Objective-1 :

Create a cGAN (Conditional GAN) to generate realistic images from sketches, driven by class labels. This involves combining losses, iterative training, and evaluating image quality for accurate synthesis.

1. Methodology

1.1. Data Loading

• The dataset comprises 10015 training images and 1514 testing images, each accompanied by its respective labels having 7 types.

• Custom dataset class *'CustomDataset'* is defined to handle loading and pre-processing of the dataset.

• It takes the image path, sketch path, CSV data frame path, and transformation functions as input parameters.
In the '*_init_*' method, it initializes the paths and loads the CSV file containing image labels.

• In the '*__len__*' method, it returns the length of the dataset.

• In the '*__getitem__*' method, it loads an image and its corresponding label and sketch based on the index.

• Images were resized to 128x128 and converted to PyTorch tensors.

• The *'CustomDataLoader'* class is used to set up and retrieve data loaders for training and testing.

• Image and mask transformations are defined using '*torchvision_transforms*' as *transforms*.

• It provides '*train_dataloader'* and '*test_dataloader'* methods to return data loaders for training and testing datasets.

• It initializes the transformation functions for images and sketches in the *'setup'* method.

• Utilized *Adam optimizer* with a learning rate of 0.001. *Loss function* used in Generator is L1-Loss & BCE (Binary Cross-Entropy) and in Discriminator BCE loss function is used .

• Metrics recorded are Training loss, testing loss, training accuracy, and testing accuracy. Evaluation Metrics are FID (Frechet Inception Distance score) and Inception Score (IS) for binary classification.

1.2. Preprocessing

1.2.1 Image and Sketch Transformation:

- Images and sketches are resized to a fixed size using *'transforms.Resize'*.

- Both images and sketches are converted to tensors using *'transforms.ToTensor'.*

- Normalization is applied to both images and sketches using *'transforms.Normalize'* with mean and standard deviation values.

1.2.2 Handling Labels :

- The CSV file contains labels for each image in one-hot encoded format.

- The *'find_class'* function is used to find the class index from the one-hot vector.

1.2.3 Data Loader Initialization :

- The data loader is initialized with appropriate batch size, shuffle, and number of workers for parallel data loading.

1.2.4 Displaying Sampling Images :

- Functions like *'display_image'* are defined to display sample images from the dataset.

1.3. Hyperparameters

• The **Batch size** taken as **32** which is the number of training samples utilized in one iteration.

• **'Learning rate' = 0.001'** : The rate at which model's parameters are updated during training.

• **'num_epochs = 100'**: The number of times the entire dataset is passed through the model during training.

• Set **'num_workers'** to **2**, determining the count of parallel processes engaged in loading data during both training and validation phases.

• **'number_of_classes = 7'**: The number of classes used the classification task.

• **'optimizer'** used here is *Adam,* it adapts the learning rate for each parameter based on estimates of the first and second moments of the gradients.

1.4. Device Selection

• Determines whether to use GPU ('cuda') or CPU ('cpu') for training based on the availability of a GPU.

1.5. Model Architecture

1.5.1 Generator :

- *Purpose*: The generator takes input sketches and generates corresponding realistic images.

- *Architecture*: The generator architecture typically consists of convolutional layers followed by batch normalization and ReLU activation functions. The architecture used in this code may vary, but it often resembles a U-Net architecture, which includes encoder and decoder pathways connected by skip connections.

- *Input*: Sketches or low-resolution images are provided as input to the generator.

- *Output:* The generator outputs high-resolution images that are expected to resemble the real images corresponding to the input sketches.

1.5.2 Discriminator :

- *Purpose*: The discriminator evaluates the realism of the generated images.

- Architecture: The discriminator is typically a convolutional neural network that takes images as input and outputs a probability indicating whether the input image is real or fake. It learns to discriminate between real images from the dataset and fake images generated by the generator.
- *Training*: The discriminator is trained to maximize the probability of assigning correct labels to real and generated images.

- *Output*: The discriminator outputs a single scalar value indicating the probability that the input image is real.

• Combining Network : (cGAN)

- In the combined network architecture, the generator and discriminator are connected in parallel.

- The generator takes input sketches and generates images, while the discriminator evaluates the realism of these generated images.

- During training, the generator's parameters are updated to minimize the discriminator's ability to distinguish between real and generated images, while the discriminator's parameters are updated to better distinguish between them.

- This adversarial training process leads to the generator producing increasingly realistic images over time, as it learns to generate images that are more difficult for the discriminator to classify as fake.

1.6. Training Setup

1.6.1 Loss Function :

- **Generator Loss**: Typically, the generator is trained to minimize a combination of two losses:

i. *Adversarial Loss*: Measures how well the generator fools the discriminator. It's computed based on the discriminator's output when fed with generated images.

ii. *Content Loss*: Measures the difference between the generated images and the corresponding real images. Commonly calculated using metrics like mean squared error or perceptual loss.

- **Discriminator Loss**: The discriminator aims to correctly classify real and generated images. Its loss is composed of:

i. *Real Loss*: Measures how well the discriminator correctly classifies real images as real.

ii. *Fake Loss*: Measures how well the discriminator correctly classifies generated images as fake.

1.6.2 Generator Training :

i. Feed a batch of sketches into the generator to generate corresponding images.

ii. Compute the generator loss based on the discriminator's output when fed with the generated images and the corresponding real images.

iii. Update the generator's parameters to minimize this loss.

1.6.3 Discriminator Training :

i. Sample a batch of real images from the dataset and a batch of generated images from the generator.

ii. Compute the discriminator loss based on the real and generated images.

iii. Update the discriminator's parameters to minimize this loss.

1.6.4 Visualization (WandB Logger) :

i. Configure the logger parameter to *wandbLogger*, facilitating experiment tracking through Weights & Biases (W&B).

ii. Define the project parameter, specifying the name of the project in W&B, and utilize the entity parameter to indicate the username or team name associated with W&B.

1.6.5 Training Epoch :

i. The code iterates over a specified number of epochs (*num_epochs_custom*) for training the model.

ii. Monitoring metrics such as generator and discriminator losses, as well as visual inspection of generated images, helps determine when to stop training.

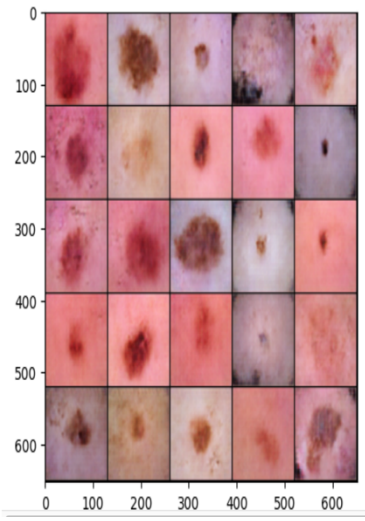
2. Experimental Result

- Average FID (Fréchet inception distance) Score - 560
- Average Inception Score - 1.008

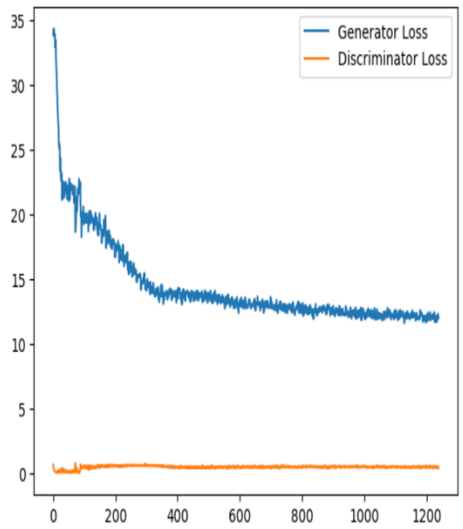
(Graph attached below)

