Machine Learning Nanodegree Capstone Project

Generating human faces using Deep Convolutional Generative Adversarial Networks

Raj Patel

26/11/2017

Definition

Domain Background

With the help of the recent advances in Machine Learning / Deep Learning and knowledge-based AI, computers are able to replicate complex tasks that are done the by humans, they can recognize and detect objects from images, recognize speech, understand natural language, try to mimic and learn how to play games etc. And they do these tasks with high level of accuracy. But they are still quite far from replacing a human. As humans, we have a special characteristic that is creativity that was believed not to be learned by the computers, but that's not true. With generative models, the idea of making machines to create new things is becoming possible. The generative models can learn to generate data similar to the data that is given to them. The generated data can be useful not only in generative tasks but also in fulling the missing data in some real samples and to get internal representation which may be useful for other machine learning tasks. One of such Generative Models is Generative Adversarial Network.

Generative Adversarial Networks have been regarded as the most prominent models according to some machine learning pioneers. Generative Adversarial Networks also known as GANs were introduced by Ian Goodfellow in 2014 (Goodfellow, et al., 2014). We are going to use a special type of GAN referred as DCGAN or Deep convolutional GAN. The whole idea behind GAN is to have a zero-sum game framework by using two neural networks contesting with each other. One neural network is the generator and the main task of the generator is generating new items the other neural network is a discriminator and the task of the discriminator is to distinguish between real and fake items. The items that would be generated by the generator would be fake items since it is trying to mimic the real data items the main goal of the generator is to make these data items as real as possible so that it can fool the discriminator, on the other hand, the goal of the discriminator is to distinguish these fake these and real items as best as possible so here the discriminator works as an adversary judging the real and the fake items. So at the start, the generator produces some fake data items these fake data items are feed into the discriminator along with the real data items and the discriminator is made to learn which are real and fake. The results of the discriminator are than further used to improve both the generator and itself. Backpropagation is used on both the networks so that so that the generator produces better images, while the discriminator becomes more skilled at flagging data items. (Goodfellow I. , 2016) This process continues indefinitely and in the end, we get two high trained models one that is highly capable of generating new data items and other that is highly capable of distinguishing these data items.

Problem Statement

The goal of this project is to build a DCGAN so that we build a generator can generate images of real looking human faces just by giving it some random noise. The output of the Generator will be an Image with possibly a human looking face on it.

Evaluation Metrics

$$\min_G \max_D V(D, G)$$

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

The training of GAN can be represented as the mathematical function above. {Image src : (SHAIKH, 2017)}

- z : Represents the initial random noise or input given to the generator
- x : Represents the actual data to be fed to the discriminator in our case it would be the real human face images
- G (z): Represents the output of the generator in our case it is the image of a face to be generated. Note: G is a differentiable function which tries to map the input z based on its distribution p_g .
- D(x): Represents the probability that x came from data rather than p_g . Here in our case, the discriminator will give out a probability suggesting how real the given real image is.
- D (G (z)): Represents the probability that how likely the output of the generator is similar to the real data. Here in our case, it will give probability suggesting how real the given fake image looks like.

We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize log (1 – D (G (z))). Error Estimation of both the networks is a form of logloss function. Initially, when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, log (1 - D (G (z))) saturates. Rather than training G to minimize log (1 - D (G (z))) we can train G to maximize log (D (G (z))). In other words, D and G play the following two-player minimax game with value function V (G; D). (Goodfellow, et al., 2014) (Bruner & Deshpande, 2017)
The metrics we will be using is the discriminator loss and generator loss after every epoch saved in numpy's npy format. Ideally, for a GAN, this losses should converge to the same number but that is not possible in a minmax relationship, but after some time of training this losses should look similar. Also, we will visually compare the output of the generator and the actual data along with the test accuracy of the discriminator.

Analysis

Data Overview

CelebFaces Attributes Dataset (CelebA) is a large-scale face attributes dataset with more than 200K celebrity images, each with 40 attribute annotations. The images in this dataset cover large pose variations and background clutter (Tang, Liu, Luo, Wang, & Xiaoou, 2015). CelebA has large diversities, large quantities, and rich annotations, including

