Deep Learning Report

Assignment - 5

Ayush Abrol

B20AI052

Question 01

Aim:

- Train a DCGAN to generate images from noise. Use the EMNIST(Extended MNIST) database to learn the GAN Network.
    - VGG11 as a discriminator.
- Perform the following tasks:
    - Uniformly generate ten noise vectors that act as latent representation vectors, and generate the images for these noise vectors, and visualize them at
        - After the first epoch.
        - After n/2 th epoch.
        - After your last epoch. (say n epochs in total)

      and comment on the image interpretation at (i), (ii) and (iii) and can you identify the images?

    - Plot generator and discriminator losses for all the iterations. Also display the best-generated images by the model.[One iteration = forward pass of a mini-batch]

Procedure:

- FIrstly, I did all the necessary imports, set up my device to GPU and setup the following configuration:

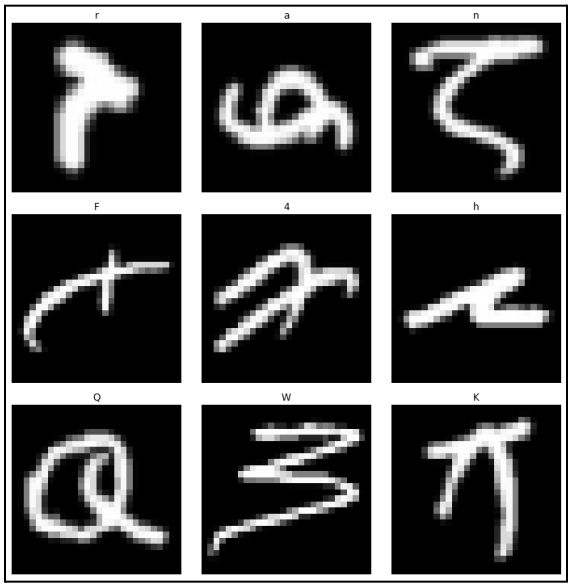
```
config = {
    'batch_size': 128,
    'num_epochs': 30,
    'num_classes': 47,
    'learning_rate': 0.0001,
    'momentum': 0.9,
    'latent_dimension': 100,
    'hidden_dimension': 256,
```

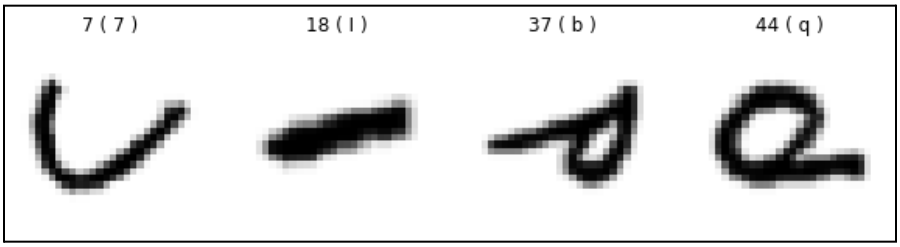
```
    'betas_generator': (0.5, 0.999),
    'betas_discriminator': (0.5, 0.999),
    'num_workers': 2,
    'img_shape': None,
}
```

- Downloaded the EMNIST dataset using torchvision.datasets and created train and test loaders.
- Shape of the data:

```
torch.Size([112800, 28, 28])
torch.Size([112800])
torch.Size([18800, 28, 28])
torch.Size([18800])
```

- Number of classes are 47, including 0-9, A-Z and a-t.
- And images are of size 1x28x28.
- Visualization of the dataset:





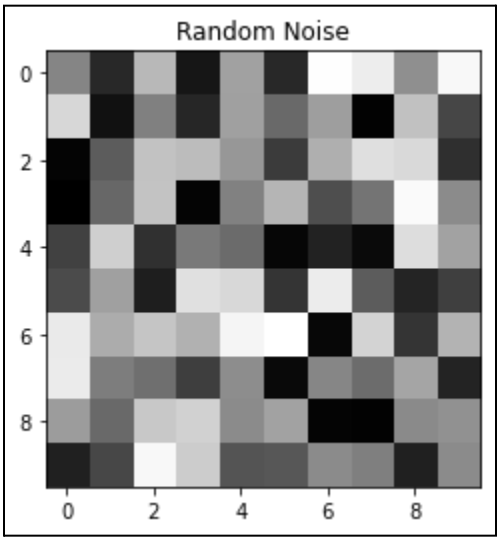
Generator Class

- The Generator class inherits from the nn.Module class, which is the base class for all neural network models in PyTorch.
- The constructor (init method) of the Generator class takes a configuration dictionary as an argument that contains hyperparameters for the model, such as the dimension of the input noise vector (latent_dimension).
- The model architecture consists of a fully connected layer (fc) that maps the input noise vector to a 256 x 7 x 7 tensor, followed by four transposed convolutional layers (trans_conv1-4) that gradually upsample the tensor to the desired output image size.
- Each transposed convolutional layer is followed by a batch normalization layer (trans_conv1-3_bn) and a rectified linear unit (ReLU) activation function (F.relu).
- The last transposed convolutional layer outputs a 1-channel image, and the output is normalized using the hyperbolic tangent function (torch.tanh) to ensure that the pixel values are between -1 and 1.
- The forward method of the Generator class defines the forward pass of the model, which takes the input noise vector (x) as an argument and returns the generated image tensor. The method first applies the fully connected layer, reshapes the output to a 4D tensor, and then applies the four transposed convolutional layers with batch normalization and ReLU activations.
- Finally, the output is normalized using the hyperbolic tangent function and returned as the generated image tensor.

Discriminator Class

- We start by loading a pre-trained VGG11 model from the torchvision.models library and modify its classifier layer to output a single value for binary classification (real/fake).
- The Discriminator class is defined as a subclass of torch.nn.Module and takes the modified VGG11 model as an argument to its constructor.
- The Discriminator architecture consists of four convolutional layers (conv0-3) followed by a fully connected layer (fc) that maps the final feature map to a single output value.
- Each convolutional layer is followed by a LeakyReLU activation function (F.leaky_relu) with a different negative slope value, and a 2D dropout layer (conv0-3_drop) with a dropout probability of 0.25 to prevent overfitting.
- The final output is normalized using the sigmoid function (torch.sigmoid) to ensure that the output value is between 0 and 1, representing the probability of the input being real.
- The forward method of the Discriminator class defines the forward pass of the model, which takes the input image tensor (x) as an argument and returns the discriminator output as a probability value.
- The method first reshapes the input tensor to a 4D tensor, applies the convolutional layers with LeakyReLU activations and dropout layers, and then applies the fully connected layer to output a single probability value.
- The num_flat_features method is defined to calculate the number of features in the final feature map before the fully connected layer, which is used in the forward method to properly reshape the tensor.

- After this, I instantiated both the models (generator and discriminator) and set up their loss functions and optimizers using the config defined earlier.
- Then, I created a random noise vector generator function which takes in the dimension that we want the latent vector to be in.



(10x10 noise) reshaped from (1x100 vector)

