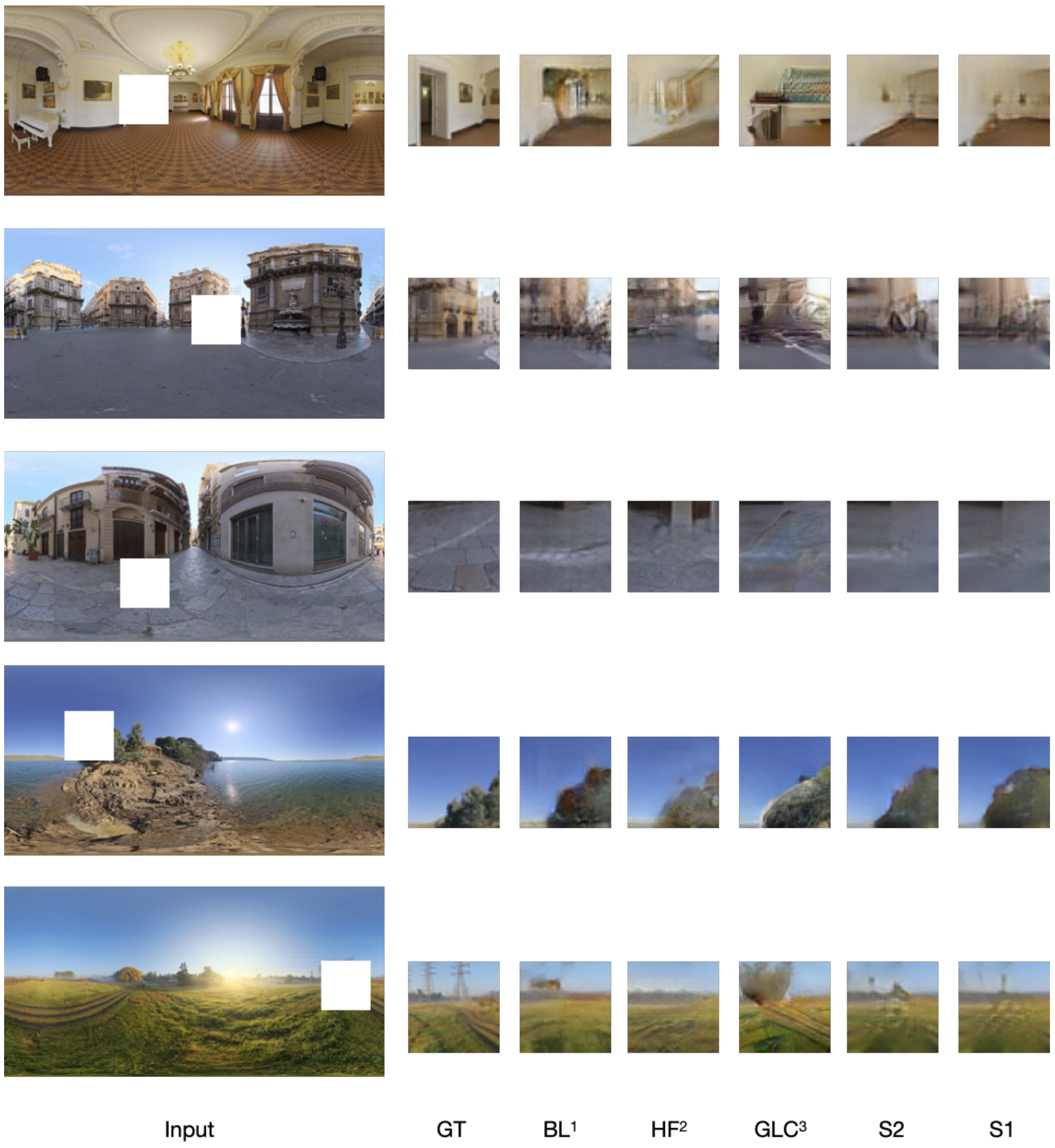

Figure 3.10: Inpainting Comparison 1- Patch: (50×50), Input: (384×192)

[1]BL- Baseline Model- [6]
[2]HF- HiFill Model- [7]
[3]GLC- Globally and Locally Consistent Model- [5]