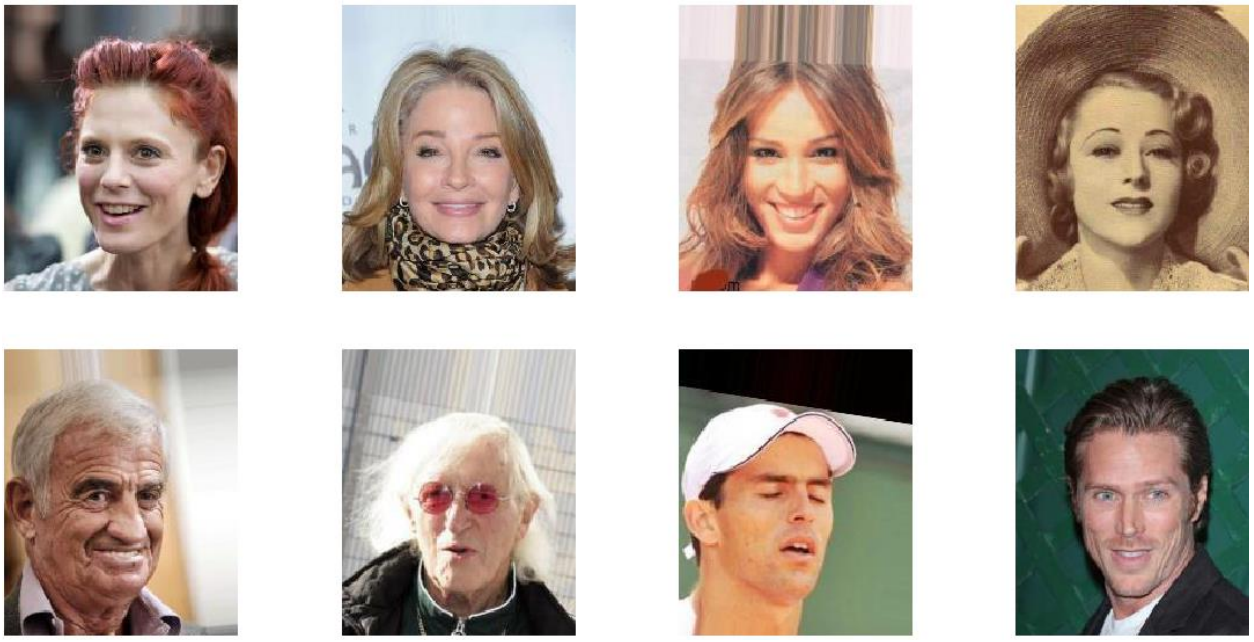
1. 10,177 number of identities,
2. 202,599 number of face images, and
3. 5 landmark locations, 40 binary attributes annotations per image.

Each of the 200k raw images is of same heights and widths, all are RGB images i.e. there are comprised of three color channels Red, Green, Blue. For the context of this project, we will be only using the raw images for the training phase other details like the identities, landmarks and the annotations will be irrelevant. Since the images are of 172 X 218 size some kind of preprocessing will be required to reduce their size.