Training

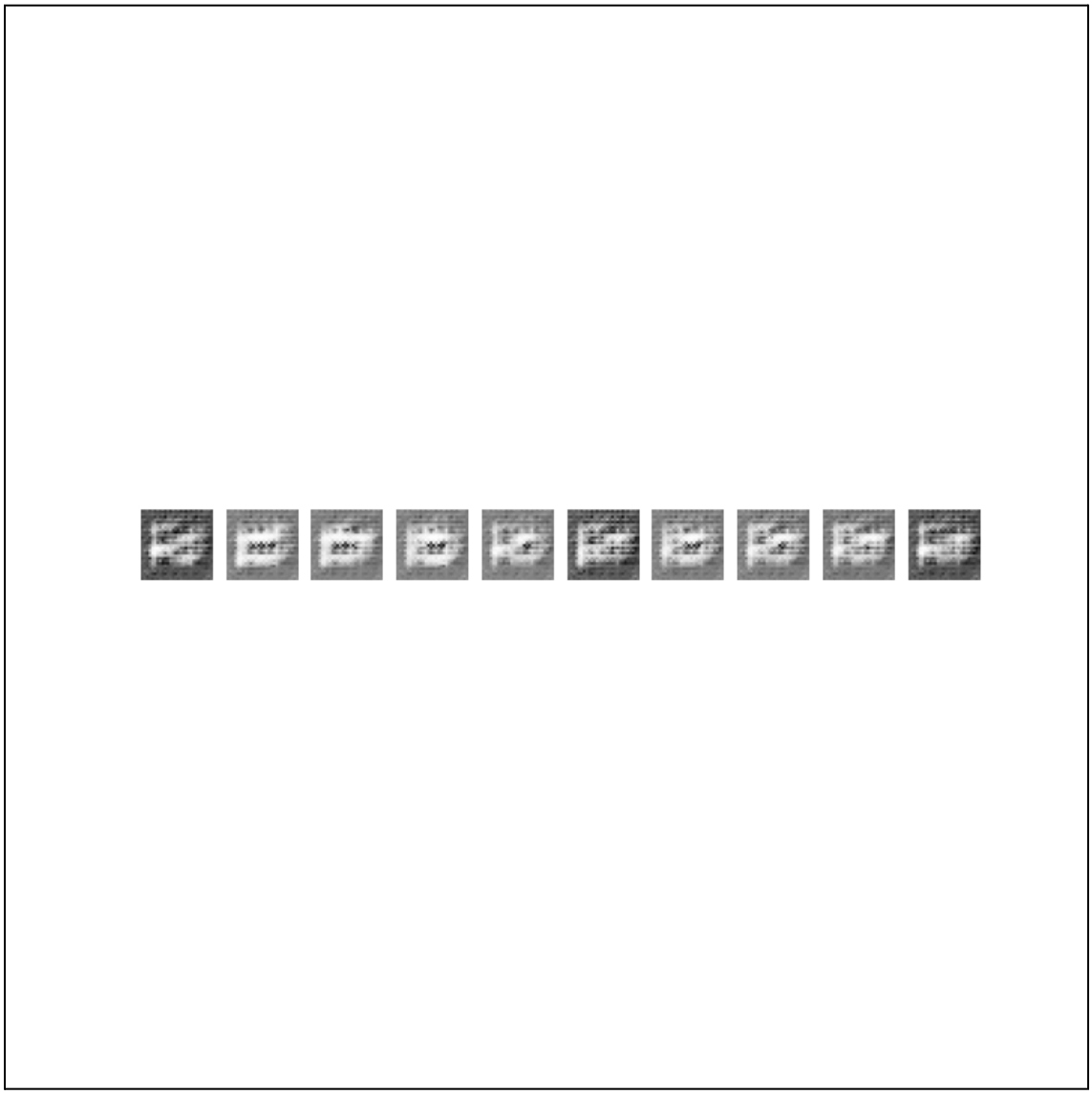
- The EMNIST dataset is loaded using DataLoaders defined, and the. The generator network takes a random noise vector as input and generates an image. The discriminator network takes an image as input and outputs a probability that the image is real.
- Then I defined the loss functions for the generator and discriminator networks. The generator loss is the binary cross-entropy loss between the output of the discriminator network and a vector of ones. The discriminator loss is the binary cross-entropy loss between the output of the discriminator network for the real images and a vector of ones, and the output of the discriminator network for the fake images generated by the generator and a vector of zeros.
- Then defined the optimizers for the generator and discriminator networks. The Adam optimizer is used for both networks.

- The main training loop is then defined, which trains the generator and discriminator networks alternately. For each epoch, the code loops through the batches of the EMNIST dataset and trains the networks on each batch. The generator loss and discriminator loss are calculated for each batch, and the overall losses are appended to lists. The code also saves the generator and discriminator models every five epochs and generates 10 images for the first epoch, the middle epoch, and the last epoch.
- Finally, the code prints the total training time and indicates that the training is finished.