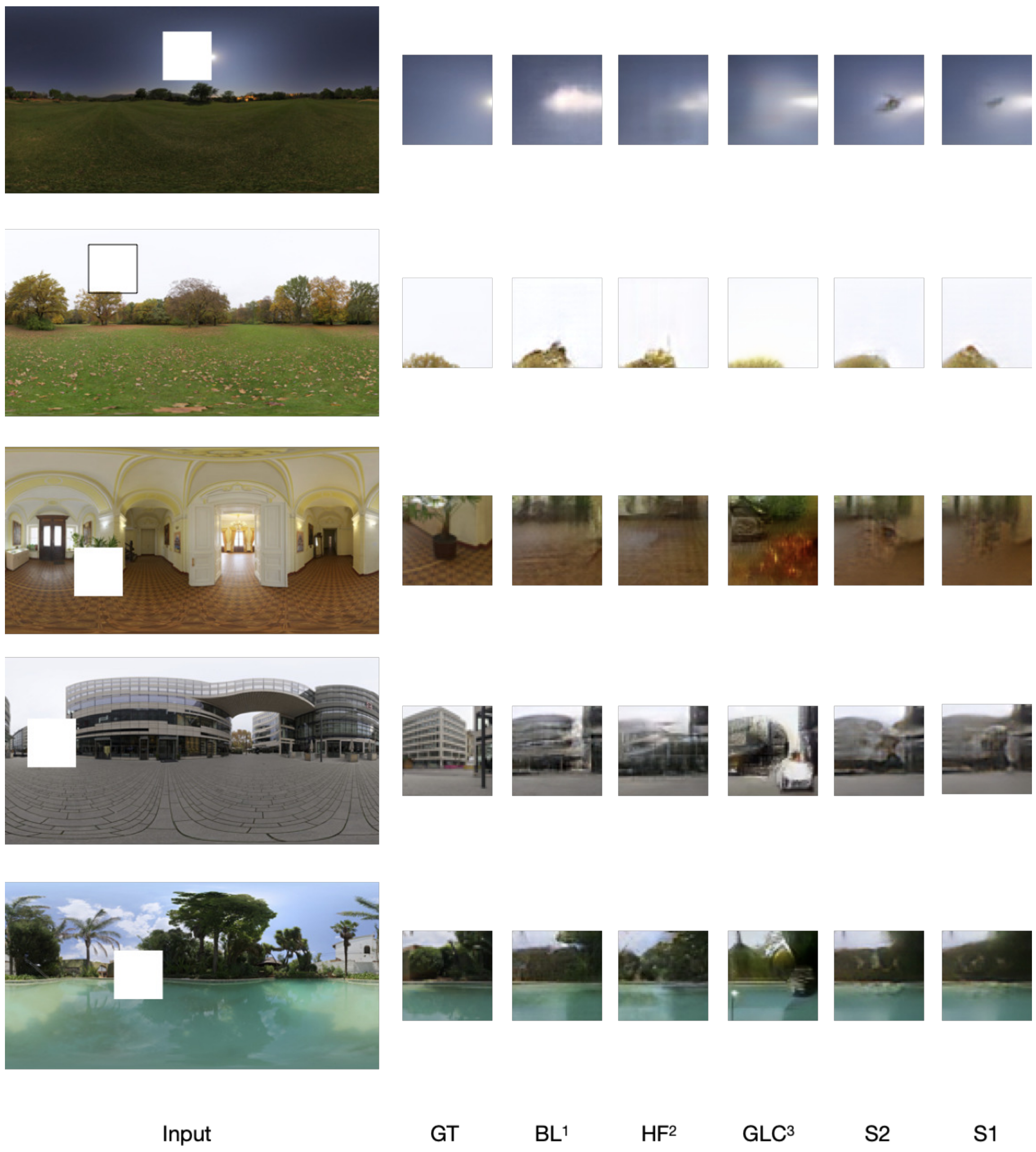

Figure 3.11: Inpainting Comparison 2- Patch: (50×50), Input: (384×192)

[1]BL- Baseline Model- [6]
[2]HF- HiFill Model- [7]
[3]GLC- Globally and Locally Consistent Model- [5]

We can see that our models performed similar or better than existing state of the art models. Also, due to higher availability of contextual information unlike cube-map and tangent images, the results are better. This is also reflected in the quantitative metrics as seen in Table 3.1. Table 3.1 shows the PSNR and SSIM values for equirectangular image inpainting by our models compared with other state of the art models. The table shows average values over 25 images, where the input image is of the size 384×192 with a patch of the size 50×50.

Model	**PSNR**	**SSIM**
Baseline [6]	23.803	0.572
HiFill [7]	23.417	0.556
GLC [5]	21.676	0.533
Ours (*S2*)	**24.815**	**0.616**
Ours (*S1*)	**24.961**	**0.619**

Table 3.1: Quantitative Comparisons- Patch: (50×50), Input: (384×192)

However, we also wanted to understand how the quality of inpainting changed with the size of the image, keeping patch of the image constant (50×50). For this, we chose three size Large (768×384), Medium (576×288) and Small (384×192), as mentioned in section 3.4. We can visualise the differences in PSNR and SSIM in Figures 3.12 and 3.13 respectively.

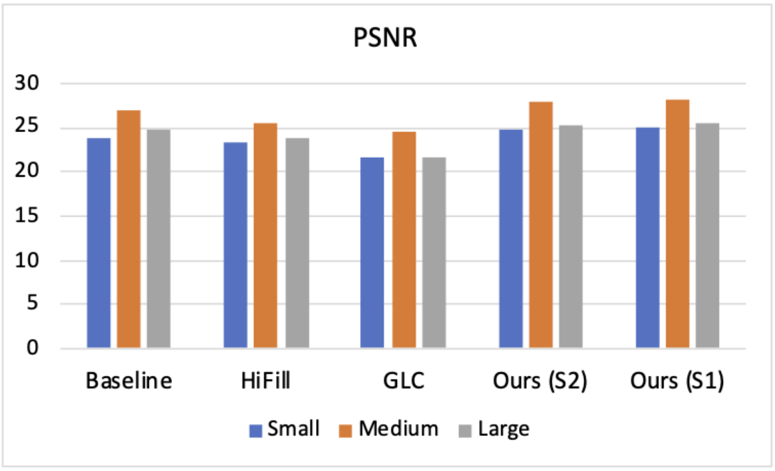

Figure 3.12: PSNR Comparison- Large vs Medium vs Small on 50×50 patch.