Figure 1: Epoch-wise Data of Generator and Discriminator Loss:

```
100%|██████████| 282/282 [01:07<00:00,  4.15it/s]
  0%|          | 0/282 [00:00<?, ?it/s]/tmp/ipykernel_34/2379273028.py:16: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer key
  label = torch.tensor(self.data_frame.iloc[idx, 1:], dtype=torch.float32)
/tmp/ipykernel_34/2379273028.py:16: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (con
  label = torch.tensor(self.data_frame.iloc[idx, 1:], dtype=torch.float32)
 10%|▉         | 29/282 [00:07<00:55,  4.53it/s]wandb: WARNING (User provided step: 97 is less than current step: 27602. Dropping entry: {'generator_loss_per_epoch': 13.86318302154541
100%|██████████| 282/282 [01:05<00:00,  4.32it/s]
  0%|          | 0/282 [00:00<?, ?it/s]/tmp/ipykernel_34/2379273028.py:16: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer key
  label = torch.tensor(self.data_frame.iloc[idx, 1:], dtype=torch.float32)
/tmp/ipykernel_34/2379273028.py:16: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (con
  label = torch.tensor(self.data_frame.iloc[idx, 1:], dtype=torch.float32)
  2%||         | 7/282 [00:02<01:08,  3.99it/s]wandb: WARNING (User provided step: 98 is less than current step: 27602. Dropping entry: {'generator_loss_per_epoch': 11.051658630371094
 29%|██▉       | 81/282 [00:18<00:43,  4.60it/s]Step 28000: Generator loss: 11.940014839172363, discriminator loss: 0.4563300907611847
```




```
wandb: WARNING (User provided step: 87 is less than current step: 24802. Dropping entry: {'inception_score': 4.452444076538086, '_timestamp': 1713063261.3160262}).
```

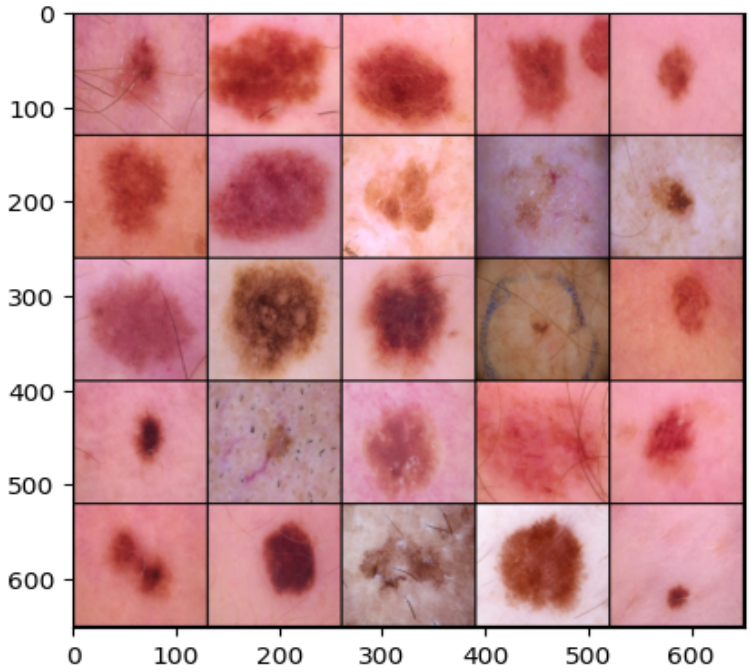



```
100%|██████████| 282/282 [01:10<00:00,  4.01it/s]
  0%|          | 0/282 [00:00<?, ?it/s]/tmp/ipykernel_34/2379273028.py:16: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer key
  label = torch.tensor(self.data_frame.iloc[idx, 1:], dtype=torch.float32)
/tmp/ipykernel_34/2379273028.py:16: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (con
  label = torch.tensor(self.data_frame.iloc[idx, 1:], dtype=torch.float32)
 18%|█▊        | 50/282 [00:11<00:52,  4.40it/s]wandb: WARNING (User provided step: 87 is less than current step: 24802. Dropping entry: {'generator_loss_per_epoch': 11.31342601776123
100%|██████████| 282/282 [01:06<00:00,  4.27it/s]
  0%|          | 0/282 [00:00<?, ?it/s]/tmp/ipykernel_34/2379273028.py:16: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer key
  label = torch.tensor(self.data_frame.iloc[idx, 1:], dtype=torch.float32)
/tmp/ipykernel_34/2379273028.py:16: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (con
  label = torch.tensor(self.data_frame.iloc[idx, 1:], dtype=torch.float32)
  5%|▍         | 15/282 [00:05<01:38,  2.71it/s]wandb: WARNING (User provided step: 88 is less than current step: 24802. Dropping entry: {'generator_loss_per_epoch': 10.59132194519043
 36%|███▌      | 101/282 [00:27<00:57,  3.16it/s]Step 25200: Generator loss: 11.991521835327148, discriminator loss: 0.46658259630203247
```

2.2. Fake and its Corresponding Generated Images

a). During Preliminary Stage of Training:

Figure 2.1: Real Image



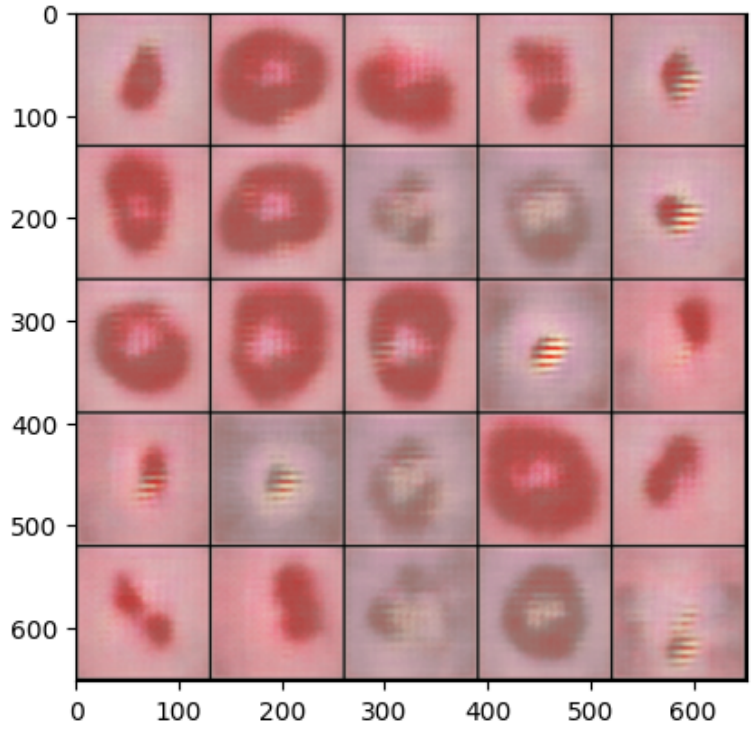


Figure 2.2: Fake Generated Image

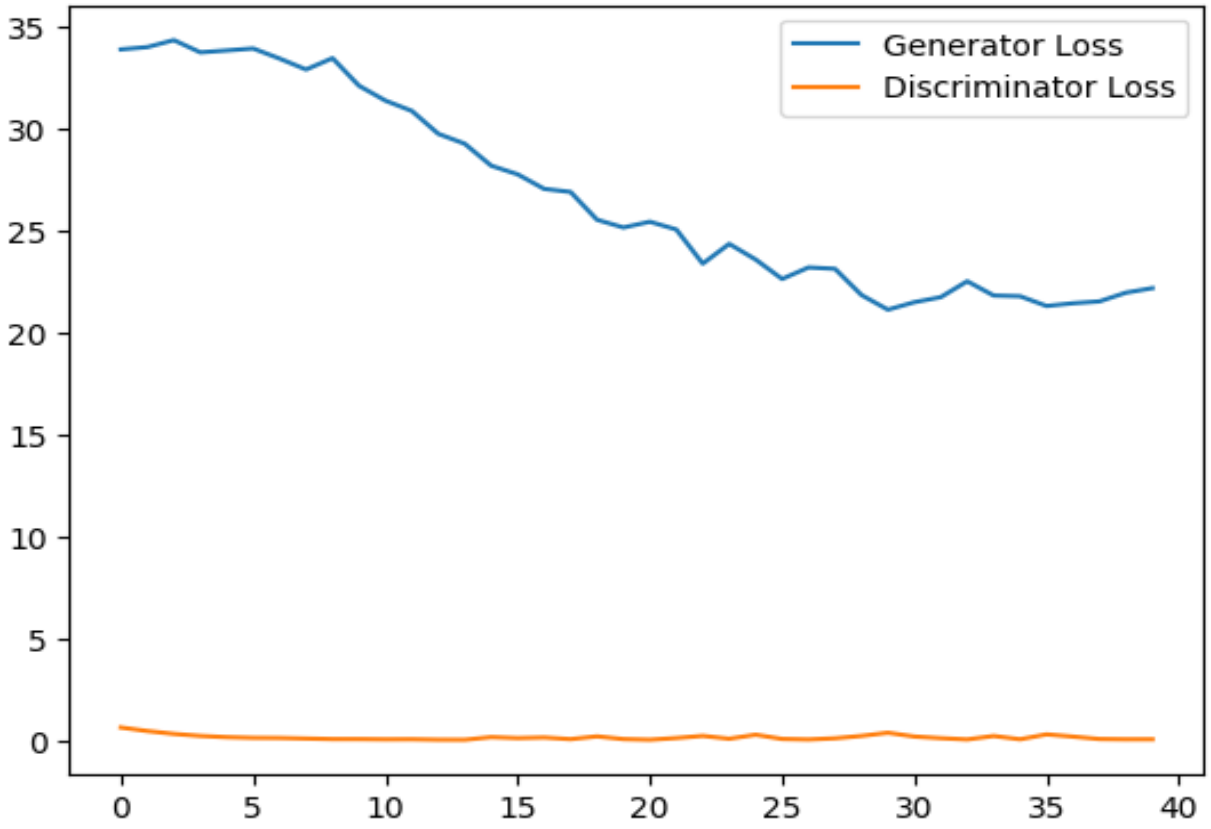


Figure 2.3: Generator & Discriminator Loss Graph

b). During Final Stage of Training: (100th Epoch)

