SAN JOSÉ STATE UNIVERSITY

# CMPE-258 PROJECT REPORT

## "Image Out - painting Using GANs"


**Team: BTS**

Bharath Gunasekaran

Tamanna Mehta

Stuti Agarwal

# TABLE OF CONTENTS


1. Abstract……..
2. Introduction……...
3. Related Work……
4. Data……...
5. Methods…….
6. Experiments……..
7. Conclusion……..……..

## Abstract:

Deep Learning has been highly successful with numerous computer vision tasks but still there are many computer vision problems that can be better resolved with traditional computer vision techniques and Image out-painting is one of them. Image Out-painting is a computer vision problem and involves restoring the missing portions of an image. The aim is to extrapolate unknown areas in such a way that it appears realistic to human eyes. In areas like panorama creation, texture creation and vertically filmed video expansion, image out-painting has been hugely successful with its visually logical structures and textures. We propose to develop an end to end model with a tfx model deployed on some cloud platform using Deep Convolutional Generative Adversarial Networks that implements image out painting on the basis of the neighboring information.


## Introduction:

Image out painting has received little attention compared to Image Inpainting. Where Image inpainting involves restoring the missing interior portions of images, Image out painting restores exterior portions of images referred to as boundaries. As both the techniques are closely related, we can use Generative Adversarial networks in order to hallucinate boundaries with minimal neighboring Information. The whole idea has been implemented on deep learning technique which is based on adversarial training of the network.In this project, we will use the centre portion of image in order to compute a realistic image which is of the same size as the Input Image.

**Figure 1:Image outpainiting**

## Related work:

The initial papers used a data driven approach along with graphical representation of the source image for Image out painting. Although the results obtained were realistic, using adversarial training can lead to better results. In [2], Pathak et al. introduced the concept of Context Encoders, a CNN which was trained adversarially to reconstruct missing portions of the images based on the neighboring pixels. Our painting concept is inspired from the Inpainting as both are closely related. Our work is inspired from the Stanford Research paper Painting Outside the Box: Image Outpainting with GANs. which is written by Mark Sabini and Gili Rusak.

## Dataset:

We used Places 365 dataset which is the most common dataset used for this domain research. It involves a diverse set of images from buildings, landscapes, rooms and scenes from everyday life.It includes 1.8 million colored images with 365 different categories. Images are resized to 128 x 128 from 256 x 256 in order to speed up the training process. The output image produced is of the same size as Input size. The resized image is cropped and masking is done before feeding into the model. For training purposes, we used 500 images and 100 images for the validation dataset.


**Figure 2: Places 365 dataset**

## Data Preprocessing:

In order to prepare images for training the model, the images were normalized in range from 0 to 1. We define a mask of size 128 * 128 . Masking is done to mask out the centre portion of the images. The final masked images are fed to the generator model.

## Methodology:

We used DCGAN architecture which includes Generator and Discriminator with convolutional layers. Generator acts as an encode-decoder convolutional network while the Discriminator uses strided convolutions repeatedly to downsample an image for binary classification as whether the image is real or fake. Figure 3 shows the training pipeline of the model where the generator produces fake images as an output from the input masked images.