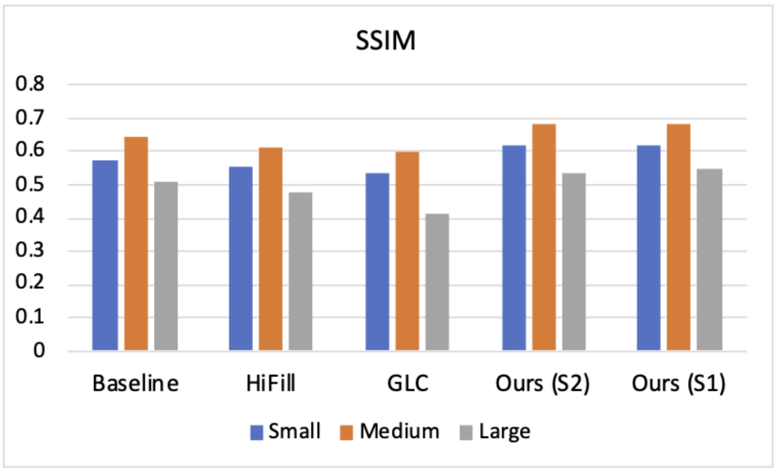

Figure 3.13: SSIM Comparison- Large vs Medium vs Small on 50×50 patch.

From these plots, it is interesting to note that the large image (768×384) inpainting is sometimes as difficult as inpainting a small image. It is to be noted that the (50×50) patch in a large image is not the same as in a medium or small image, as the contents vary. Apart from that, as mentioned earlier, PSNR and SSIM are not the perfect metrics for inpainting and image generation tasks.

For patch resolution comparison, we fixed the resolution of the image, however varied the patch size. For this task, as mentioned in section 3.3, we used a model which was trained on images of the resolution (384×192) with patches of size (32×32). However, while testing, the patch resolution was varied from (10×10) to (32×32) and to (50×50) as shown in Table 3.2. Similar to previous comparison, it is to be noted that, due to varying patch resolution, with image resolution being constant, the region that needs to be inpainted keeps changing. The recorded values are averages over 21 images. We can infer that it was quality of the model inpainting a smaller patch was higher, followed by medium and large patches when trained to inpaint a medium patch.

Patch Size	PSNR	SSIM
10×10	28.602	0.691
32×32	23.654	0.567
50×50	20.996	0.502

Table 3.2: Patch Size Comparison- Model Trained on 32×32 patch

Chapter 4

Conclusion and Future Work

4.1 Conclusion

The primary goal of this research was to be able to inpaint omni-directional images in a way that is semantically meaningful and coherent, and we were able to successfully demonstrate that in this research. The results of this research as discussed were at par or better than existing state of the art. We explored various representations of ODIs for the purpose of inpainting. During the various experiments, we were also able to hypothesise the results presented, especially the shortcomings of representations like cube-map and tangent images. Despite the promising results of our fine-tuned model, we believe that the quality of inpainting could be made better with further work on tuning the network parameters and the network architecture, catering to the omni-directional images specifically.

While research on inpainting has been explored more frequently in the recent years, inpainting on omni-directional image is an area seldom discussed about. Our results show that externally handling distortion through various projection techniques are less significant due to the way generative networks are designed. Therefore, exploring techniques to handle distortion within the network architecture is right way forward for this research.

4.2 Future Work

One of the key reasons for a relatively lower quality inpainting on ODIs than on a regular 2D image, by the models discussed, is in the process of convolution. Convolutional Neural Networks was fundamentally designed for images where objects, animals, humans and everything else in the image are spatially invariant. This means that an object like a ball, or an animal like a dog would appear the same irrespective of its position in the image. However, that is not the case with ODIs. The same object discussed earlier would appear normal (distortion-free) when in centre of an ODI and distorted (stretched, for example) if the object is placed in the extremities of the ODI. This means by using regular kernels or techniques like max-pooling, result in a poorer outcome. This calls for a change in convolution process to mitigate the distortion and spatial variance that exist in ODIs. One such potential solution to work in future could be Spherical CNNs, [50, 51] which suggests changes to sampling locations of convolutional filters for ODIs.

In this study we also discussed the impact of change in resolution of image and patch size. Implementations like partial convolution [52, 53] and gated convolution [7, 54], allow not only higher sized patches to be inpainted on training with a lower sized patch, but also allows for random shaped, arbitrary sized patches to be inpainted. This removes any constraint on patches, and therefore is also something that needs to be explored.