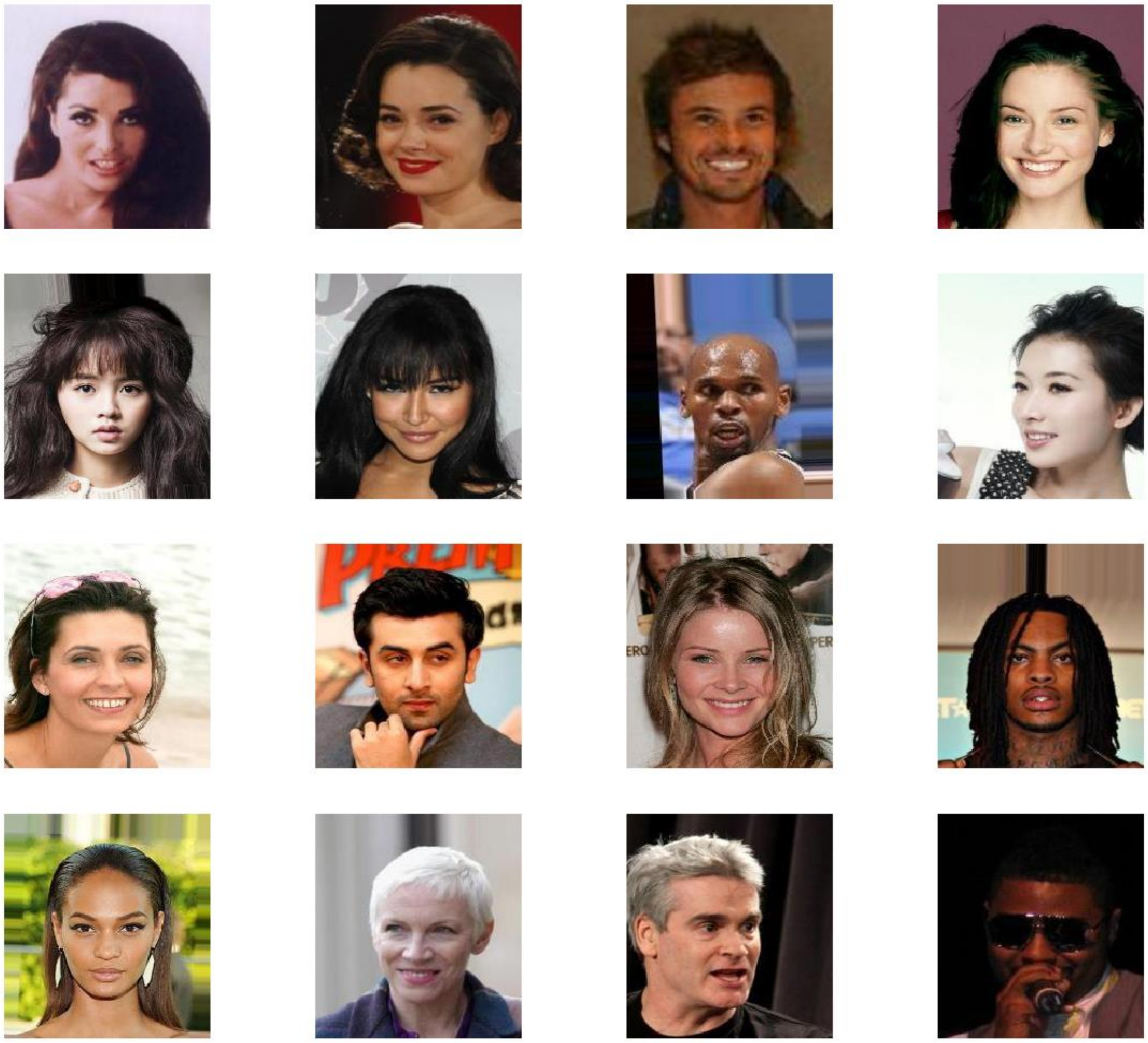
Exploratory Visualization

The dataset contains images with many different visual variations. Here is a summary of this variations.

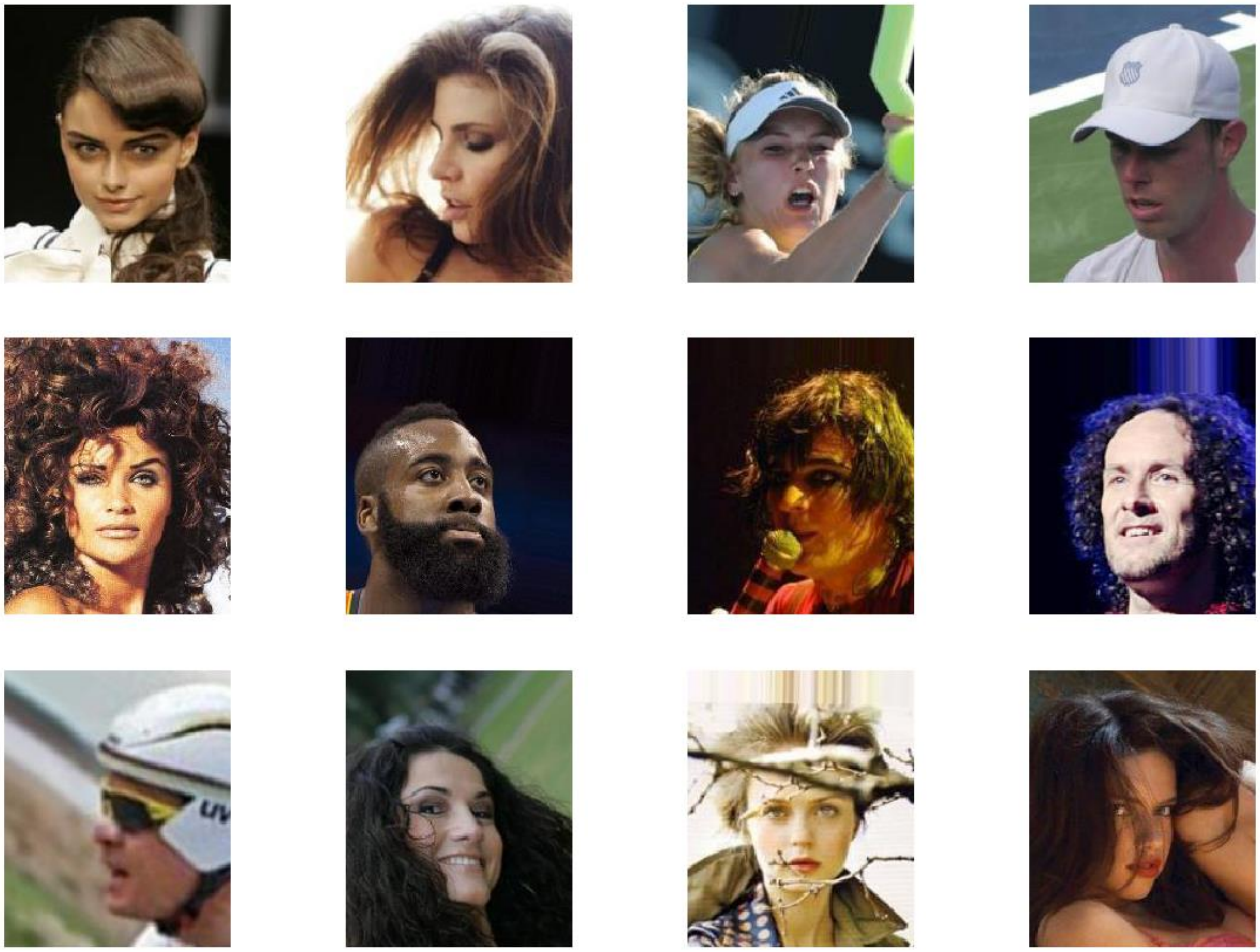
• Varying gender and ages



- Varying color complexity and ethnicity



- Varying hairs and face positions



Thus, we can see quite some variety and ambiguity in the dataset.

