Deep Learning CS60010
Image-Reconstruction Challenge
(Kaggle Competition)
Final Report

<u>Group No. 26:</u>

Pranav Nyati
Pranav Mehrotra
Rahul Chugh
Aninidita Mandal

Google Drive Link of the Folder containing Trained Model (in .h5 format) and Training Logs:
https://drive.google.com/drive/folders/1qIczR4pKD3khhpQ98P-_NPpphN4UTWRc?usp=sharing

- ## Dataset Description:
    - The training dataset consists of **256*256*3 (RGB) images** of 4 types of animals: **Cat, Dog, Elephant, and Tiger**, with **1750** images of each animal type. There are Masked and Unmasked versions of each image, where the masked image has 2 black square patches of size 75*75. The coordinates of the top-left corner of each patch in each image are given in the CSV file corresponding to each animal.
    - Total no of training images: 4*1750 = 7000 images.

- The test data consists of 50 images of each of the 4 types of animals with two 75*75 black square patches in each image, so a total of 200 images in the test set.
- The goal is to train a model using the masked and unmasked versions of the training data to predict the missing black patches in the test images.

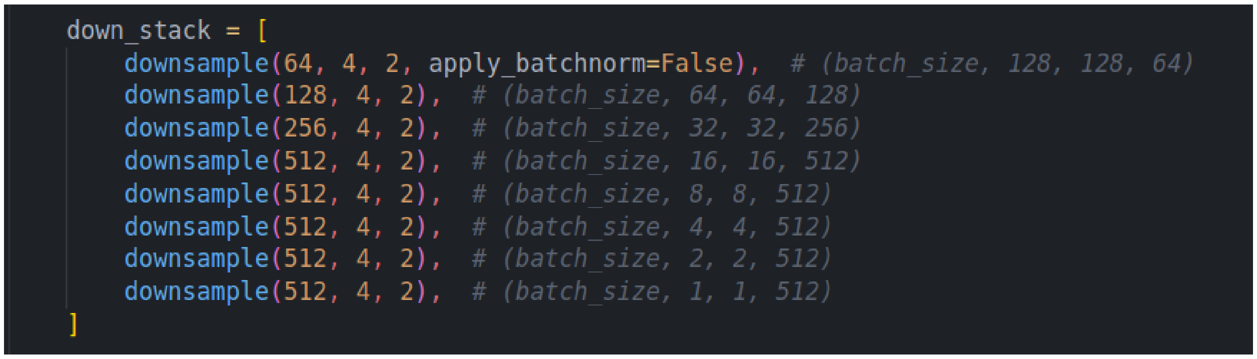
Data-Preprocessing:

- We first obtained the Masked version of the images of the training dataset.
- Next, we also applied various Data Augmentation techniques, such as **Resizing and cropping images, flipping images, Normalization, and adding random noise** to pixels to make learning more robust and increase the training data. We created **4 augmented images from each training image, with a rotation of 90, 180 and 270 degrees and a vertical flip from left to right, thus increasing our training data 5-fold (7000*5 = 35,000),** which considerably reduced our trainingt loss, and the RMSE on the test set.

DL Model used:

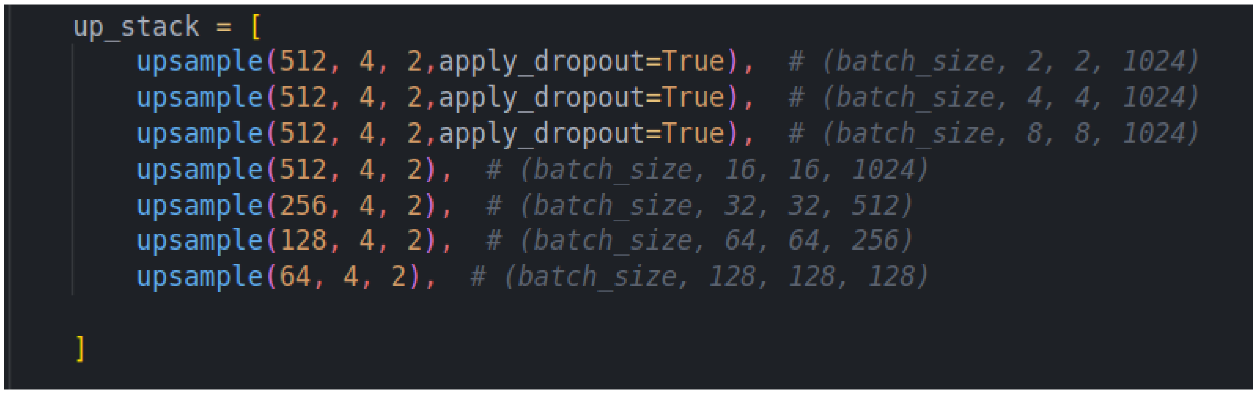
- GAN-based model: We used a **Pix2Pix-GAN-based** architecture with slight modifications for our training. The **Pix2Pix-GAN** model consists of a Generator and Discriminator segments.
- The generator tries to generate the images during the training phase. The discriminator examines the generated image and decides whether it is similar to the images that the discriminator has seen before. The discriminator judges whether the generated image from the generator is similar to the real one by giving some metric value. Based on the discriminator's output, the generator will learn to predict better images.
- Both the Discriminator and the Generator modules consist of 2D convolutional layers to learn the spatial features of the images progressively.
- During prediction, the test images are given as input to the Generator, which gives the complete image with the patch part predictions (predicts the pixel values for the patches) as its output.
- Model Parameters:
  - Trainable params => **54.4 M**
  - Non-Trainable params => **10,880**
- The Generator uses upsampling and downsampling to process the input image and generate the missing pixels.

- The **Downsampling** layers of the **Generator** are as follows:



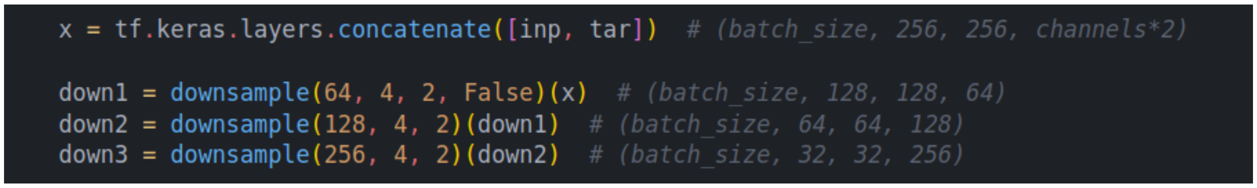
```
down_stack = [
    downsample(64, 4, 2, apply_batchnorm=False),  # (batch_size, 128, 128, 64)
    downsample(128, 4, 2),  # (batch_size, 64, 64, 128)
    downsample(256, 4, 2),  # (batch_size, 32, 32, 256)
    downsample(512, 4, 2),  # (batch_size, 16, 16, 512)
    downsample(512, 4, 2),  # (batch_size, 8, 8, 512)
    downsample(512, 4, 2),  # (batch_size, 4, 4, 512)
    downsample(512, 4, 2),  # (batch_size, 2, 2, 512)
    downsample(512, 4, 2),  # (batch_size, 1, 1, 512)
]
```

- The **Upsampling** layers of the **Generator** are as follows:



```
up_stack = [
    upsample(512, 4, 2,apply_dropout=True),  # (batch_size, 2, 2, 1024)
    upsample(512, 4, 2,apply_dropout=True),  # (batch_size, 4, 4, 1024)
    upsample(512, 4, 2,apply_dropout=True),  # (batch_size, 8, 8, 1024)
    upsample(512, 4, 2),  # (batch_size, 16, 16, 1024)
    upsample(256, 4, 2),  # (batch_size, 32, 32, 512)
    upsample(128, 4, 2),  # (batch_size, 64, 64, 256)
    upsample(64, 4, 2),  # (batch_size, 128, 128, 128)

]
```

- The **Discriminator** also downsamples the input and target image via a series of layers as follows:

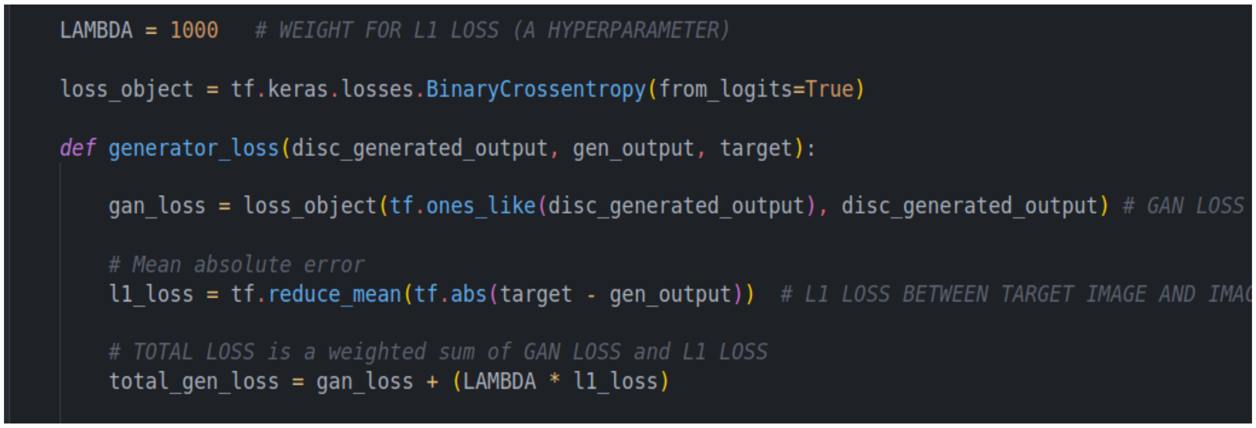


```
x = tf.keras.layers.concatenate([inp, tar])  # (batch_size, 256, 256, channels*2)

down1 = downsample(64, 4, 2, False)(x)  # (batch_size, 128, 128, 64)
down2 = downsample(128, 4, 2)(down1)  # (batch_size, 64, 64, 128)
down3 = downsample(256, 4, 2)(down2)  # (batch_size, 32, 32, 256)
```

- We also experimented with **Stable Diffusion based Image-Inpainting model** from **StabilityAI** using HuggingFace APIs, but the model didn't perform well on the training/test dataset as it is a pre-trained model and is available as a pipeline, and cannot be fine-tuned directly, as its architecture and interface was not available on Hugging Face. Moreover, since it was a prompt-based model, it many a times gave erroneous and highly noisy predictions for some masked images and very accurate predictions on other, but overall didn't do better than our Pix-To-Pix model trained from scratch.

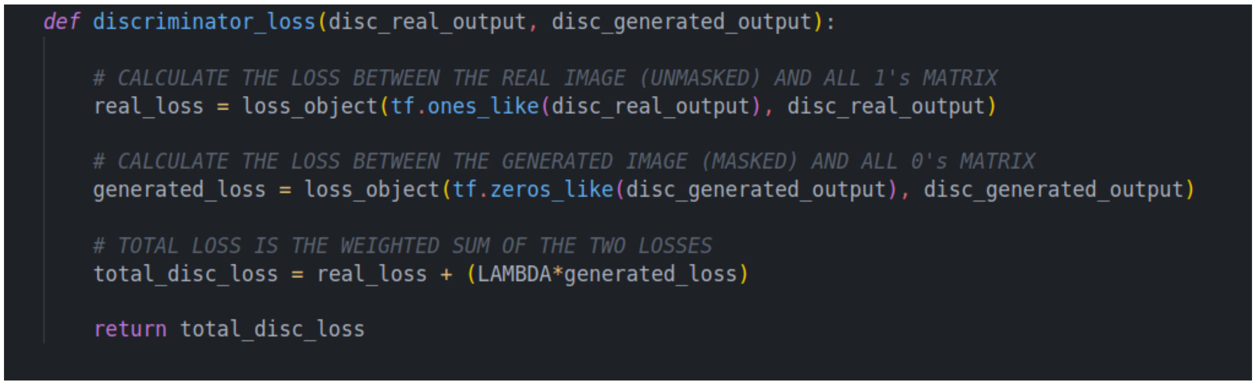
- <u>Hyper-parameter tuning:</u>
    - On running the models a number of times, we found the following set of values best for our predictions:
        1. **A number of upsample and downsample layers in the Generator:**
           We experimented with a different number of upsample and downsample Conv layers for the Generator (from 4-10) and found no of **upsample layers = 7** and **downsample layers = 8** to work best.
        2. **Learning rate, no of training steps, Adam optimizer parameters:**
           After experimenting with a range of values for these hyperparameters, we found a learning rate of **R = 0.0002** to be good (based on training data loss) and used the standard values **b1 = 0.5 and b2 = 0.999 for the Adam optimizer**.
        3. **Value of Lambda for Weighted Losses of Generator and Discriminator:**
           We have a hyperparameter **LAMBDA** which weights the two parts of Generator and Discriminator Losses differently, and we varied the value of LAMBDA from 100, 200, 400, 800, 1000, and 1200 and got the Best Test predictions (MIN RMSE) for the case of **LAMBDA = 1000** over repeated training.

