Visual Recognition

Assignment-6

May 11th 2019.

Prepared By: **Durga Yasasvi Y, IMT2016060.**

Srujan Swaroop G, IMT2016033.

Siddharth Reddy D, IMT2016037.

Task(1):

- New Faces generation using GAN.

Approach(1):

- **(This code is written in Face_Generation.ipynb file).**
- **Approach and Procedure:**
  - **Used DC_GAN to generate new images.**
  - **It mainly composes of convolutional layers without max pooling or fully connected layers. Following are the things a DCGAN contains:**
    - **It has convolutional strides.**
    - **For upsampling, it uses transposed convolution.**
    - **Fully connected layers are eliminated.**
    - **Uses Batch Normalization multiple times except for the generator output layer and discriminator's input layer.**
    - **Uses ReLU for the generator and LeakyReLU for the discriminator.**

- **Below is the code snippet that describes the structure of generators network:**

```
def generator(self):

    model = Sequential()

    model.add(Dense(128 * 7 * 7, activation="relu", input_dim=100))
    model.add(Reshape((7, 7, 128)))
    model.add(UpSampling2D())

    model.add(Conv2D(128, kernel_size=3, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Activation("relu"))
    model.add(UpSampling2D())

    model.add(Conv2D(64, kernel_size=3, padding="same"))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Activation("relu"))

    model.add(Conv2D(self.channels, kernel_size=3, padding="same"))
    model.add(Activation("tanh"))
```

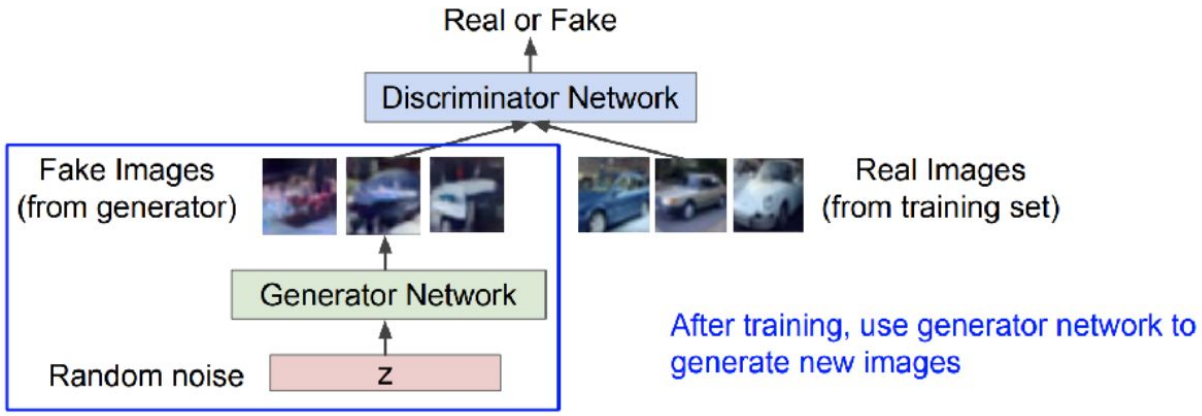
- **Below is the code snippet that describes the structure of the discriminator's network:**

```
def discriminator(self):

    model = Sequential()

    model.add(Conv2D(32, kernel_size = 3, strides = 2, input_shape = (28, 28, 1), padding="same"))
    model.add(LeakyReLU(alpha = 0.2))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, kernel_size = 3, strides = 2, padding = "same"))
    model.add(ZeroPadding2D(padding = ((0, 1), (0, 1))))
    model.add(BatchNormalization(momentum = 0.8))
    model.add(LeakyReLU(alpha = 0.2))
    model.add(Dropout(0.25))

    model.add(Conv2D(128, kernel_size = 3, strides = 2, padding="same"))
    model.add(BatchNormalization(momentum = 0.8))
    model.add(LeakyReLU(alpha = 0.2))
    model.add(Dropout(0.25))

    model.add(Conv2D(256, kernel_size = 3, strides = 1, padding = "same"))
    model.add(BatchNormalization(momentum = 0.8))
    model.add(LeakyReLU(alpha = 0.2))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(1, activation = 'sigmoid'))
```

- **The loss function for the DC GAN is calculated as given below:**

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

- **Whereas the D(X) is called the Discriminator net and G(Z) is the Generator net. X is the real data and the Z is the fake data.**
- **The first net, Generator net generates the data. And the second net, Discriminator net classifies the input as either fake or real.**
- **At the optimal point, the generator net will model the real data and the discriminator net will output the probability with 0.5 as not leading to differentiate between real data and the generated data.**
- **Let's look at how the GAN looks like at a higher level:**