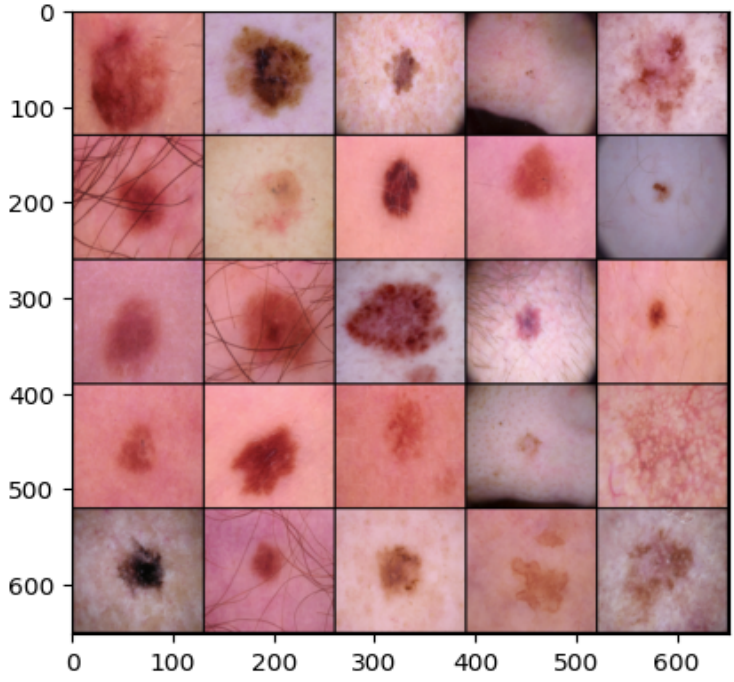


Figure 2.4: Real Image

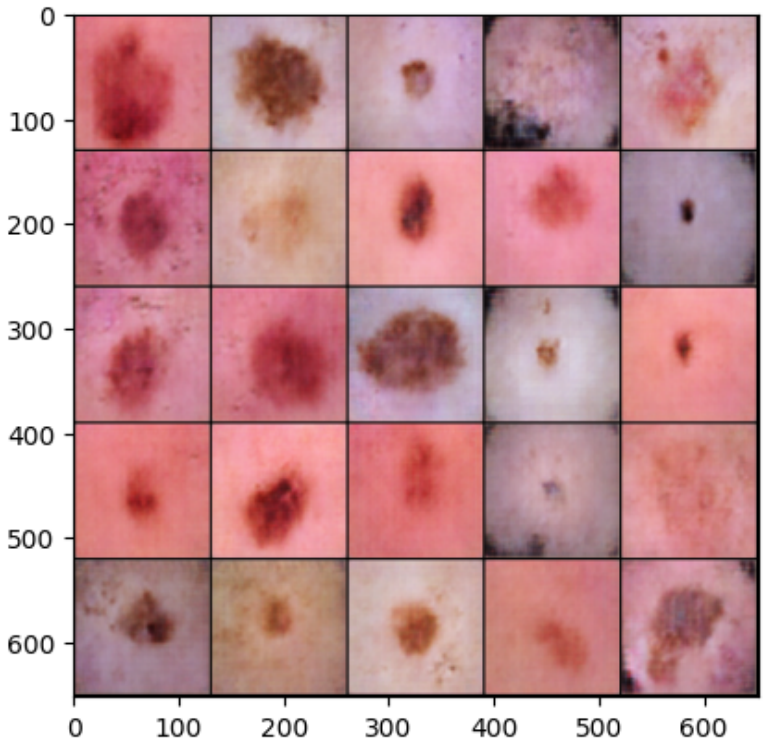


Figure 2.5: Fake Generated Image

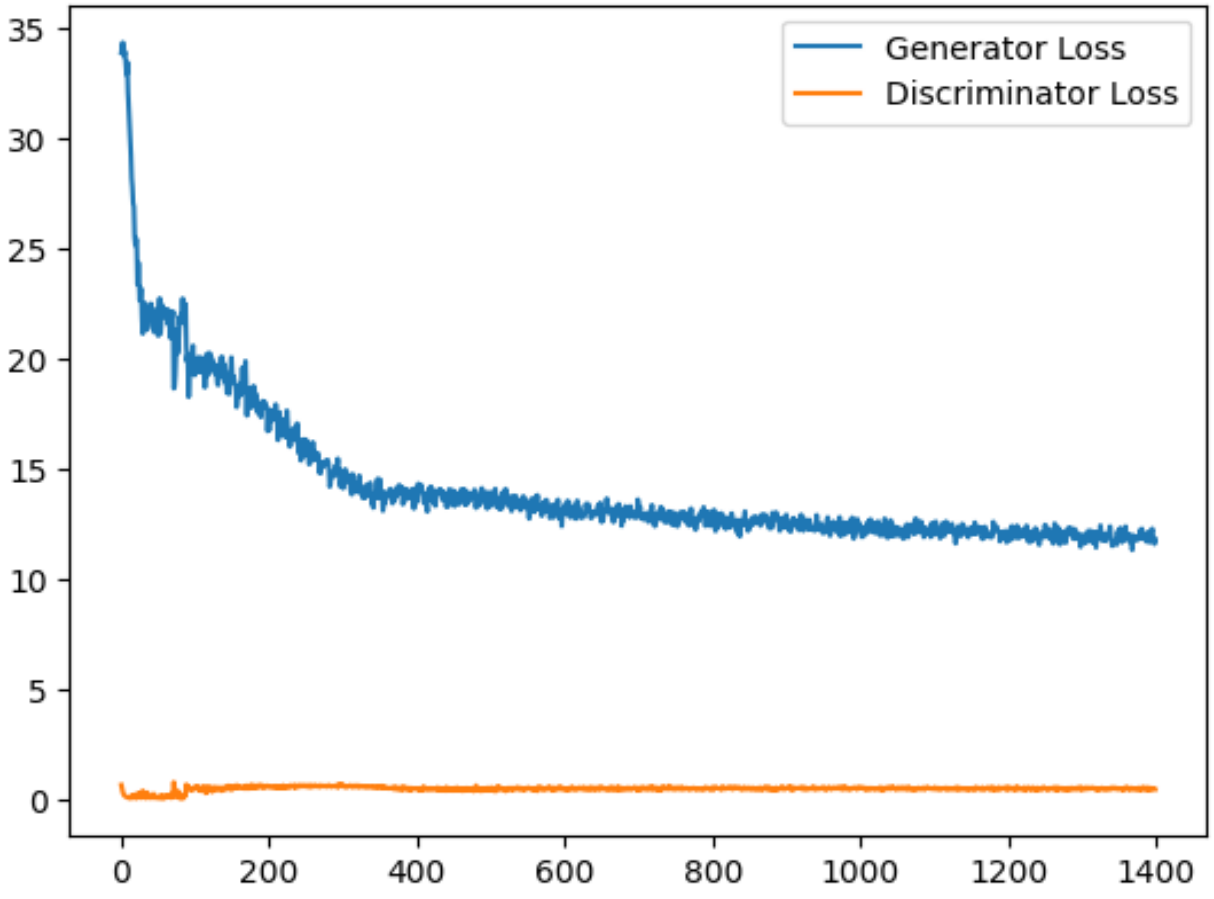


Figure 2.6: Generator & Discriminator Loss Graph