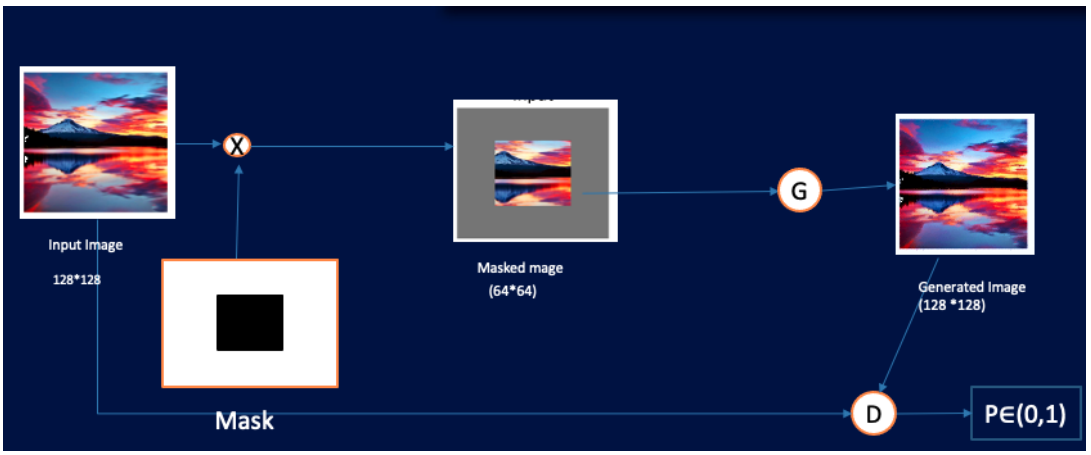


**Figure 3: Training pipeline**

These generated images along with the real images are fed into the discriminator in order to classify images as fake or real. During training, the mini batch of random sample images are fed during each epoch. The parameters of the model are updated based on loss computed in the form of MSE.

## Network Architecture:

For Generator G, we used dilated convolutions along with an Encoder Decoder network in order to increase the receptive field of neurons and provide realistic

images. Figure 3 and 4 provides the layer architecture of the Generator and Discriminator model. It specifies the number of filters, strided convolutions , dilation rate and the number of outputs for each layer. Generator model has been trained with different dilation rates ranging from 1 to 8 inorder to improve realism in the output images. Generator first downsampled the image and then up-sampled it to the original image size.

.

| Type | Filters size | Strides | Dilation rate | Outputs |
|---|---|---|---|---|
| CONV | 5 | 1 | 1 | 64 |
| CONV | 3 | 2 | 1 | 128 |
| CONV | 3 | 1 | 1 | 256 |
| CONV | 3 | 1 | 2 | 256 |
| CONV | 3 | 1 | 4 | 256 |
| CONV | 3 | 1 | 8 | 256 |
| CONV | 3 | 1 | 1 | 256 |
| DECONV | 4 | 0.5 | 1 | 128 |
| CONV | 3 | 1 | 1 | 64 |
| OUT | 3 | 1 | 1 | 3 |

**Figure 4: Generator Layer Architecture**

| Type | Filters size | Strides | Outputs |
|---|---|---|---|
| CONV | 5 | 2 | 32 |
| CONV | 5 | 2 | 64 |
| CONV | 5 | 2 | 64 |
| CONV | 5 | 2 | 64 |
| CONV | 5 | 2 | 64 |
| FC | - | - | 512 |
| FC(output) | - | - | 1 |

**Figure 5: Discriminator Layer Architecture**

Figure 6 is the architecture of our DCGAN model. Every layer of discriminator is followed by the RELU activation except the final layer which uses sigmoid activation since it has to perform binary classification i.e. whether the image is real or not.Generator model uses LeakyRelu activation functions in his each layer as a parameter.

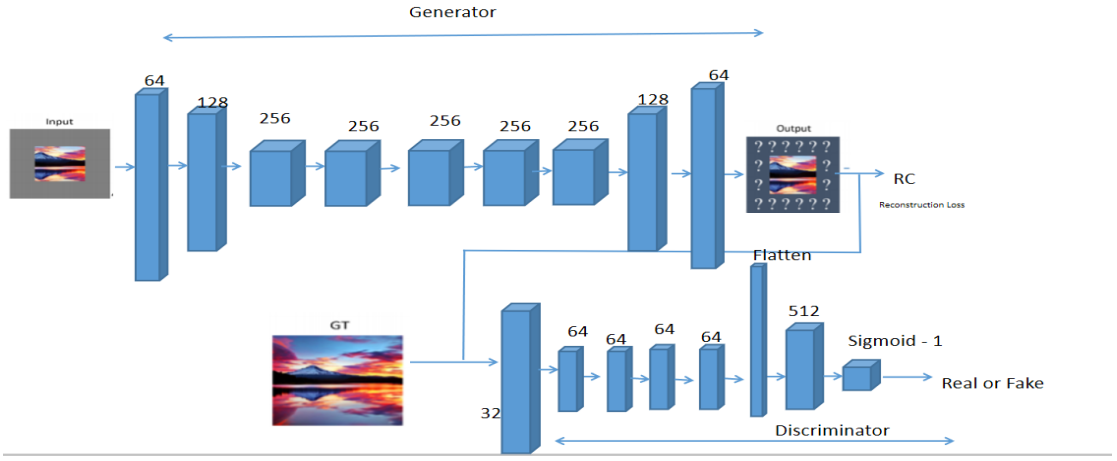


**Figure 6: Architecture of DCGAN model**

**Evaluation Metrics:**

Though the output of the Generator is sufficient to evaluate the model, we used MSE loss of our training batch and epoch as our quantitative metric. We also have the Tensor board visualization for the epoch loss and the batch loss (Figure 10 and 11).