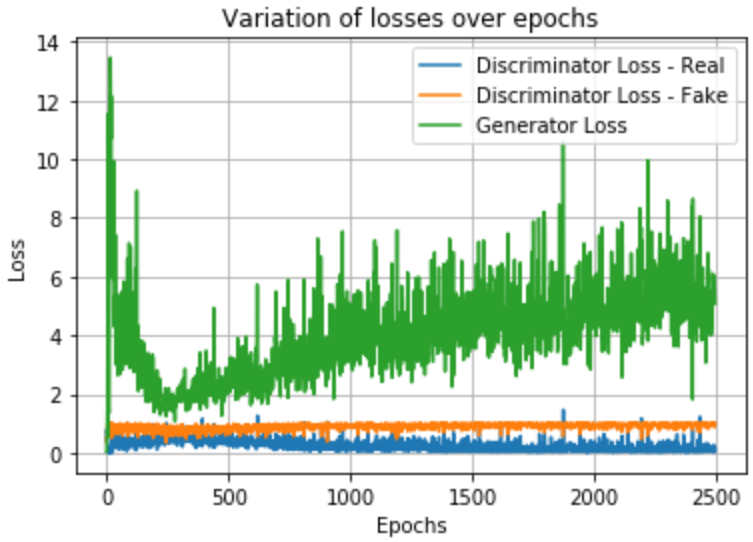


- **Hyperparameters:**
  - **Epochs: 2500.**
  - **Batch_size: 32.**
  - **Input shape: 28 x 28 x 3.**

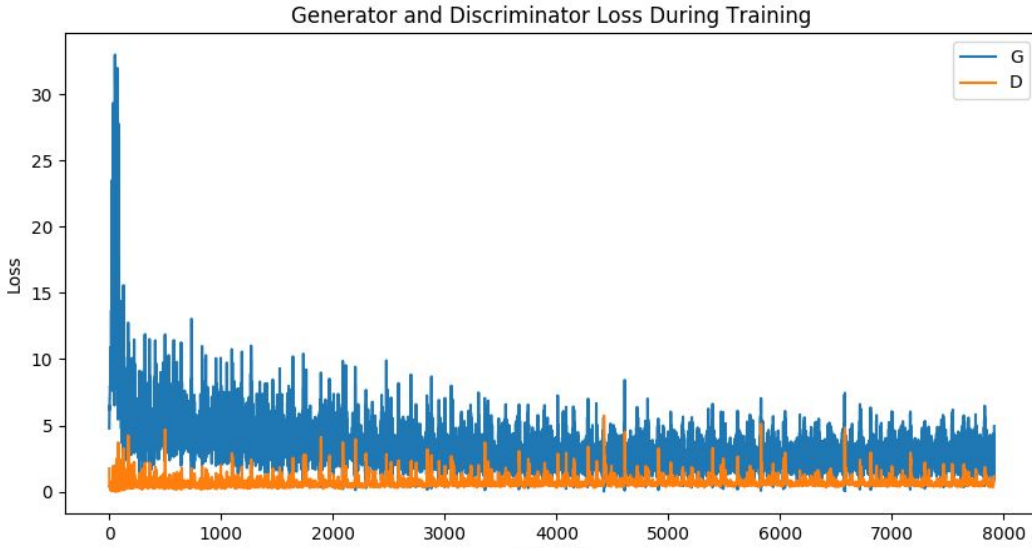
- **Let's look at the results obtained for the different number of epoch values (250, 500, 2000 and 2500 respectively).**




- **Below is the loss curve obtained for this DCGAN:**



- **For different hyperparameters like,**
  - **Epochs = 8000.**
  - **Batch size = 128.**
  - **Input shape = 64 x 64 x 3.**
  - **Learning rate = 0.0002**
- **The Loss curve is obtained as given below:**



- **And the generated images obtained looks like:**

- **Observations:**
    - **As you increase the number of epochs and the input image quality, the better are the generated images obtained.**

Task(2):

- Real vs Fake Face Verification.

Approach(2):

- **(This code is written in Face_Verification.ipynb file).**

- **For every image in the dataset, we used pre-trained FaceNet weights to generate image vector encodings.**
- **And there are 42 different face classes.**
- **Training and Testing Data preparation:**
    - **Randomly considered two images from each face class and generated image vector encodings for both the images as mentioned above.**
    - **Having these two images under training data, we labelled number 0 as the testing data.**
    - **Similarly, for the two images of different face classes, we labelled number 1 as the testing data.**
    - **0 denotes that the two images of the training data are similar face classes. 1 denotes that the two images of the training data are different face classes.**
    - **This process is repeated for a sufficient amount of data.**
- **Then on this data, applied SVM to classify whether the two images belong to the same face class or not.**
- **Below are the results obtained:**

```
from sklearn.svm import SVC # "Support Vector Classifier"
clf = SVC(kernel='linear')
```

```
clf.fit(X_train, y_train)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

```
accuracy = clf.score(X_test, y_test)
```

```
print('Acuuracy Score: ', accuracy * 100, '%')
```

```
Acuuracy Score:  82.16870342771982 %
```

- **Accuracy Score: ~82%.**
- **Observations:**
  - **Used FaceNet architecture on top. Using DeepFace and DeepID might result in higher accuracies.**

Prepared By:

Durga Yasasvi Y, IMT2016060.

Srujan Swaroop G, IMT2016033.

Siddharth Reddy D, IMT2016037.