2.3. WandB Logger and Plots:

Figure 1: Generator Loss per epoch graph

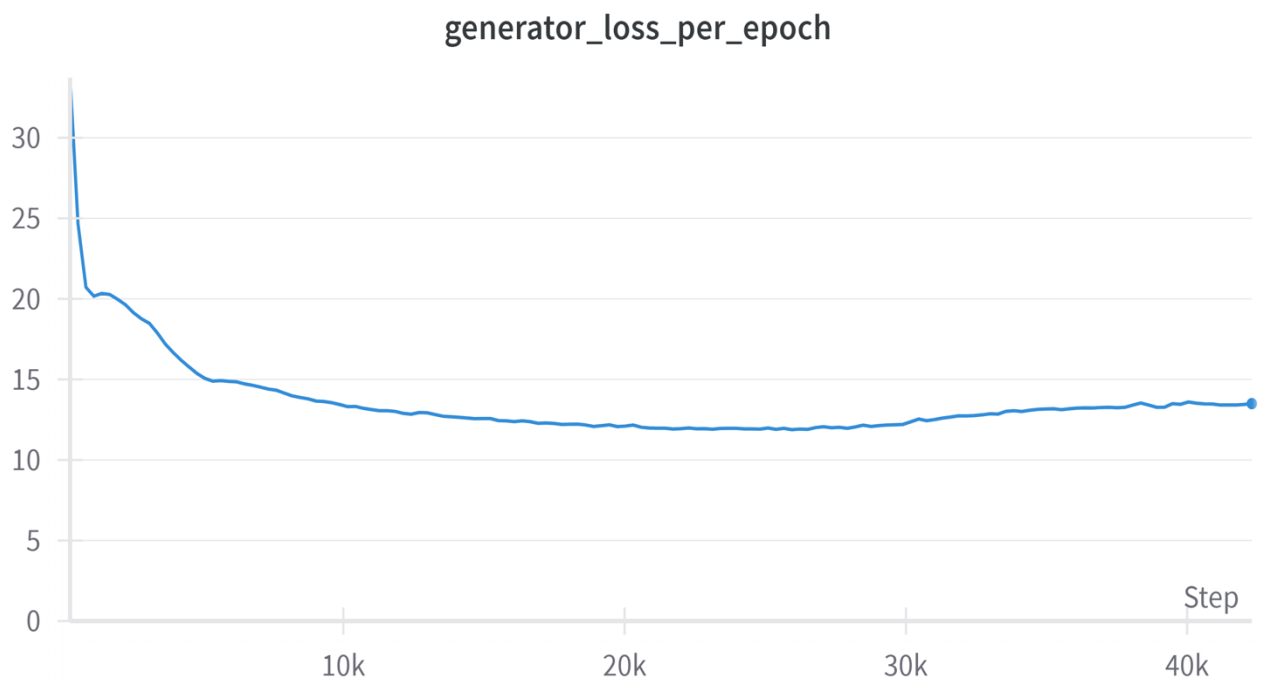


Figure 2: Discriminator Loss per epoch graph

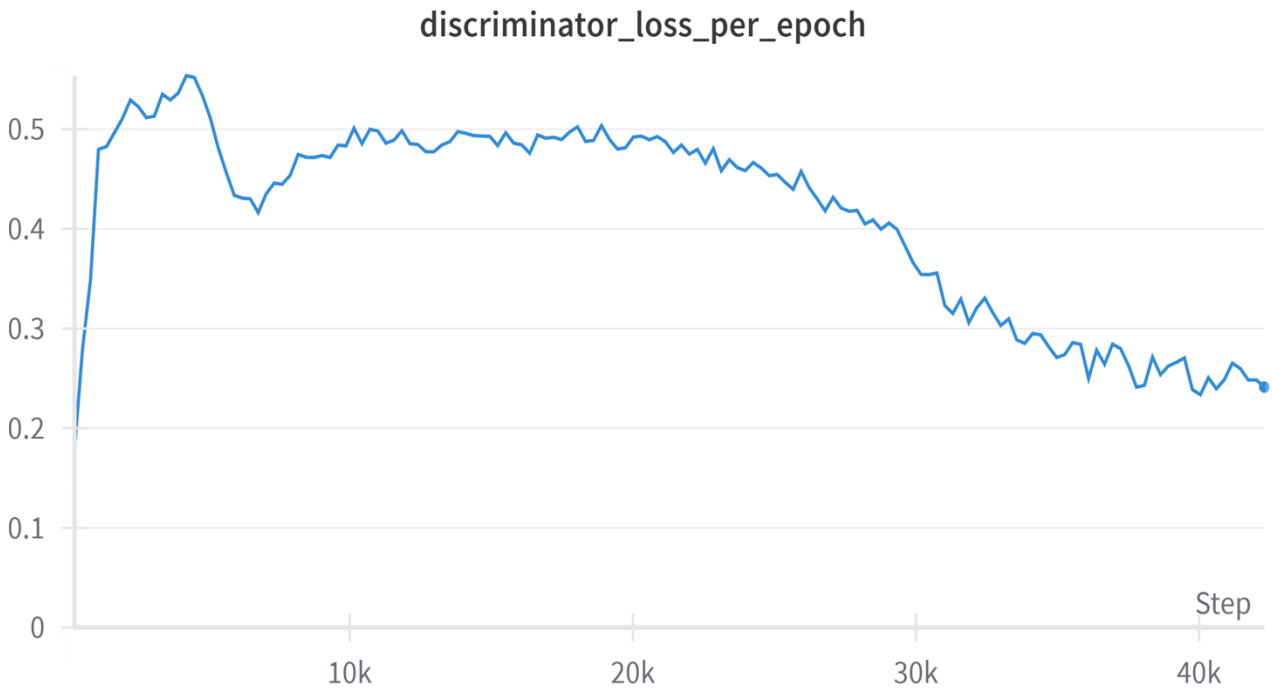


Figure 3: Generator Loss per step graph