**Experiments & Deployment:**

Initially, we built our 3 models namely Generator, Discriminator and GAN with different specifications. Figure 7, 8 and 9 is the summary model obtained after models creations. For Hyper parameter tuning, we used different dilation rates for our generator model. We used a batch size of 32 for our training. We built the TFX pipeline in which our features and labels were both images. So we have to do the same preprocessing to our labels. The trainer component was trained with all the 3 models after generating TF records, then computing Statistics and Schema Generation. Figure 10 and 11 depicts the visualization obtained from tensorboard. Generator was trained using 200 epochs, discriminator and GAN models were trained with 150 and 220 epochs. Since we had less computing resources, training the entire

dataset would take more than 48 hours and high compute power such as TPUs. So we trained our model on a small dataset and with limited resources.

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 128, 128, 3)]     0
_________________________________________________________________
layer1conv2d (Conv2D)        (None, 64, 64, 32)        2432
_________________________________________________________________
layer1leakyRelu (LeakyReLU)  (None, 64, 64, 32)        0
_________________________________________________________________
layer1dropRate (Dropout)     (None, 64, 64, 32)        0
_________________________________________________________________
layer2conv2d (Conv2D)        (None, 32, 32, 64)        51264
_________________________________________________________________
layer2leakyRelu (LeakyReLU)  (None, 32, 32, 64)        0
_________________________________________________________________
layer2dropRate (Dropout)     (None, 32, 32, 64)        0
_________________________________________________________________
layer2batch (BatchNormalizat (None, 32, 32, 64)        256
_________________________________________________________________
layer3conv2d (Conv2D)        (None, 16, 16, 64)        102464
_________________________________________________________________
layer3leakyRelu (LeakyReLU)  (None, 16, 16, 64)        0
_________________________________________________________________
layer3dropRate (Dropout)     (None, 16, 16, 64)        0
_________________________________________________________________
layer3batch (BatchNormalizat (None, 16, 16, 64)        256
_________________________________________________________________
layer4conv2d (Conv2D)        (None, 8, 8, 64)          102464
_________________________________________________________________
layer4leakyRelu (LeakyReLU)  (None, 8, 8, 64)          0
_________________________________________________________________
layer4dropRate (Dropout)     (None, 8, 8, 64)          0
_________________________________________________________________
layer4batch (BatchNormalizat (None, 8, 8, 64)          256
_________________________________________________________________
layer5conv2d (Conv2D)        (None, 4, 4, 64)          102464
_________________________________________________________________
layer5leakyRelu (LeakyReLU)  (None, 4, 4, 64)          0
_________________________________________________________________
layer5dropRate (Dropout)     (None, 4, 4, 64)          0
_________________________________________________________________
layer5batch (BatchNormalizat (None, 4, 4, 64)          256
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 513
=================================================================
Total params: 887,425
Trainable params: 0
Non-trainable params: 887,425
_________________________________________________________________
```

**Figure 7: Model Summary for Generator**

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 128, 128, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 128, 128, 64)      4864
_________________________________________________________________
activation (Activation)      (None, 128, 128, 64)      0
_________________________________________________________________
batch_normalization (BatchNo (None, 128, 128, 64)      256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 64, 128)       73856
_________________________________________________________________
activation_1 (Activation)    (None, 64, 64, 128)       0
_________________________________________________________________
batch_normalization_1 (Batch (None, 64, 64, 128)       512
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 256)       295168
_________________________________________________________________
activation_2 (Activation)    (None, 64, 64, 256)       0
_________________________________________________________________
batch_normalization_2 (Batch (None, 64, 64, 256)       1024
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 256)       590080
_________________________________________________________________
activation_3 (Activation)    (None, 64, 64, 256)       0
_________________________________________________________________
batch_normalization_3 (Batch (None, 64, 64, 256)       1024
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 64, 64, 256)       590080
_________________________________________________________________
activation_4 (Activation)    (None, 64, 64, 256)       0
_________________________________________________________________
batch_normalization_4 (Batch (None, 64, 64, 256)       1024
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 64, 64, 256)       590080
_________________________________________________________________
activation_5 (Activation)    (None, 64, 64, 256)       0
_________________________________________________________________
batch_normalization_5 (Batch (None, 64, 64, 256)       1024
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 64, 64, 256)       590080
_________________________________________________________________
activation_6 (Activation)    (None, 64, 64, 256)       0
_________________________________________________________________
batch_normalization_6 (Batch (None, 64, 64, 256)       1024
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 128, 128, 128)     524416
_________________________________________________________________
activation_7 (Activation)    (None, 128, 128, 128)     0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 128, 128, 64)      73792
_________________________________________________________________
activation_8 (Activation)    (None, 128, 128, 64)      0
_________________________________________________________________
batch_normalization_7 (Batch (None, 128, 128, 64)      256
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 128, 128, 3)       1731
=================================================================
Total params: 3,340,291
```