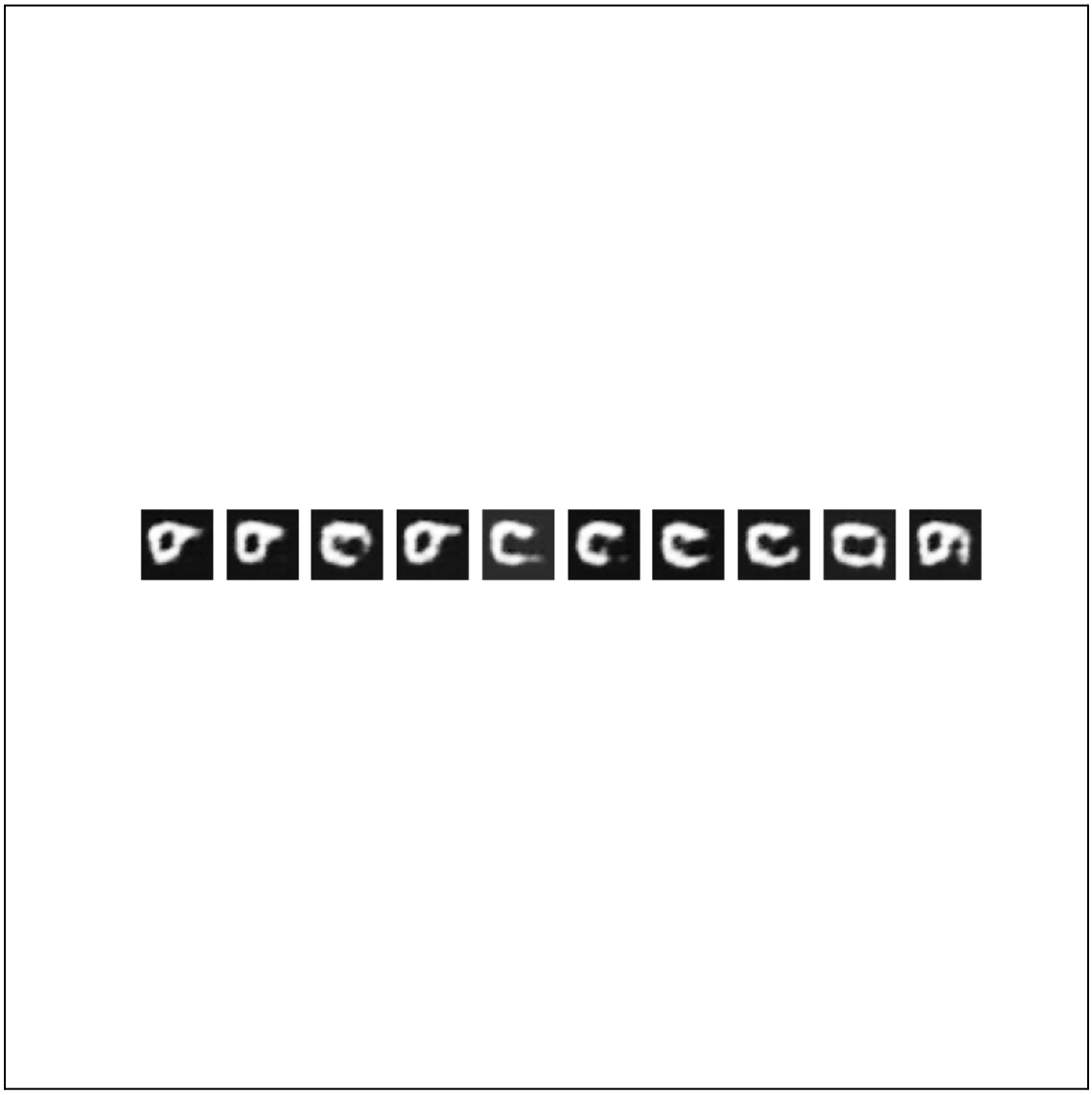
```
Epoch [30/30] Batch 1/882             Loss D: 0.0002, loss G: 14.7777
Epoch [30/30] Batch 101/882             Loss D: 0.0377, loss G: 12.6397
Epoch [30/30] Batch 201/882             Loss D: 0.0007, loss G: 13.0891
Epoch [30/30] Batch 301/882             Loss D: 0.0085, loss G: 16.5239
Epoch [30/30] Batch 401/882             Loss D: 0.0042, loss G: 15.6910
Epoch [30/30] Batch 501/882             Loss D: 0.0019, loss G: 13.4861
Epoch [30/30] Batch 601/882             Loss D: 0.0000, loss G: 18.0797
Epoch [30/30] Batch 701/882             Loss D: 0.0320, loss G: 9.8773
Epoch [30/30] Batch 801/882             Loss D: 0.0006, loss G: 12.5127
Epoch [30/30] Batch 882/882             Loss D: 0.0427, loss G: 15.2530
Time Elapsed for Epoch 30: 1844.6095629 seconds
Total Time Elapsed: 28885.836192100003 seconds
--------------------------------------------------

Training Finished!
Total Time Elapsed: 28885.836192100003 seconds
```