Algorithms and Techniques

Deep Convolutional Generative Adversarial Network or DCGAN consist of two primary networks a Deconvolutional network as a generator for generating images and a Convolutional network as a discriminator for recognizing images. The output of the generator will be an image and the output of the discriminator will be single value suggesting the probability of high likely the generated image from the generator is to the real dataset. The architecture is similar to the one proposed in this paper (Radford, Metz, & Chintala, 2015).

DCGAN architecture :

1. Layers :

    a. **Convolution**: Convolutional layers convolve around the image to detect edges, lines, blobs of colors and other visual elements. Convolutional layers hyperparameters are the number of filters, filter size, stride, padding and activation functions for introducing nonlinearity.

    b. **Flatten**: Flattens the output of the convolution layers to feed into the Dense layers.

    c. **Dense**: Dense layers are the traditional fully connected networks each neuron in one layer is connected to each and every neuron in the next layer in CNN they map the scores of the convolutional layers into the correct labels with some activation function.

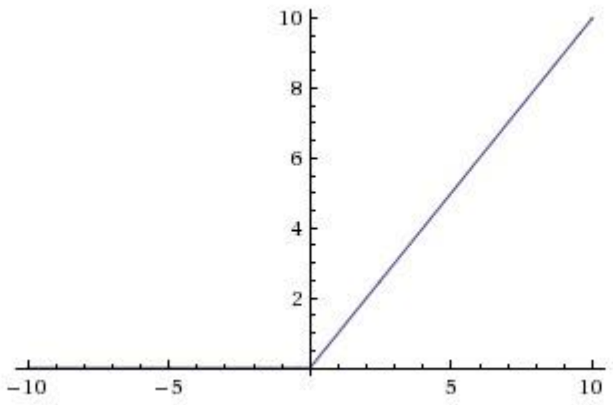
    d. **Dropout**: Dropout is a simple and effective technique to prevent the neural network from overfitting during the training. Dropout is implemented by only keeping a neuron active with some probability p and setting it to 0 otherwise. This forces the network to not learn redundant information.

    e. **UpSampling**: UpSampling refers to a technique that upsamples an image to a better resolution. Upsampling is commonly used in the context of Convolutional Neural network to denote reverse max pooling. (D & Rob, 2013)

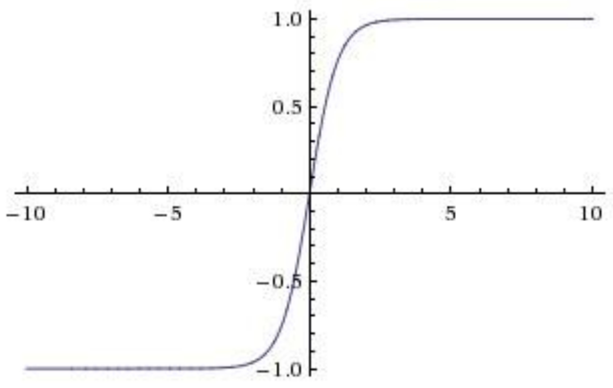
    f. **BatchNormalization**: This layer stabilizes learning by normalizing the input to each unit to have zero mean and unit variance. This helps deal with training problems that arise due to poor initialization and helps gradient flow in deeper models. This proved critical to getting deep generators to begin learning, preventing the generator from collapsing all samples to a single point which is a common failure mode observed in GANs. Directly applying batchnorm to all layers, however, resulted in sample oscillation and model instability. This was avoided by not applying batchnorm to the generator output layer and the discriminator input layer. (Radford, Metz, & Chintala, 2015)

2. Activation functions:

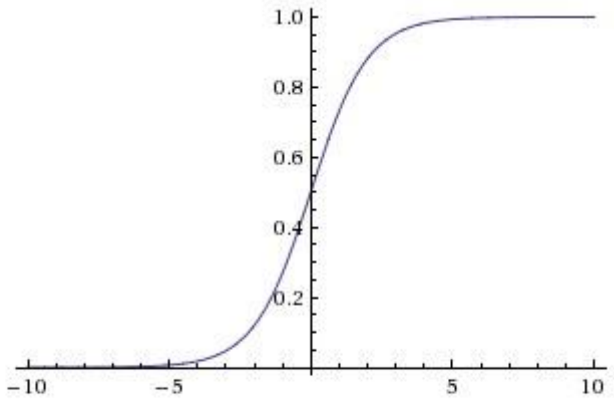
a. **Relu:** ReLu or Rectified Linear Unit computes the function $f(x)=max(0,x) to threshold the activation at 0. (Img src : (Johnson & Karpathy, 2017))