Generator Loss:



```
LAMBDA = 1000   # WEIGHT FOR L1 LOSS (A HYPERPARAMETER)

loss_object = tf.keras.losses.BinaryCrossentropy(from_logits=True)

def generator_loss(disc_generated_output, gen_output, target):

    gan_loss = loss_object(tf.ones_like(disc_generated_output), disc_generated_output) # GAN LOSS

    # Mean absolute error
    l1_loss = tf.reduce_mean(tf.abs(target - gen_output))  # L1 LOSS BETWEEN TARGET IMAGE AND IMAG

    # TOTAL LOSS is a weighted sum of GAN LOSS and L1 LOSS
    total_gen_loss = gan_loss + (LAMBDA * l1_loss)
```

Discriminator Loss:



```
def discriminator_loss(disc_real_output, disc_generated_output):

    # CALCULATE THE LOSS BETWEEN THE REAL IMAGE (UNMASKED) AND ALL 1's MATRIX
    real_loss = loss_object(tf.ones_like(disc_real_output), disc_real_output)

    # CALCULATE THE LOSS BETWEEN THE GENERATED IMAGE (MASKED) AND ALL 0's MATRIX
    generated_loss = loss_object(tf.zeros_like(disc_generated_output), disc_generated_output)

    # TOTAL LOSS IS THE WEIGHTED SUM OF THE TWO LOSSES
    total_disc_loss = real_loss + (LAMBDA*generated_loss)

    return total_disc_loss
```

- ## Results:

We tried for multiple different values of the no of training steps keeping other hyper-params the same, and got the following scores as per the Leaderboard: **(results are from training on the unaugmented dataset)**

(a) 1,50,000 steps: RMSE on 40% of Test Data => **0.19092**
(b) 1,80,000 steps: RMSE on 40% of Test Data => **0.18235**
(c) 2,10,000 steps: RMSE on 40% of Test Data => **0.1926**
(d) 2,50,000 steps: RMSE on 40% of Test Data => **0.19255**

After this we used the **augmented** training dataset with **35,000** images (7000 * 5) to train (with **LAMBDA = 1000)**, and we got the following scores:

(a) 1,50,000 steps: RMSE on 40% of Test Data => **0.16904**
(b) 1,70,000 steps: RMSE on 40% of Test Data => **0.16737**
(c) 1,74,300 steps: RMSE on 40% of Test Data => **0.16781**

- ## Best score achieved on Kaggle:
  - Best score = **0.16737** on the Leaderboard for the model trained on **1,70,000 steps** with an augmented dataset of size **35,000** training images, and for **LAMBDA = 1000**, with the Training Loss curve given below:

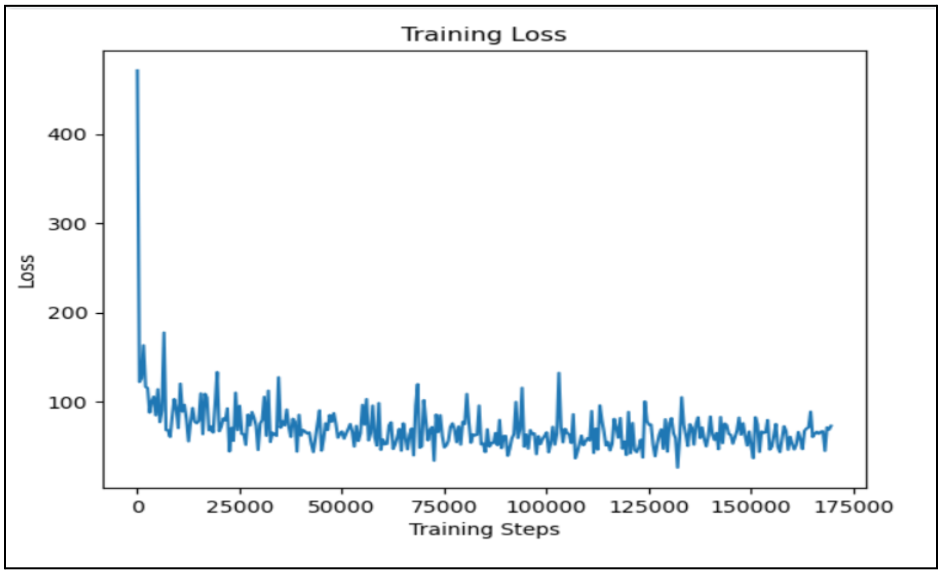
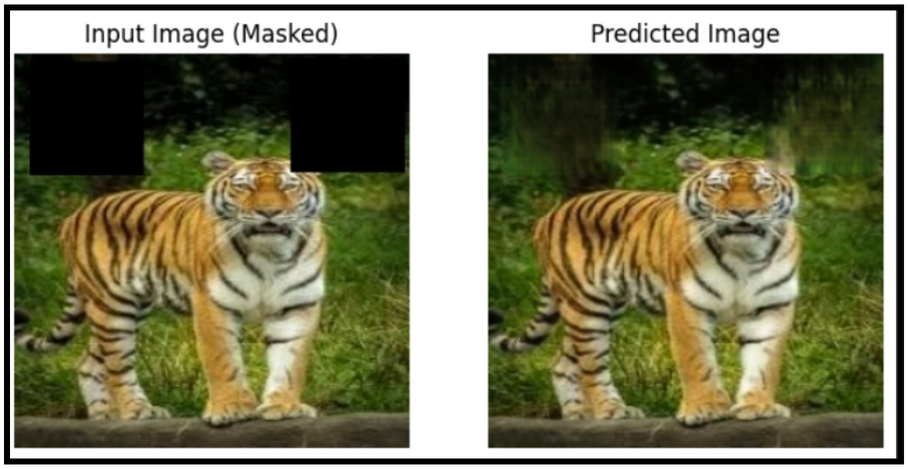
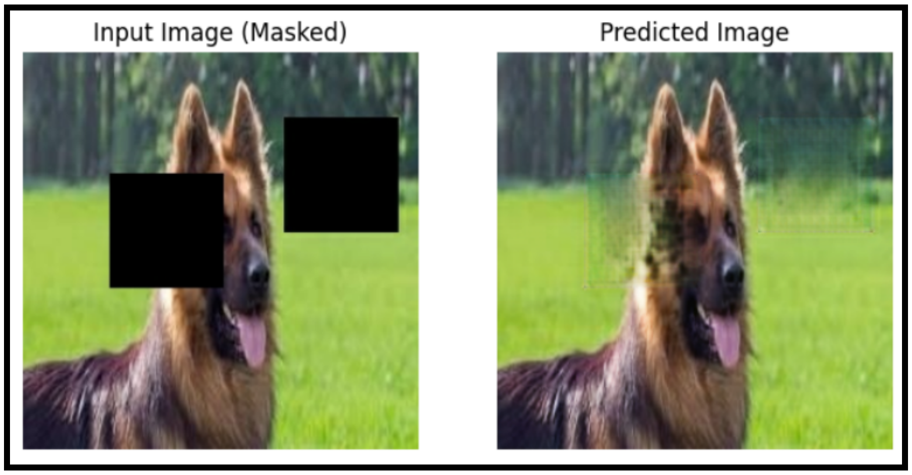