Apart from these, there have been studies that have used Flow Normalising for tasks like Super resolution whose outputs have been more stochastic and much more richer distribution than deterministic outputs of GANs [55, 56]. Exploring flow normalisation for the task of inpainting is also an interesting potential future work.

Bibliography

[1] R. Atienza, *Advanced Deep Learning with Keras: Apply Deep Learning Techniques, Autoencoders, GANs, Variational Autoencoders, Deep Reinforcement Learning, Policy Gradients, and More.* Packt Publishing, 2018.

[2] T. Bonu, "Image inpainting with deep learning." "`https://medium.com/jamieai/image-inpainting-with-deep-learning-dd8555e56a32`", 2018. [online] Accessed: 21-Aug-2020.

[3] N. Zhang, H. Ji, L. Liu, and G. Wang, "Exemplar-based image inpainting using angle-aware patch matching," *EURASIP Journal on Image and Video Processing*, vol. 2019, p. 70, Jul 2019.

[4] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2536–2544, 2016.

[5] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Globally and locally consistent image completion," *ACM Trans. Graph.*, vol. 36, July 2017.

[6] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, "Generative image inpainting with contextual attention," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5505–5514, 2018.

[7] Z. Yi, Q. Tang, S. Azizi, D. Jang, and Z. Xu, "Contextual residual aggregation for ultra high-resolution image inpainting," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7505–7514, 2020.

[8] E. Ghaderpour, "Some Equal-area, Conformal and Conventional Map Projections: A Tutorial Review," *Journal of Applied Geodesy*, vol. 10, pp. 197–209, Sept. 2016.

[9] D. Podborski, E. Thomas, M. Hannuksela, S. Oh, T. Stockhammer, and S. Pham, "Virtual reality and dash," in *Proceedings of International Broadcast Convention (IBC) Conference*, (Amsterdam, The Netherlands), pp. 1–11, 09 2017.

[10] M. Eder, "360° computer vision: Problems, solutions, and a suggestion for the path ahead." "`https://medium.com/@meder411/360-computer-vision-problems-solutions-and-a-suggestion-for-the-path-ahead-438831c6277b`", 2020. [online] Accessed: 28-Aug-2020.

[11] M. Eder, M. Shvets, J. Lim, and J. Frahm, "Tangent images for mitigating spherical distortion," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (Los Alamitos, CA, USA), pp. 12423–12431, IEEE Computer Society, jun 2020.

[12] O. Elharrouss, N. Almaadeed, S. Al-Maadeed, and Y. Akbari, "Image inpainting: A review," *Neural Processing Letters*, vol. 51, pp. 2007–2028, Apr 2020.

[13] J. Khan, "Guide to image inpainting: Using machine learning to edit and correct defects in photos." "`https://heartbeat.fritz.ai/guide-to-image-inpainting-using-machine-learning-to-edit-and-correct-defects-in-photos-3c1b0e13bbd0`", 2019. [online] Accessed: 05-Sep-2020.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2672–2680, Curran Associates, Inc., 2014.

[15] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, *Object Recognition with Gradient-Based Learning*, pp. 319–345. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, p. 1735–1780, Nov. 1997.

[17] L. Weng, "From gan to wgan." "`http://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html`", 2017. [online] Accessed: 20-Aug-2020.

[18] G. Biau, B. Cadre, M. Sangnier, and U. Tanielian, "Some Theoretical Properties of GANs," Mar. 2018. arXiv preprint, [arXiv:1803.07819].

[19] L. Weng, "From gan to wgan," 2019. arXiv preprint, [arXiv:1904.08994].

[20] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems 29* (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), pp. 2234–2242, Curran Associates, Inc., 2016.

[21] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," vol. 70 of *Proceedings of Machine Learning Research*, (International Convention Centre, Sydney, Australia), pp. 214–223, PMLR, 06–11 Aug 2017.

[22] J. Allingham, "Wasserstein gan." "`https://www.depthfirstlearning.com/2019/WassersteinGAN`", 2019. [online] Accessed: 20-Aug-2020.

[23] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), p. 5769–5779, Curran Associates Inc., 2017.

[24] J. Hui, "Gan - wasserstein gan & wgan-gp." "`https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490`", Mar 2020. [online] Accessed: 20-Aug-2020.

[25] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, "Image inpainting," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, (USA), p. 417–424, ACM Press/Addison-Wesley Publishing Co., 2000.

[26] A. Criminisi, P. Perez, and K. Toyama, "Object removal by exemplar-based inpainting," in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 2, pp. II–II, 2003.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing*

Systems 25 (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," vol. 115, p. 211–252, Dec. 2015.

[29] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2015.

[30] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 315–323, JMLR Workshop and Conference Proceedings, 11–13 Apr 2011.

[31] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *International Conference on Learning Representations (ICLR)*, May 2016.

[32] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[33] R. Tyleček and R. Šára, "Spatial pattern templates for recognition of objects with regular structure," in *Proc. GCPR*, (Saarbrucken, Germany), 2013.