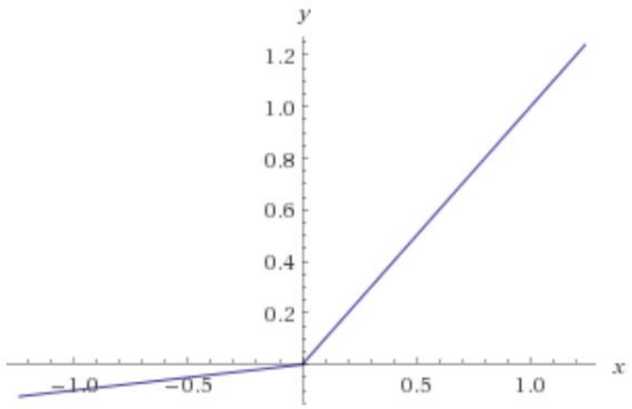
b. **Tanh:** Tanh or hyperbolic tangent is a function that computes f(x) = tanh(x) it gives output from range (-1, 1). (Img src: (Johnson & Karpathy, 2017))



c. **Sigmoid:** The sigmoid non-linearity has the mathematical form σ(x)=1/(1+e−x). It has output range from (0, 1) (Img src: (Johnson & Karpathy, 2017))



d. **LeakyRelu:** LeakyRelu or Rectified Linear Unit solves the “dying rellu problem” Instead of the function being zero when x < 0, a leaky ReLU will instead have a small negative slope (of 0.01, or so). That is, the function computes f(x)=1(x<0)(αx)+1(x>=0)(x)f(x)=1(x<0)(αx)+1(x>=0)(x) where α is a small constant. Alpha is a parameter in leakyRelu



3. Optimizers :

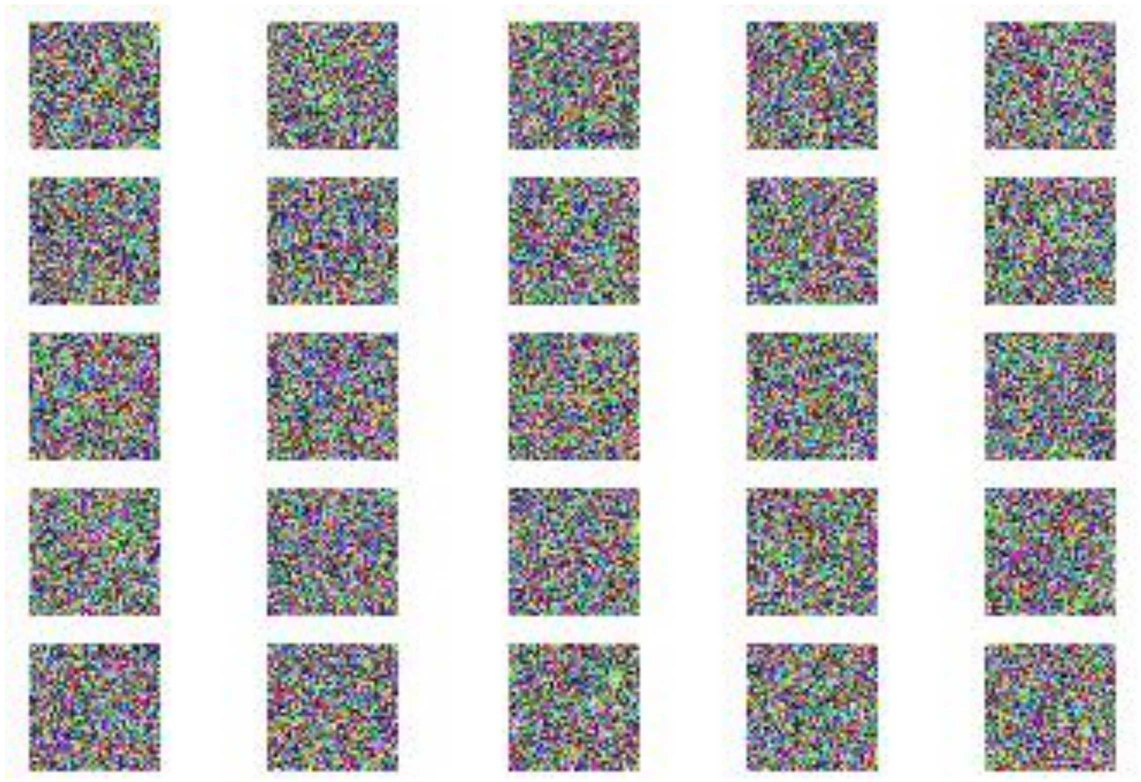
   a. Adam: Adam (Adaptive moment estimation) is an update to RMSProp optimizer in which the running average of both the gradients and their magnitude is used.In practice, Adam is currently recommended as the default algorithm to use and often works slightly better than RMSProp. Learning rate, decay, epsilon are the parameters.

The convolutional network or the discriminator will start with an Input layer to take in the input as an image the next layers will be a set of convolutional layers with increasing number of filters between the convolutional layers will be leakyRelu activation layer along with BatchNormalization and Droupout. After which follows the flatten layer which flattens the output from the convolutional layers, the final layer is a dense layer with 1 output with a sigmoid activation to get a single value probability.
For the deconvolutional network or the generator, the first layer is a dense layer to take in the input noise the next layer is a reshape layer to reshape the outputs from the dense layer, the reshaped image is then passed to a series of convolutional layers with relu activations with decreasing filter size as per the color channels and the image size needed along with upsampling

layers to increase the height and width of the image hence increasing the resolution after every convolutional layer there is also batch normalization layers. The final layer is a convolutional layer with either 3 filters for 3 color channels or 1 filter for 1 color channel this depends as per the required image the activation function used here is tanh.