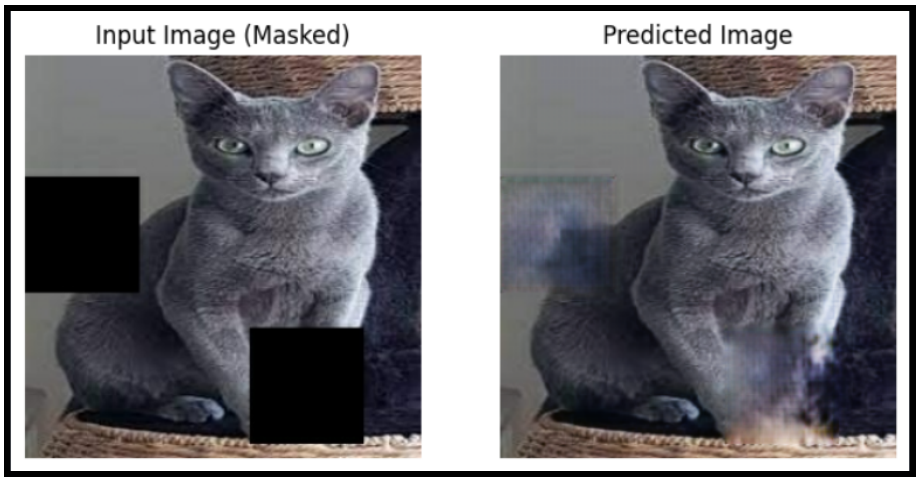
Figure. Training Loss curve (for 1,70,000 steps and Lambda = 1000)

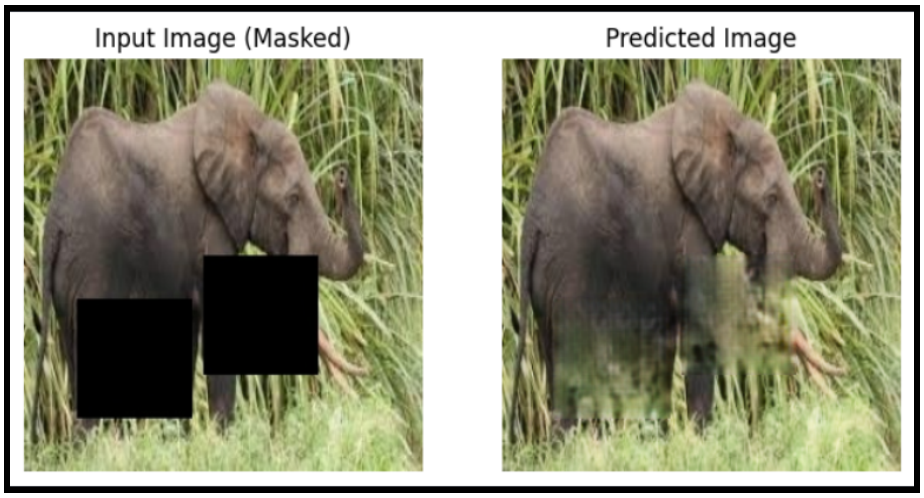
- Next best score = **0.16781** for the model trained on **1,74,300 steps** and **LAMBDA = 1000** on the same augmented dataset.

- <u>Few Examples Of Image Predictions on the Test Dataset (for our Model with Best Predictions (170K steps, LAMBDA = 1000, Trained on Augmented Dataset):</u>




Input Image (Masked)
Predicted Image


Input Image (Masked)
Predicted Image


Input Image (Masked)
Predicted Image