**Figure 8: Model Summary for Discriminator**

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 128, 128, 3)]     0
_________________________________________________________________
model_1 (Functional)         (None, 128, 128, 3)       3340291
_________________________________________________________________
model (Functional)           (None, 1)                 887425
=================================================================
Total params: 4,227,716
Trainable params: 3,337,219
Non-trainable params: 890,497
```

**Figure 9: Model Summary for GAN Model**


**Figure 10: Visualizing Batch loss using tensorboard**

The tfx pipeline was finally deployed on GCP cloud with Kube flow frameworks Figure 12 and 13 are the snapshots of Kubeflow Run Graph and notebook on GCP cloud where we have our tfx pipeline deployed. We made a flask application which accepts our colored image as input and predicts the output image of the same size as Input (Fig 14).

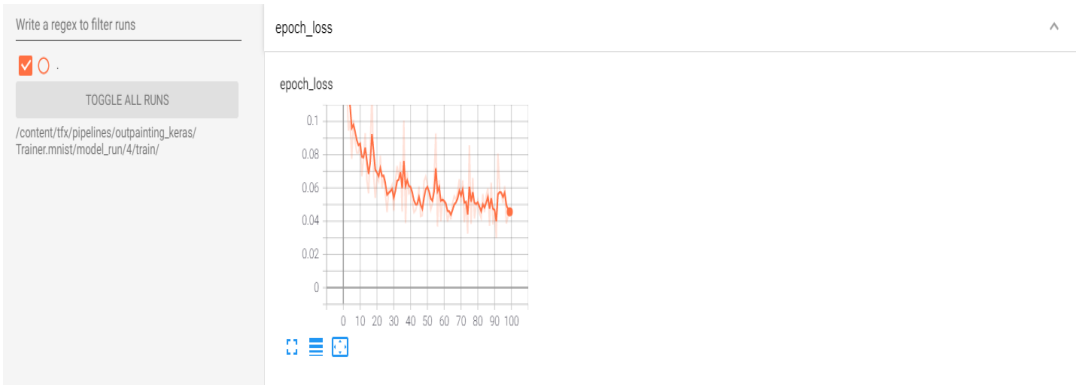


**Figure 11: Visualizing Batch loss using tensorboard**

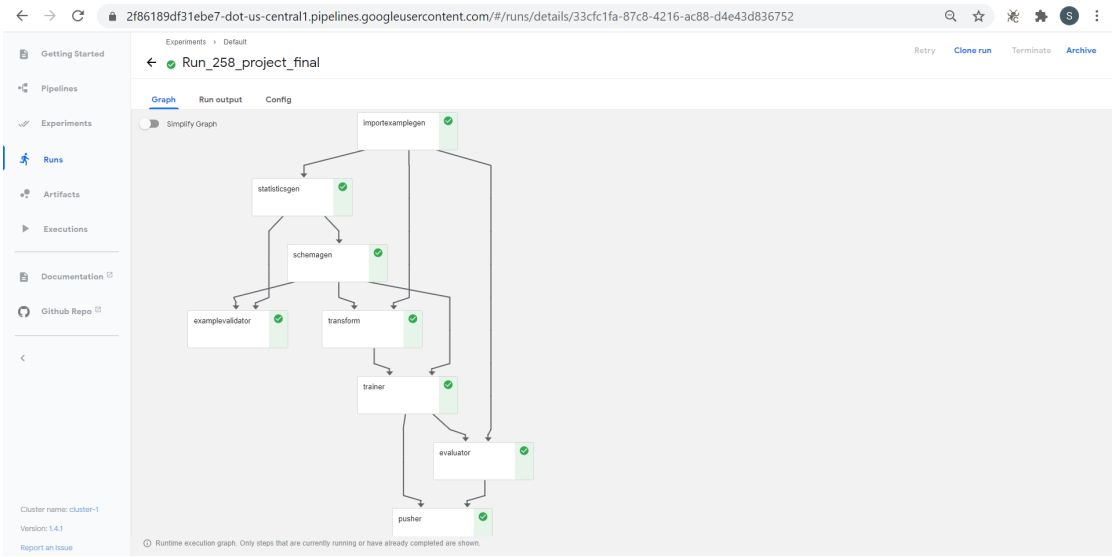