Benchmark

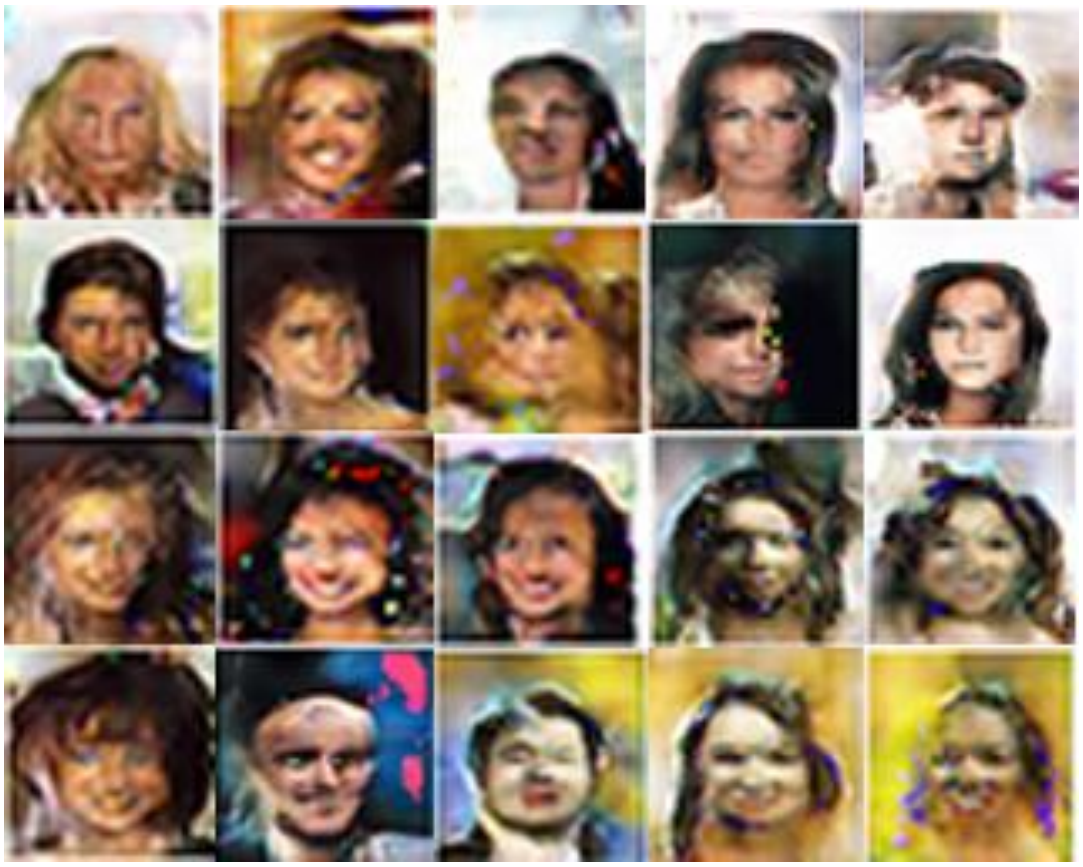
1. Random Generator: The random generator is implemented to generate a random image. This is done by using numpy's random module the code implementation can be found in the benchmark Folder. The images obtained from the random generator are :



a. Images produced by the Random Generator

The images produced from the random generator look nowhere near the actual images. Also, it produces 6% accuracy when passed to the CNN trained on the real dataset.

2. HyperGAN: The hypergan was trained on the same actual face dataset. I trained the HyperGAN with the default parameters and resized the input images to 64 X 64. The training was done for 8hrs. The result obtained was :



b. Images Produced by HyperGAN

The images produced do seem to capture the character of the real dataset but the produced images are not clear there is quite a lot of noise involved in these images. But it gives us a good benchmark to evaluate the results of our GAN.

Methodology

Data Preprocessing

All the images in the CelebA dataset are of size 172 x 218 if without resizing, these images were fed to the DCGAN then the network will consist of 61,746,499 trainable parameters. Training such network will take a long time so in order to reduce the training time of the network per step, we will resize the images to 64 x 64. After resizing the network will only consist of 8,328,003 trainable parameters which will be much easier to train and will take less time to train.

For resizing we will first load the image using OpenCV, by default OpenCV will load the images in BGR colorspace so a conversion to RGB will be required After converting to RGB we will resize the images to 64x64x3. The resultant image will be appended to a numpy array along with the other images. The complete dataset contains 200k images but for the training, we will only use 41k images or 20% of the dataset this number can be adjusted as per the requirements. But 41k images will be enough to train the discriminator. After all the 41k images are resized and stored in the numpy array we will save the array to be used later for training.

i. Before and after resizing

Also before feeding the images to the discriminator, we will normalize each pixel value of the image by dividing it by 255 so that the resultant pixel value will be in between 0 and 1. This normalization is required because neural network only takes inputs which are between -1 and 1.