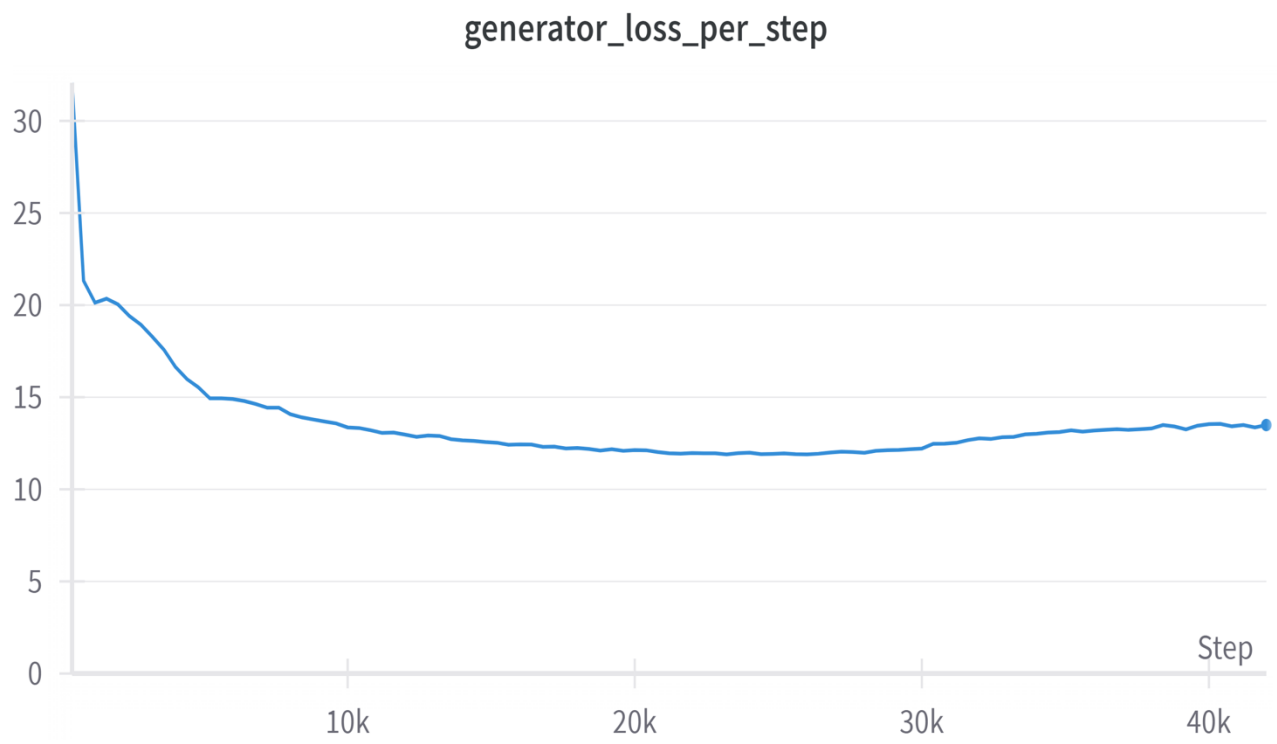


Figure 4: Discriminator Loss per step graph

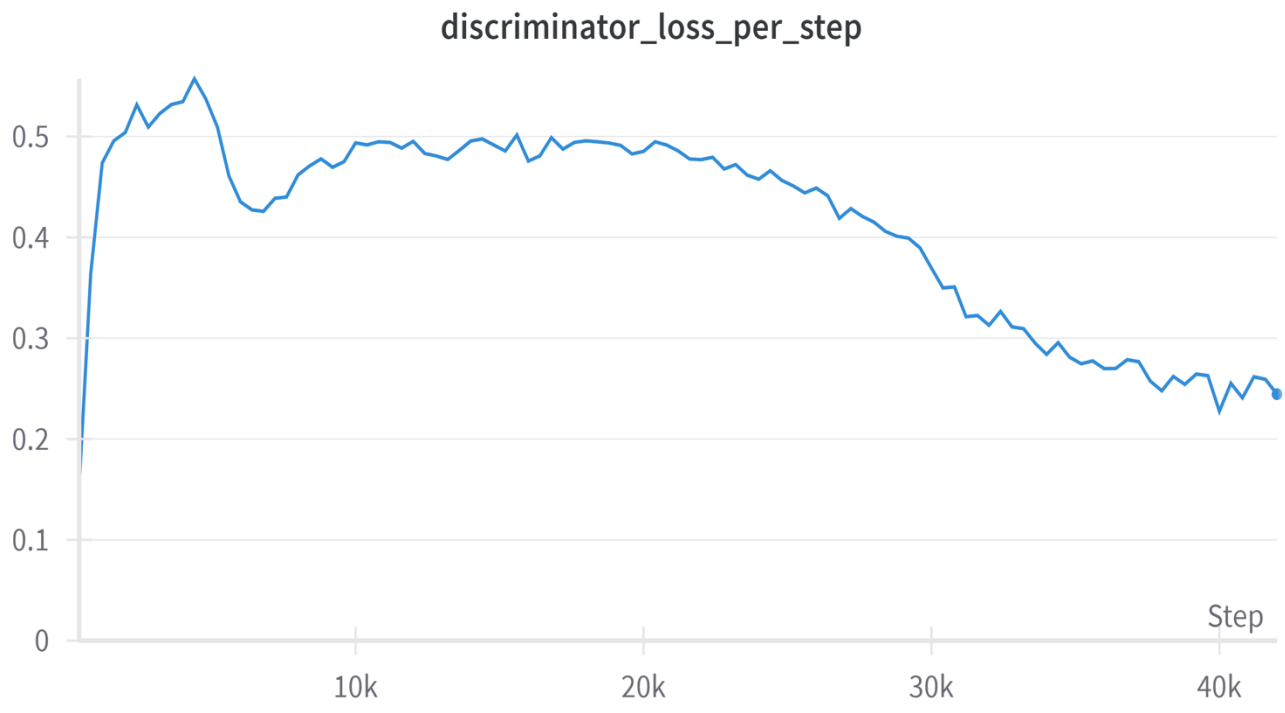


Figure 5: FID (Fréchet Inception Distance) Score graph

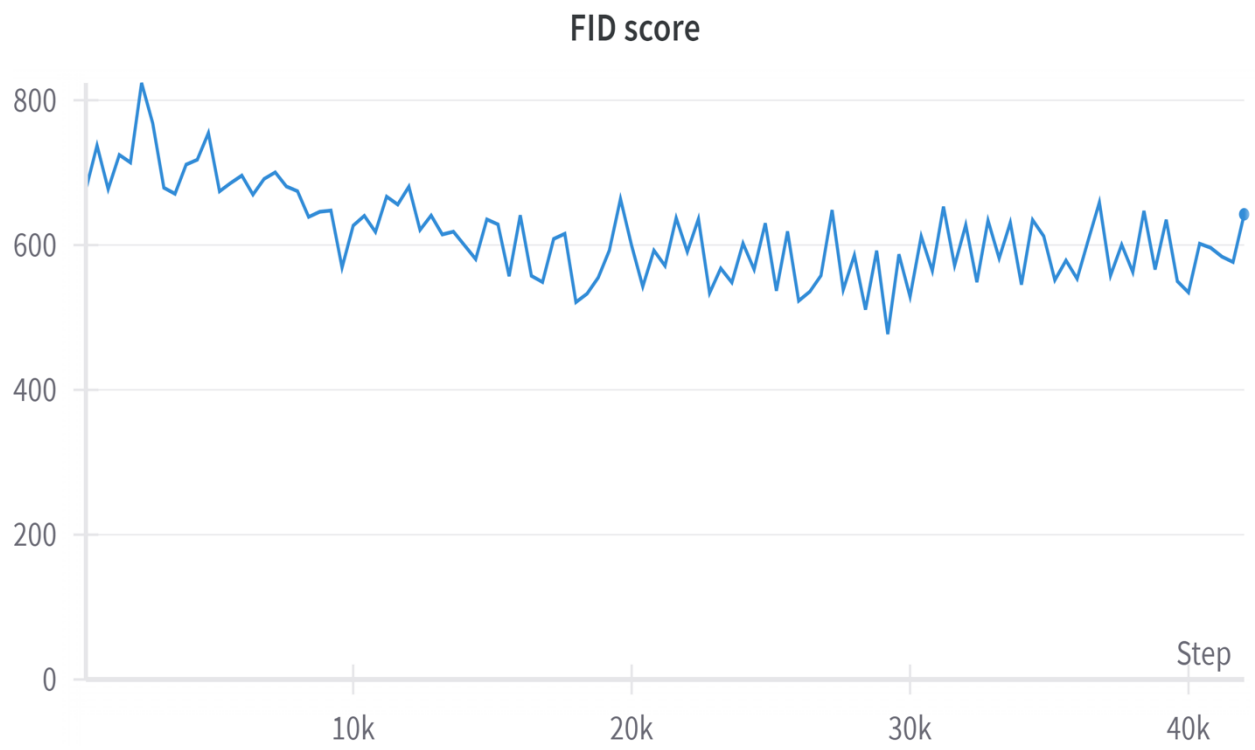
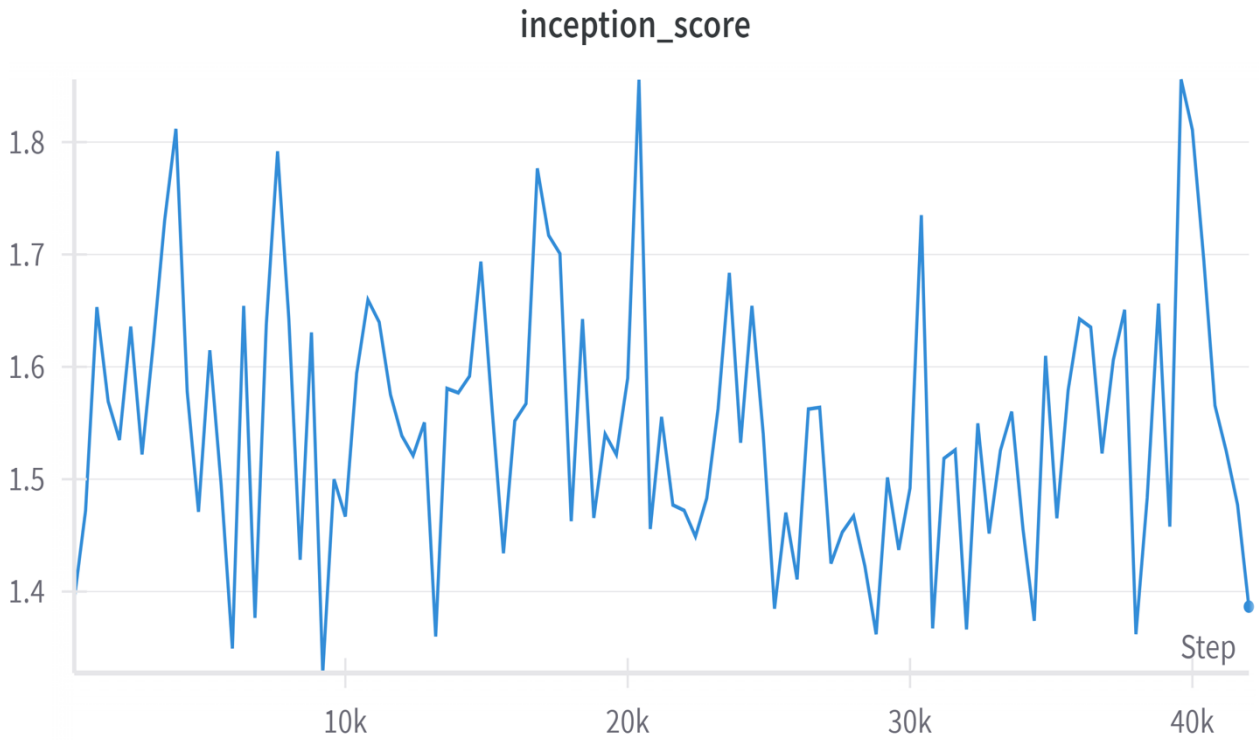