**Figure 12: Kube flow Run Graph**

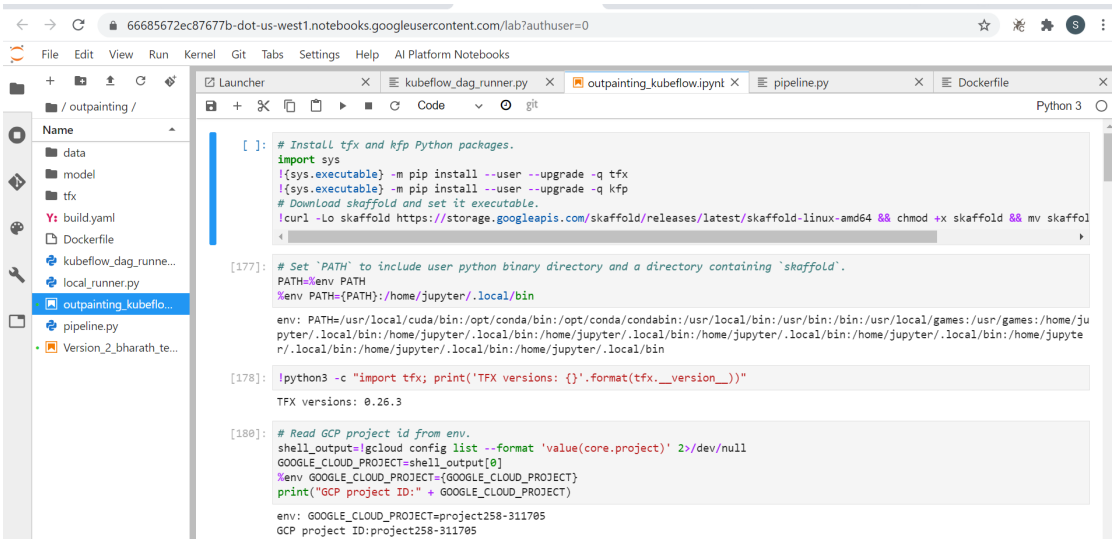


**Figure 13: Out painting notebook running on GCP Cloud**


**Figure 14: Image Out painting (Flask Application Snapshot)**

## Conclusion:-

We were able to successfully do image out painting. Dilated convolutions provided sufficient receptive fields to perform out painting. The quality of the output image can be improvised by increasing the number of epochs during the training process, and increasing the training data. But this is only possible with high compute power and memory resources.

## Future Work:

Image Extrapolation can be done by out painting out of the image boundaries. Recursive Outpainting can also be done in order to increase realism in the image outputs.

## References:

[1]. Mark Sabini, Gill Rusak , “Painting outside the box: Image Outpainting with GANs.” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* Aug 25, 2018.

[2].D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2536–2544, 2016.

[3].Basile Van Hoorick, “Image Out painting and Harmonization using Generative Adversarial Networks” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* Feb 28, 2017.