Implementation

To make the final implementation of the GAN, I created a GAN class in python since keras doesn’t have an adversarial module yet. The class has three main methods 1. Build_generator, 2. Build_discriminator, 3. Init, 4.train

1. build_generator: This method builds a generator model according to the architecture and returns the model the parameter to this method is img_dim i.e. image dimensions which are the required image dimensions
2. build_discriminator: This method builds a discriminator model according to the architecture and returns the model the parameter to this method is the image dimensions which is to be given as input to the discriminator
3. __init__: The init method is the default method that is invoked whenever an instance of class GAN is created. The init method calls the build_generator and build_discriminaotr methods to make instances of generator and discriminator that are trainable on discrete batches. It then compiles both the models and assigns them the optimizer. This method also makes an instance of DCGAN by combining the instances generator and discriminator which is used to generate output images and later on compiles the resulting model. The parameters of this method are the image dimensions and the learning rate for the optimizer.
4. Train: This method is used to train the DCGAN as per the training phase it also saves the output of the DCGAN as per the save interval and also saves the model after every 10,000 epoch this number can be adjusted as per the need. This method also keeps track of the losses of the generator and discriminator at every step. The parameters of this method are the batch_size, n: the number of steps or epochs and the save interval.

Training Phase :

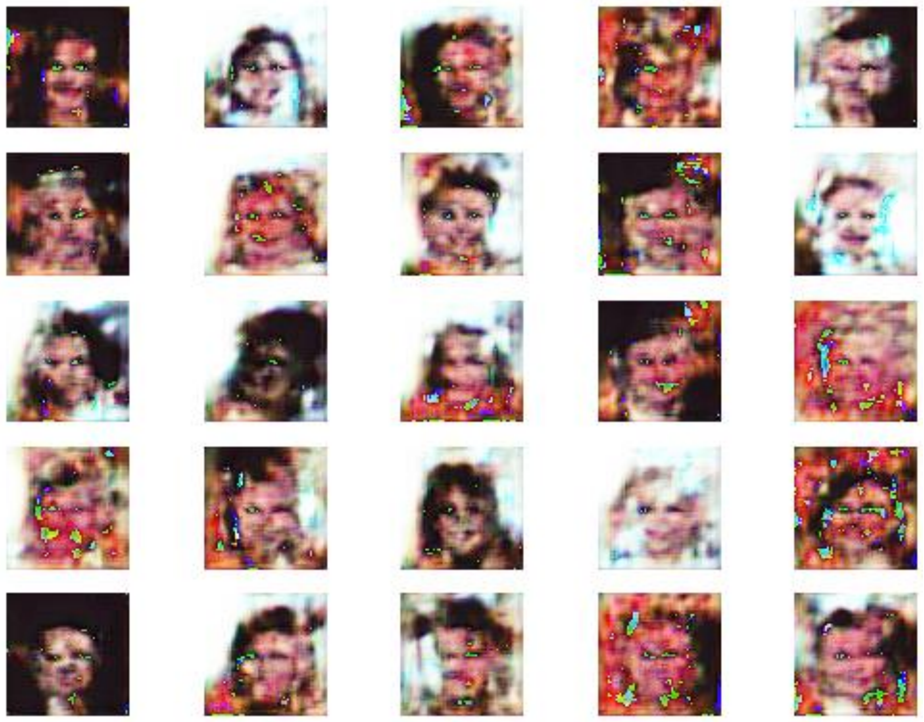
To train the DCGAN to output we carry out the following training process :

1. Firstly we generate random noise and we feed the random noise to the generator the output of the generator is an image this image generated is used to train the discriminator in distinguishing the fake images from the real ones.
2. The generated image is given input to the discriminator along with a real image from the dataset for the backpropagation the label attached to the generated image is 0 since it was generated by the generator hence not the real image, the label attached with the image from the dataset is 1 since it is the real image. The goal here is to train the discriminator to distinguish between real and the fake image as best as possible. At this stage, while training discriminator we set the trainable argument of the discriminator to true and after

training, we set it to false. Also, the loss and the accuracy obtained from the discriminator at every epoch is saved.

3. The last stage which is left is to train the generator, for training the generator we again generate random noise and feed the noise to the combined DCGAN model which is consisting of the generator and the discriminator combined. The image is generated by the generator and the same image gets into the discriminator which at the end gives out the probability how likely the image generated by the generator was to the real dataset then backpropagation is carried out at this time the real label given to the discriminator is 1. The goal here is to generate images that look as real as the original images in the dataset thus eventually we want the generator to generate images that look real hence the label attached to it is 1 i.e. the discriminator is 100 % certain that the image is belonging from the original dataset. Here the discriminator's trainable argument is set to false so only the generator is learning and not the discriminator. The generator loss at each epoch is also saved at this stage.
4. These three stages are performed at each epoch, and at each epoch, the generator becomes stronger at generating images identical to the dataset and the discriminator becomes stronger at distinguishing this images and at the end, we get two very strong models which train itself by trying to outperform each other. In this way, the training of DCGAN is carried out. (Note: Here the training phase is explained in context of a single image but during training, we do this on a batch of images)