Figure 6: Inception Score (IS) graph



Objective – 2 :

Evaluating the generated images (at least 10 for each class) by considering them as test images and report the accuracy. Compare the accuracy with original test set images of the dataset. For this, you can train any classifier of your choice with the training data available to you.

1. Methodology

1.1 Model Architecture

1.1.1 CNN Classifier :

- *Purpose*: The CNN classifier is used to classify the generated images into different classes.

- *Architecture*: It is based on the ResNet50 architecture, a widely used convolutional neural network (CNN) architecture.

- *Training*: The classifier is pre-trained on a large dataset (e.g., ImageNet) and then fine-tuned on the dataset specific to this task.

- *Output*: The output of the classifier is a probability distribution over the different classes.

1.1.2 Generator :

- *Purpose*: The generator takes input sketches and generates corresponding realistic images.

- *Architecture*: The generator architecture typically consists of convolutional layers followed by batch normalization and ReLU activation functions. The architecture used in this code may vary, but it often resembles a U-Net architecture, which includes encoder and decoder pathways connected by skip connections.

- *Input*: Sketches or low-resolution images are provided as input to the generator.

- *Output:* The generator outputs high-resolution images that are expected to resemble the real images corresponding to the input sketches.

1.1.3 Discriminator :

- *Purpose*: The discriminator evaluates the realism of the generated images.

- Architecture: The discriminator is typically a convolutional neural network that takes images as input and outputs a probability indicating whether the input image is real or fake. It learns to discriminate between real images from the dataset and fake images generated by the generator.

- *Training*: The discriminator is trained to maximize the probability of assigning correct labels to real and generated images.

- *Output*: The discriminator outputs a single scalar value indicating the probability that the input image is real.

• Combining Network : (cGAN)

- In the combined network architecture, the generator and discriminator are connected in parallel.

- The generator takes input sketches and generates images, while the discriminator evaluates the realism of these generated images.

- During training, the generator's parameters are updated to minimize the discriminator's ability to distinguish between real and generated images, while the discriminator's parameters are updated to better distinguish between them.

- This adversarial training process leads to the generator producing increasingly realistic images over time, as it learns to generate images that are more difficult for the discriminator to classify as fake.

1.2. Training Setup

1.2.1 Loss Function :

- **Generator Loss**: Typically, the generator is trained to minimize a combination of two losses:

    i. *Adversarial Loss*: Measures how well the generator fools the discriminator. It's computed based on the discriminator's output when fed with generated images.

    ii. *Content Loss*: Measures the difference between the generated images and the corresponding real images. Commonly calculated using metrics like mean squared error or perceptual loss.

- **Discriminator Loss**: The discriminator aims to correctly classify real and generated images. Its loss is composed of:

    i. *Real Loss*: Measures how well the discriminator correctly classifies real images as real.

    ii. *Fake Loss*: Measures how well the discriminator correctly classifies generated images as fake.

1.2.3 Generator Training :

Feed a batch of sketches into the generator to generate corresponding images.

1. Compute the generator loss based on the discriminator's output when fed with the generated images and the corresponding real images.

2. Update the generator's parameters to minimize this loss.

1.2.4 Discriminator Training :

i. Sample a batch of real images from the dataset and a batch of generated images from the generator.

ii. Compute the discriminator loss based on the real and generated images.

iii. Update the discriminator's parameters to minimize this loss.

1.2.5 Visualization (WandB Logger) :

i. Configure the logger parameter to *wandb Logger*, facilitating experiment tracking through Weights & Biases (W&B).

ii. Define the project parameter, specifying the name of the project in W&B, and utilize the entity parameter to indicate the username or team name associated with W&B.

1.2.6 Training Epoch :

i. The code iterates over a specified number of epochs (*num_epochs_custom*) for training the model.

ii. Monitoring metrics such as generator and discriminator losses, as well as visual inspection of generated images, helps determine when to stop training.

1.3. Report on Evaluation

- The classification report furnishes an intricate overview of the model's effectiveness concerning each class within a multi-class classification scenario.

i. **Frechet Inception Distance (FID)**: It is commonly used to evaluate the quality and diversity of generated images. It measures the similarity between the distribution of real images and generated images in feature space extracted by a pre-trained Inception model.

ii. **Inception Score (IS):** It evaluates the quality and diversity of generated images. It measures the KL divergence between the conditional class distribution and the marginal class distribution of generated images.

1.4. Evaluation Setup

- **Metric Selection:** Frechet Inception Distance (FID) and Inception Score (IS).

- **Interpretation and Analysis:**

  i. We assess the classifier's effectiveness using the original test set images.

  ii. Every image undergoes classification by the classifier, and the predicted class is compared against the actual label.

  iii. The classification accuracy is determined by the proportion of accurately classified images relative to the total number of images in the test set.

  iv. The evaluation metrics provide quantitative measures of the quality and diversity of generated images.

  v. Results are interpreted and analysed to assess the performance of the generator model.

vi. Discrepancies between different evaluation metrics may indicate specific strengths or weaknesses of the model

2. Experimental Results

• We evaluate the classifier's performance in categorizing synthetic images produced by the cGAN.

• To ensure a comprehensive assessment, we choose a minimum of 10 generated images per class.

• Similar to the evaluation process with the original test set, we classify each generated image and compute the accuracy accordingly.