[34] G. Chantas, T. Gkamas, and C. Nikou, "Variational-bayes optical flow," *Journal of Mathematical Imaging and Vision*, vol. 50, 11 2014.

[35] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[36] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing textures in the wild," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[37] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of GANs for improved quality, stability, and variation," in *International Conference on Learning Representations*, 2018.

[38] E. Agustsson and R. Timofte, "Ntire 2017 challenge on single image super-resolution: Dataset and study," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[39] R. Timofte, E. Agustsson, L. Van Gool, M.-H. Yang, L. Zhang, B. Lim, *et al.*, "Ntire 2017 challenge on single image super-resolution: Methods and results," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[40] . Apple Inc., "iphone 6 - technical specifications." "`https://support.apple.com/kb/sp705?locale=en_IN`", 2020. [online] Accessed: 28-Aug-2020.

[41] M. Miller (https://mattdm.org), "So my iphone 6 camera lens is as wide as my full frame 35mm dslr lens?." Photography Stack Exchange, 2014. [online] Accessed: 28-Aug-2020.

[42] M. Xu, C. Li, S. Zhang, and P. Le Callet, "State-of-the-art in 360° video/image processing: Perception, assessment and compression," *IEEE Journal of Selected Topics in Signal Processing*, vol. PP, pp. 1–1, 01 2020.

[43] E. Ghaderpour, "Map projection," 2014. arXiv preprint, [arXiv:1412.7690].

[44] A. Papadopoulos, "A note on nicolas-auguste tissot: at the origin of quasiconformal mappings," in *Handbook of Teichmüller theory, Volume VII*, (Zürich, Switzerland), pp. 289–299, European Mathematical Society Publishing House, 2020.

[45] . Google Inc., "Google colab pro." "`https://colab.research.google.com/signup`". [online] Accessed: 31-Aug-2020.

[46] J. Xiao, K. A. Ehinger, A. Oliva, and A. Torralba, "Recognizing scene viewpoint using panoramic place representation," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2695–2702, 2012.

[47] G. Zaal, “Hdris- image collection webpage.” “`https://hdrihaven.com/hdris/`”. [online] Accessed: 1-Sep-2020.

[48] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[49] J. Nilsson and T. Akenine-Möller, “Understanding ssim,” 2020. arXiv preprint, [arXiv:2006.13846].

[50] T. S. Cohen, M. Geiger, J. Köhler, and M. Welling, “Spherical CNNs,” in *International Conference on Learning Representations*, 2018.

[51] B. Coors, A. P. Condurache, and A. Geiger, “Spherenet: Learning spherical representations for detection and classification in omnidirectional images,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[52] G. Liu, K. J. Shih, T.-C. Wang, F. A. Reda, K. Sapra, Z. Yu, A. Tao, and B. Catanzaro, “Partial convolution based padding,” 2018. arXiv preprint, [arXiv:1811.11718].

[53] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro, “Image inpainting for irregular holes using partial convolutions,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[54] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, “Free-form image inpainting with gated convolution,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[55] I. Kobyzev, S. Prince, and M. Brubaker, “Normalizing flows: An introduction and review of current methods,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020.

[56] A. Lugmayr, M. Danelljan, L. V. Gool, and R. Timofte, “Srflow: Learning the super-resolution space with normalizing flow,” 2020. arXiv, Accepted at ECCV 2020, [arXiv:1811.11718].

Appendix A

A.1 Network Architecture

A.1.1 Local and Global WGAN-GP Discriminator

Kernel Shape	**Strides**	**Filters**
5	2	64
5	2	128
5	2	256
5	2	512

Table A.1: Local WGAN-GP Discriminator Architecture

Kernel Shape	**Strides**	**Filters**
5	2	64
5	2	128
5	2	256
5	2	256

Table A.2: Global WGAN-GP Discriminator Architecture

Tables A.1 and A.2 describe the network architecture for local and global WGAN-GP discriminators respectively. All the layers are convolutional layers.

A.1.2 Inpainting Network

Type	**Kernel Shape**	**Strides**	**Filters**	**Dilation**
Convolution	5	1	32	-
Convolution	3	2	64	-
Convolution	3	1	64	-
Convolution	3	2	128	-
Convolution	3	1	128	-
Convolution	3	1	128	-
Convolution	3	1	128	2
Convolution	3	1	128	4
Convolution	3	1	128	8
Convolution[1]	3	1	128	16
Convolution	3	1	128	-
Convolution	3	1	128	-
Deconvolution[2]	-	-	-	-
Convolution	3	1	64	-
Deconvolution[2]	-	-	-	-
Convolution	3	1	16	-
Convolution	3	1	3	-
Clip[3]	-	-	-	-

Table A.3: Inpainting Network

Table A.3 describes architecture of course and refinement network. The entire two stage network is therefore two networks as in Table A.3 stacked together [6].

[1]In refinement network, until this point is considered as hallucination layer.
[2]Deconvolution here refers to convolution layer based upsampling done using nearest neighbour.
[3]Clipping between -1 to 1 instead of using an activation function.