Results

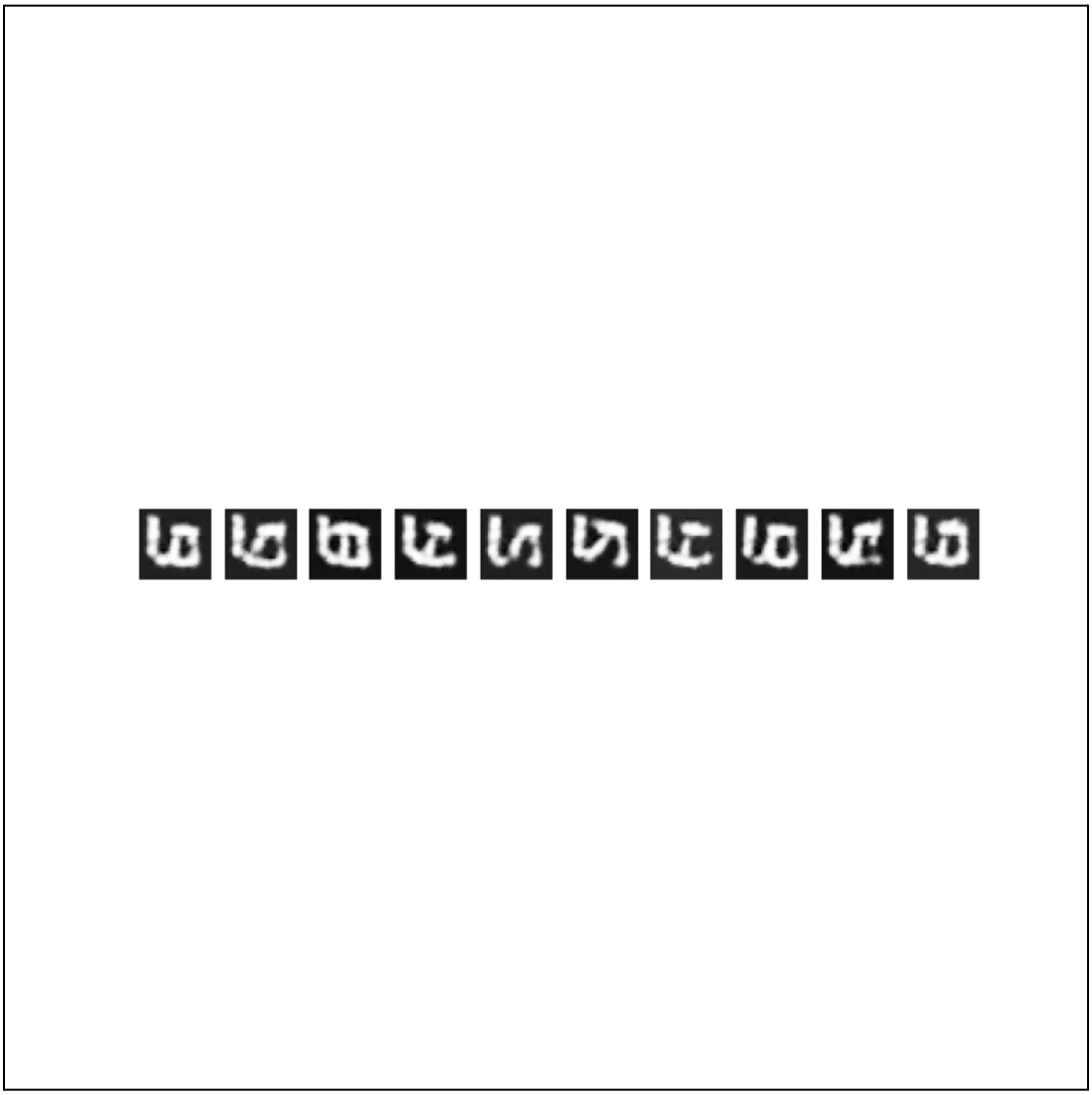
- Images generated by generator after 1st epoch:



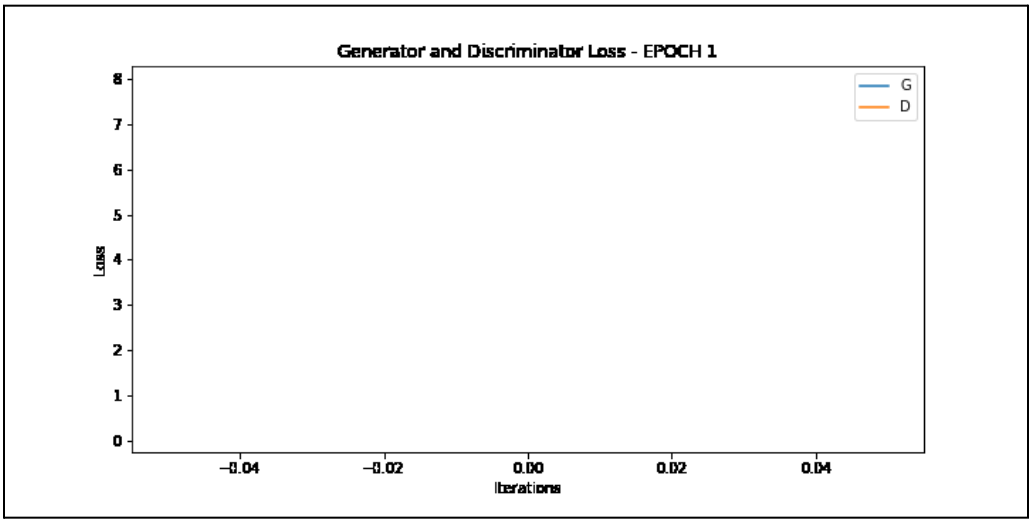
- Images generated after 15 epochs



- Images generated after 30 epochs

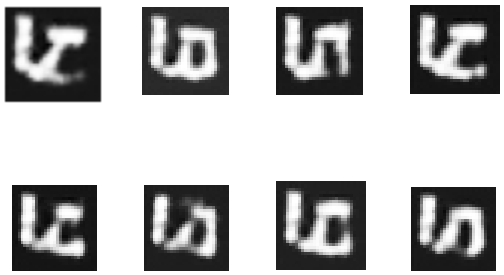


- Then, I created a gif for generator and discriminator losses for all the epochs



(PS: This is GIF and can't be seen moving in a pdf, we can watch this in .ipynb file)

- Displaying the best-generated images by the model



Question 02

Aim:

- Download the pre-trained StyleGan(v1, v2 or v3).
    - Generate 10 realistic images using the StyleGAN.
    - Take your face image and 5 different face images of your friends (One image per friend).
- Perform feature disentanglement and linear interpolation between your face and your friend's face.
- An example is shown below:

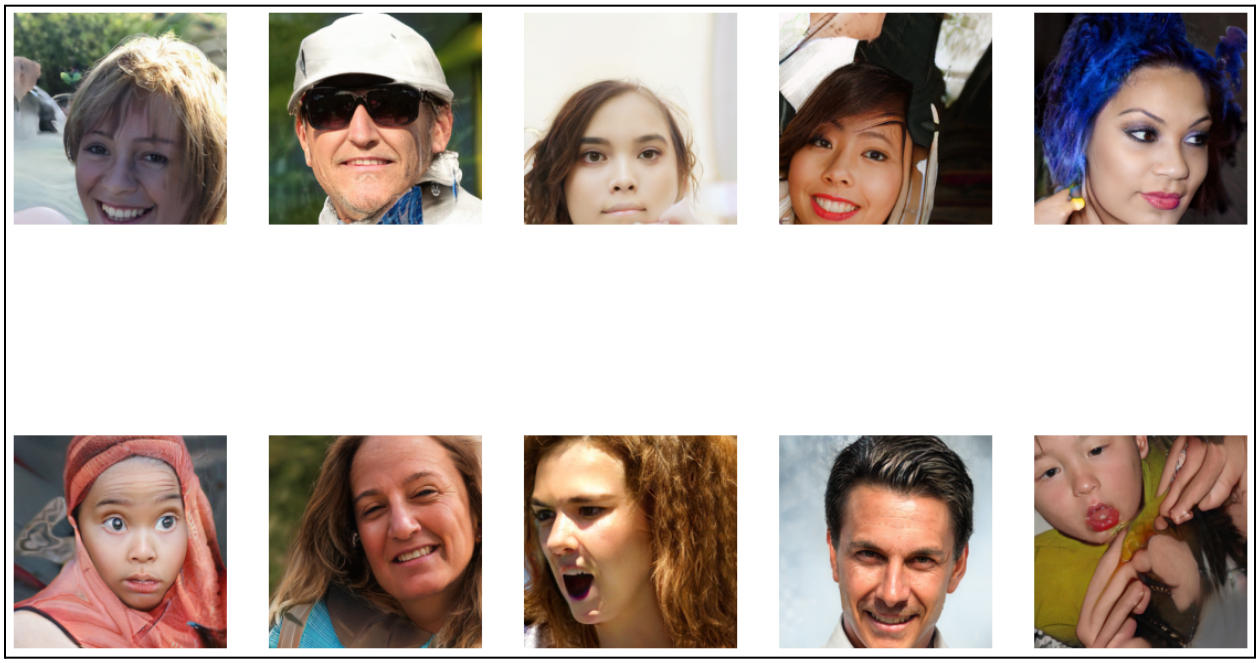
Procedure:

- Set up the device to GPU.
- Cloned the official repository of StyleGAN3 by NVIDIA. (Link)
- Set up the working directory inside the repo.
- Installed the ninja library.
- Ran the command:

```
!python3 ./stylegan3/gen_images.py --outdir=out --trunc=1
--seeds=2-11 --network=stylegan3-r-ffhqu-256x256.pkl
```

This runs the gen_images.py code in the official NVIDIA repo and uses a pre-trained StyleGAN3 model for generating 256x256 images and seeds defined from 2-11 generates 10 different random images and saves them in the out directory.

- VIsualization of the images generated:



For Part 2 of the Question 2, I have created a different .ipynb as I used Google Colab for this part as there were some issues with the versions in my local environment.

- Imported me and my friend's images and resized them to 256x256.

- Installed the dnnlib-util and legacy libraries.
- Then, I defined a function named expand_seed that takes in two parameters:
    - seeds: a list of integers representing the seed values for the random number generator
    - vector_size: an integer representing the size of the vector
- The function loops through each seed value in the seeds list, creates a random number generator using the numpy.random.RandomState() method, and generates a random vector of size vector_size using the randn() method of the random number generator. The resulting vectors are stored in a list named result.
- Then I defined a constant URL containing a URL to download a pre-trained StyleGAN3 model from NVIDIA's NGC cloud service. The code uses the load_network_pkl() method from the legacy module of the DNNLib library to load the pre-trained generator network from the downloaded file, and assigns the resulting generator network to a variable named G.
- Then the vector_size sets the variable to the size of the latent vector used by the generator network, and defines a list of seed values consisting of two integers: 8193 and 8201. The expand_seed() function is then called with the seeds and vector_size arguments to generate a list of random vectors.

Latent space

- Then I started to generate images using a pre-trained StyleGAN3 model. The get_label function creates a one-hot encoded label tensor based on the class index and the number of conditional dimensions of the generator network. The generate_image function generates an image by feeding the given latent code and label tensor to the generator network and returning the resulting image as a PIL.Image object. The truncation_psi and noise_mode parameters are used to control the style and noise used in the generated image.
- The G parameter is the generator network object. The z parameter is the latent code used to generate the image. If class_idx is not None, then the get_label function creates a one-hot encoded label tensor based on the class index and the number of conditional dimensions of the generator network. Finally, the resulting image is converted to a PIL.Image object and returned.

- Then, I cloned the StyleGAN2 repo for using the projection function which creates the latent vectors for the images provided.

```
!git clone https://github.com/NVlabs/stylegan2-ada-pytorch.git
```

- Obtaining latent vectors for all the 3 images.

```
!python3 stylegan2-ada-pytorch/projector.py --outdir=latent/0
--target=images/0.jpg
--network=https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-pytorch/pre
trained/ffhq.pkl
```

```
!python3 stylegan2-ada-pytorch/projector.py --outdir=latent/1
--target=images/1.jpg
--network=https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-pytorch/pre
trained/ffhq.pkl
```

```
!python3 stylegan2-ada-pytorch/projector.py --outdir=latent/2
--target=images/2.jpg
--network=https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-pytorch/pre

trained/ffhq.pkl
```

- Reconstructed images using latent vectors:
- Then, latent vectors created were loaded using numpy and reshaped to (18, 512)
- Then, I created frames for the interpolation video to be created.
- Created the interpolation video using:

```
!ffmpeg -r 30 -i /content/stylegan3/results/frame-%d.png -vcodec
mpeg4 -y movie.mp4
```