• Accuracy of Classifier : 0.702857142855

2.1. WandB Logger and Plots:

Figure 1: Training Loss of Classifier graph

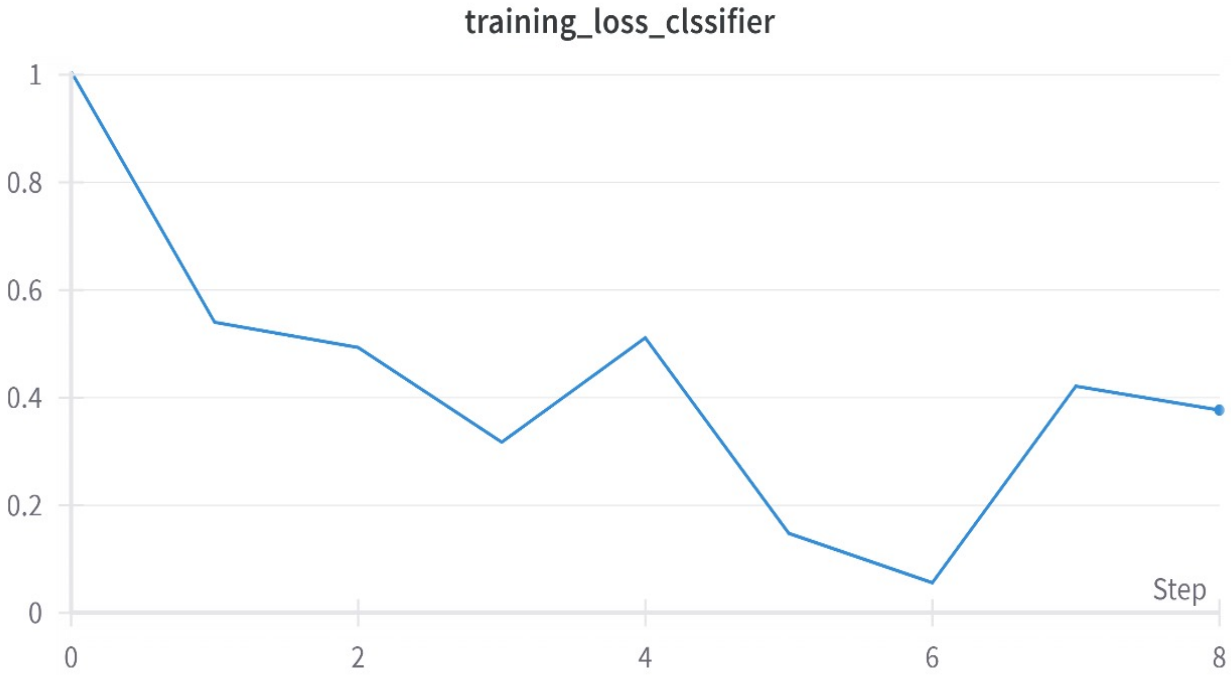


Figure 2: Training Accuracy of Classifier graph

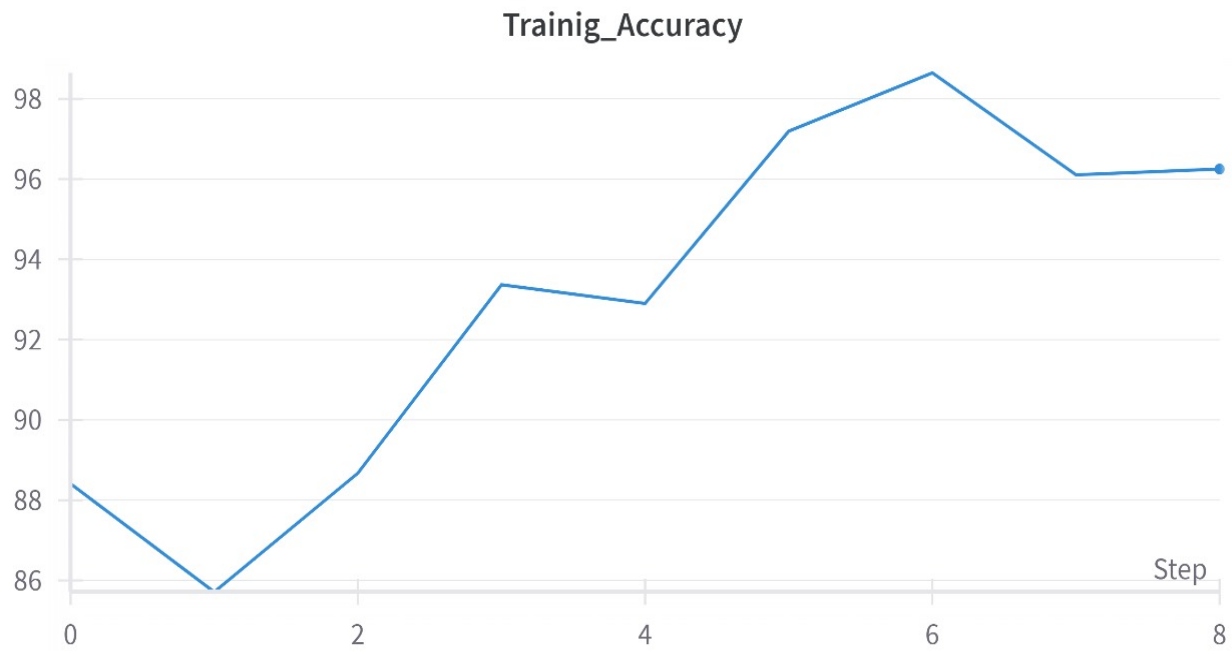


Figure 3: Test Accuracy of Classifier graph

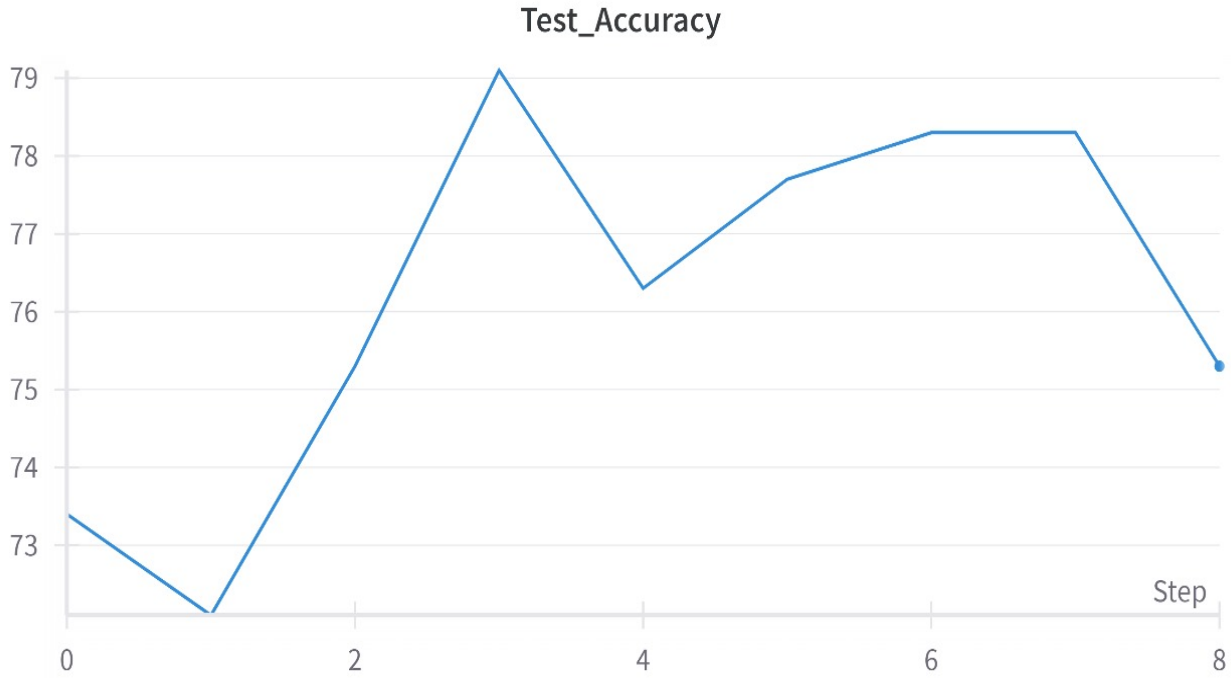


Figure 4: FID (Fréchet Inception Distance) Score graph

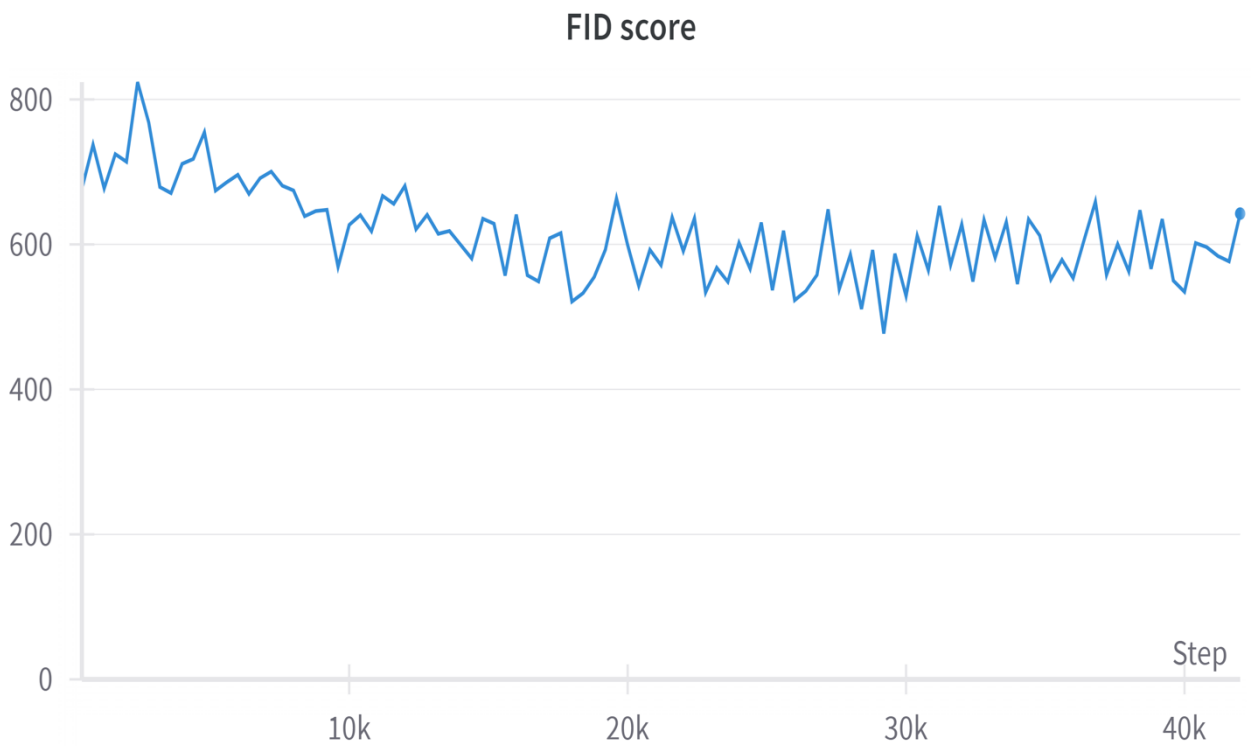
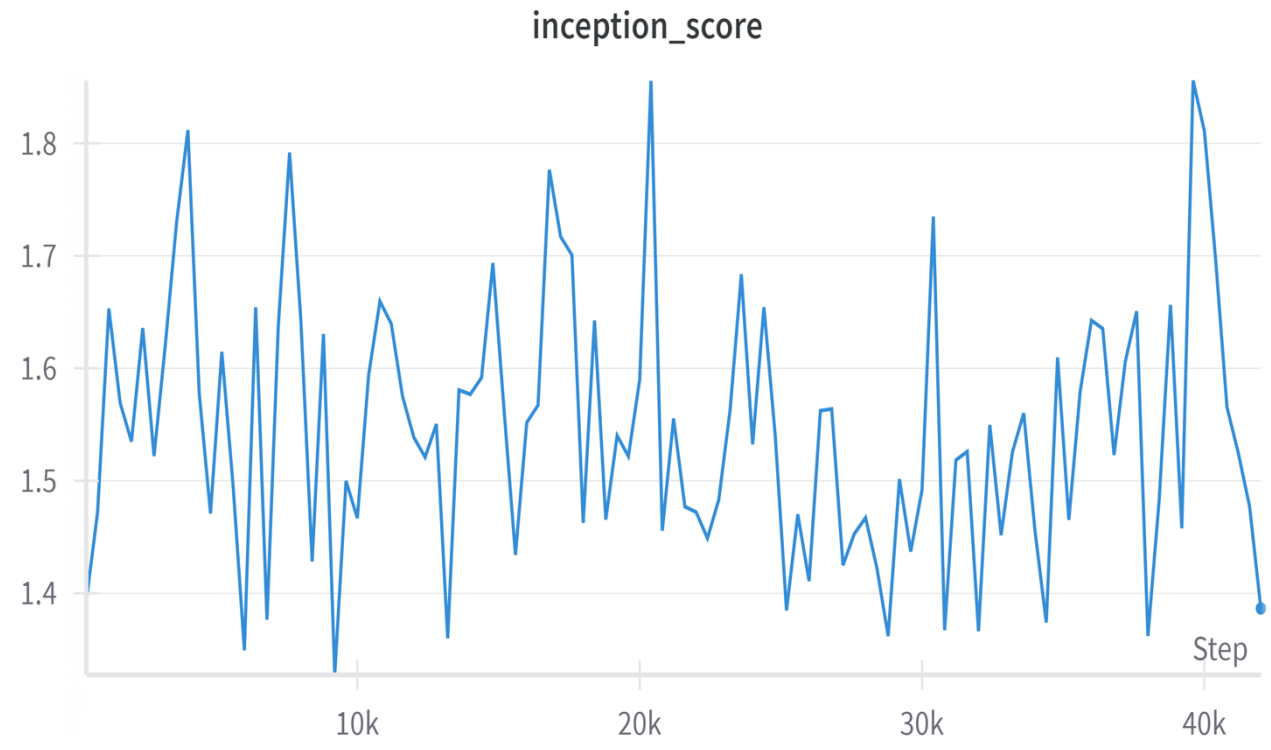


Figure 5: Inception Score (IS) graph



2.2 Evaluation Metrics

Metric	Score
FID	560
Inception Score	1.008

2.3 Classwise Images Data

Class	No. of Data
MEL	887
NV	6130
BCC	480
ALIEC	317
BKL	972
DF	101
VASC	128

2.4 Classwise Accuracy

Class	Accuracy (%)
MEL	20.03
NV	96.875
BCC	11.032
ALIEC	3.094
BKL	5.09
DF	7.98
VASC	6.98

discriminator_loss_per_epoch	0.02426
discriminator_loss_per_step	0.45633
generator_loss_per_epoch	26.76892
generator_loss_per_step	11.94001

Figure 2.4.1: Losses per epoch

3. Observations for both tasks

3.1. For First Task:

i. Enhancing the image generation model's performance can be achieved through the adoption of a more sophisticated architecture such as the U-Net architecture for the generator and leveraging the VGG pretrained architecture for the discriminator.

ii. A notable increase in the FID score signals a substantial dissimilarity between the distribution of real images and that of generated images, indicative of a deficiency in realism or fidelity within the generated images.

iii. A lower Inception Score implies a moderate degree of diversity observed within the generated images, serving as an empirical observation derived from experimentation.

3.2 For Second Task:

i. In the analysis of our data, we discern a notable variance in accuracy, indicative of bias within the dataset.

ii. Our model exhibits suboptimal performance in accuracy for select images, a phenomenon attributed to data scarcity within specific subsets.

4. Result and Analysis

i. *Generator Loss*: Decreases over time as the generator improves in generating realistic images.

ii. *Discriminator Loss*: Initially high as the discriminator learns to distinguish between real and fake images. It decreases as the generator improves.

iii. *Inception Score*: Indicates the quality and diversity of generated images. Higher scores indicate better quality and diversity (Here we got: 1.008).

iv. *FID Score*: Indicates how similar the distribution of generated images is to the distribution of real images. Lower scores indicate better similarity (Here we got: 560).

v. *Accuracy*: Classification accuracy of the CNN classifier on the generated images (Here we got: 0.70285).

5. Conclusion

- The GAN successfully learns to generate realistic images from sketches, as evidenced by the quality of generated images and low FID scores.
- The classifier is trained to accurately classify the generated images, indicating that the generator produces images that are representative of different classes.
- The Inception Score provides additional insight into the diversity and quality of the generated images.
- Overall, the GAN training process achieves the desired outcome of generating realistic images from sketches and demonstrates the effectiveness of the proposed approach.

References:

- Referance-1
- Referance-2

----------End of Report----------