A.1.3 Contextual Attention Branch

Type	**Kernel Shape**	**Strides**	**Filters**
Convolution	5	1	32
Convolution	3	2	32
Convolution	3	1	64
Convolution	3	2	128
Convolution	3	1	128
Convolution	3	1	128
CA Layer[1]	-	-	-
Convolution	3	1	128
Convolution	3	1	128

Table A.4: Contextual Attention Branch Architecture

Table A.4 describes the architecture of contextual attention branch that is present in refinement network in parallel to hallucination branch. The output of this layer is concatenated with hallucination branch's output and used further in refinement network.

[1]Contextual Attention Layer

A.2 Additional Results

A.2.1 Cube-Map Projection based Inpainting

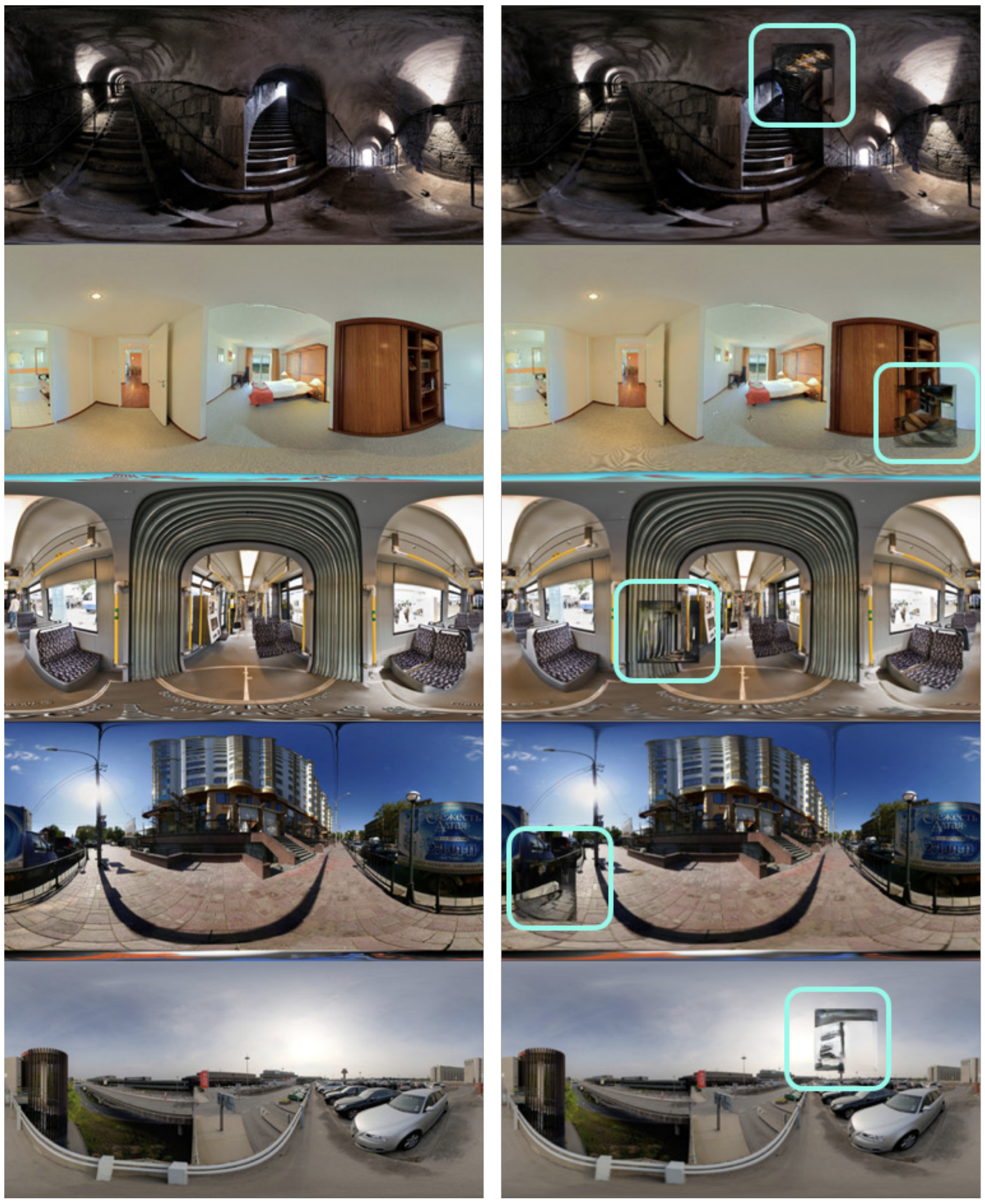

Figure A.1: Cube Map- Ground Truth (left) and Inpainted Output (Right)

A.2.2 Tangent Images

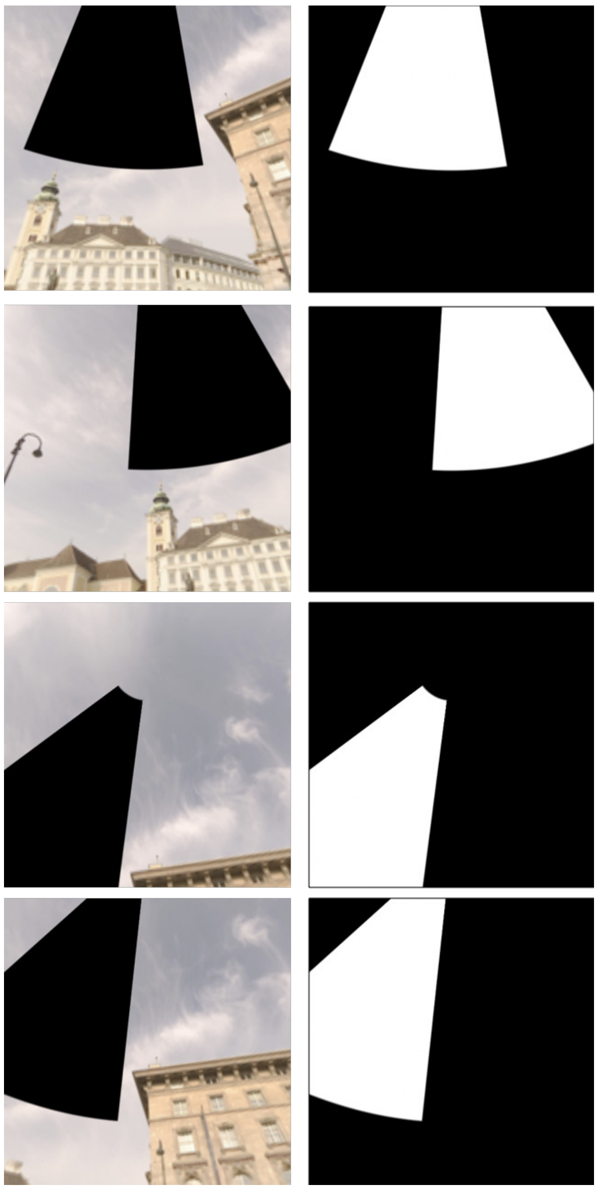
Figure A.2: Tangent Images Input (Left) and its Mask (Right)

In Figure A.2, we can see few of the tangent images and their respective masks, for $k = 1$, making 80 subdivided tangent images, of resolution (512×512). We can see how much area the patch covers and how few information the generator has, to inpaint from.

A.2.3 Equirectangular Inpainting

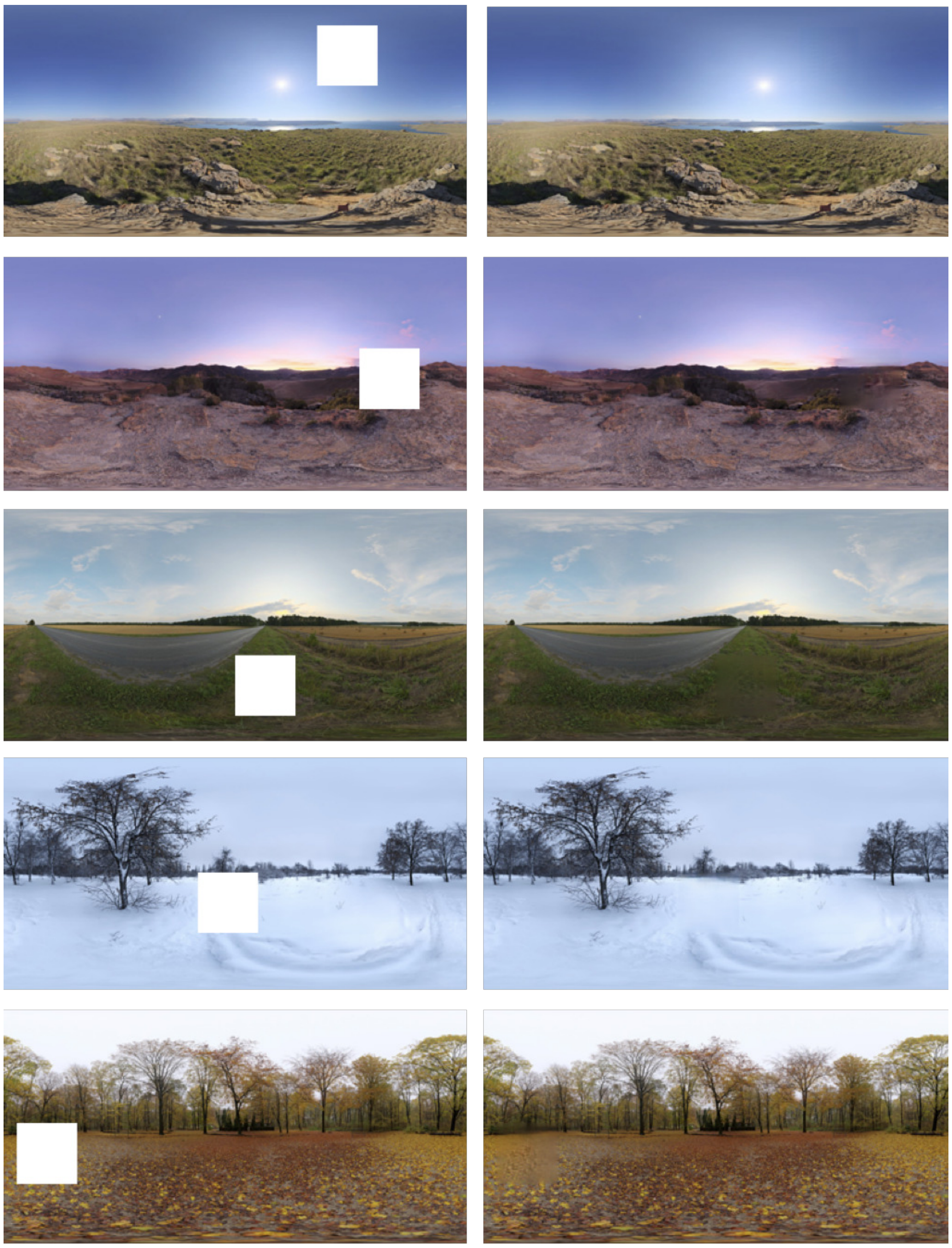

Figure A.3: Input Image (left) and S1 inpainted output (Right) - 1

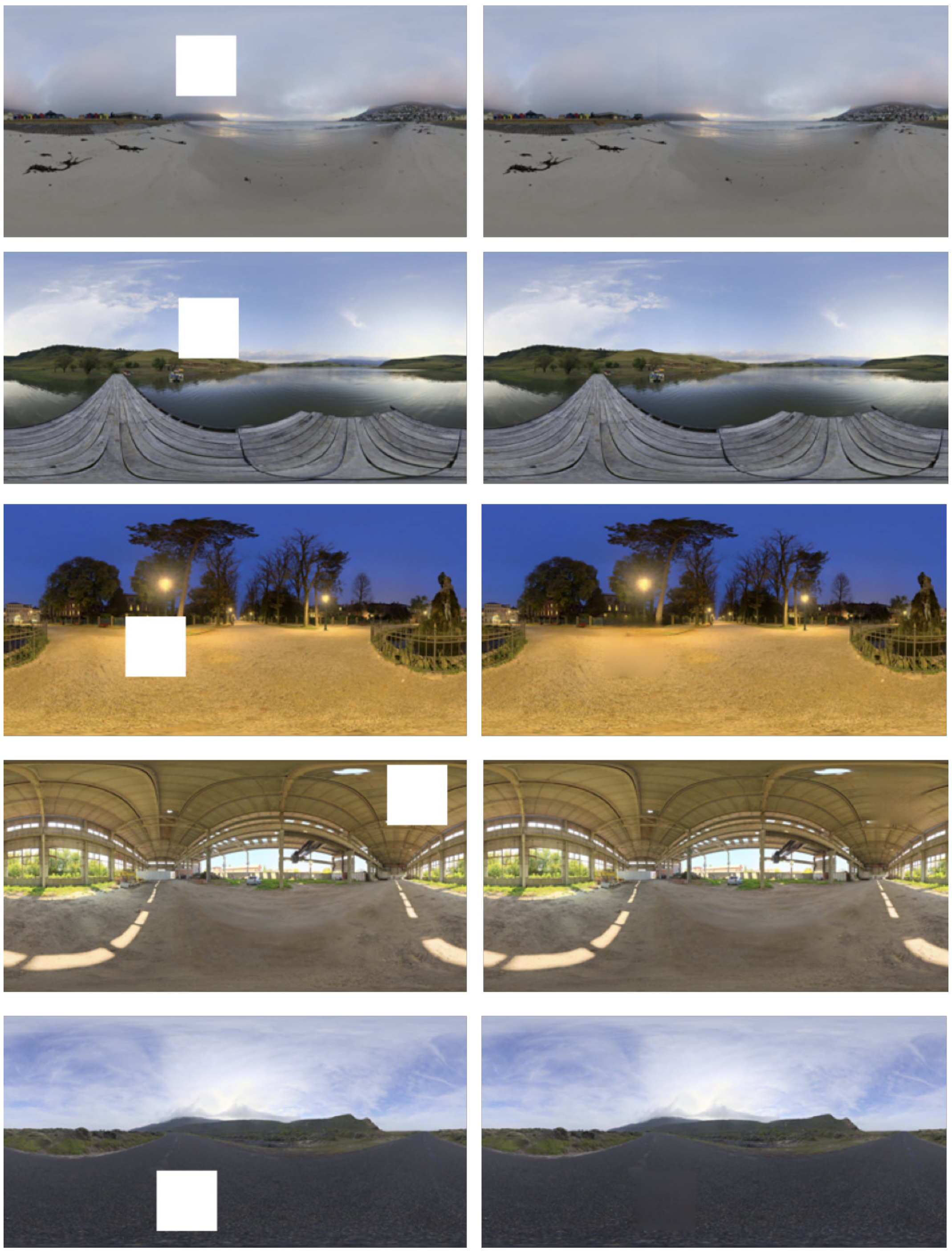
Figure A.4: Input Image (left) and S1 inpainted output (Right) - 2