For designing the generator and the discriminator I initially built simple models as per the architecture we discussed earlier with only a few convolutional layers and one fully connected layer. The generator had one dense layer to take in the input noise followed by two convolutional layers with 64, 3 filters and upsampling layers in between with relu activation and the discriminator consisted of one input layer to take in the input and two convolutional layers with 32, 64 filters and one fully connected layer with size on 1. I ran this model for 500 epoch just to get an initial benchmark the model did not train quite well the discriminator loss was about 8 and the generator loss fluctuated between 2- 14 this model was nowhere near to produce any decent images. Neither the discriminator nor the generator seemed to learn the features from the face dataset. This was the clear indication of the lack of layers in the generator and the discriminator. So I increased the number of convolutional layers in both the generator and the discriminator with additional layers with filters 64, 128, 256 respectively so that it could capture the required features from the faces in images. After modifying the architecture I again ran the model for 500 epoch and got quite a few improvements the discriminator and generator losses went down to 2 and 3 and the images obtained were not quite close to the real dataset but they would have improved eventually if training was continued. The output obtained was :

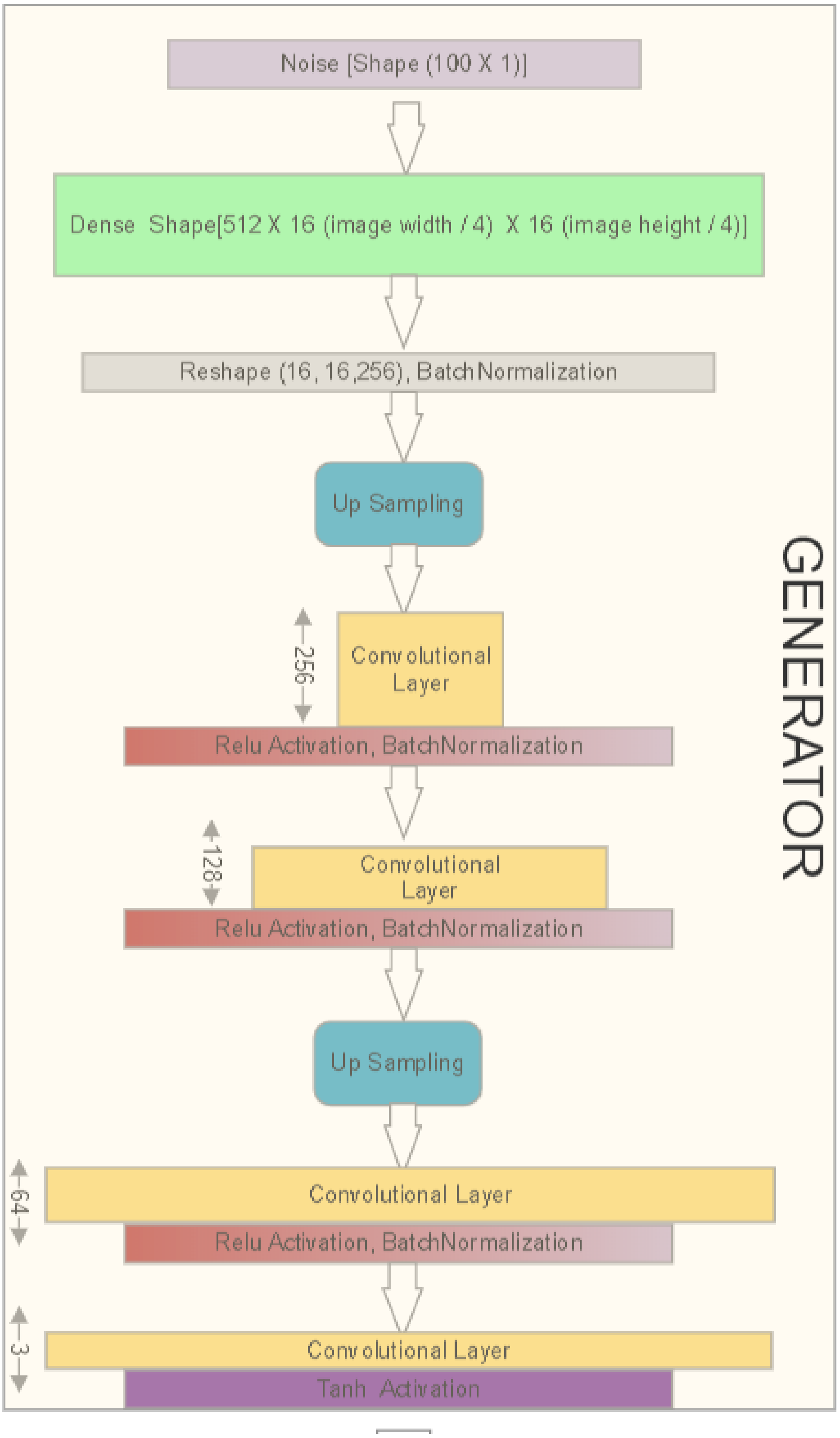


a. Results obtained from improved model after 500 epochs

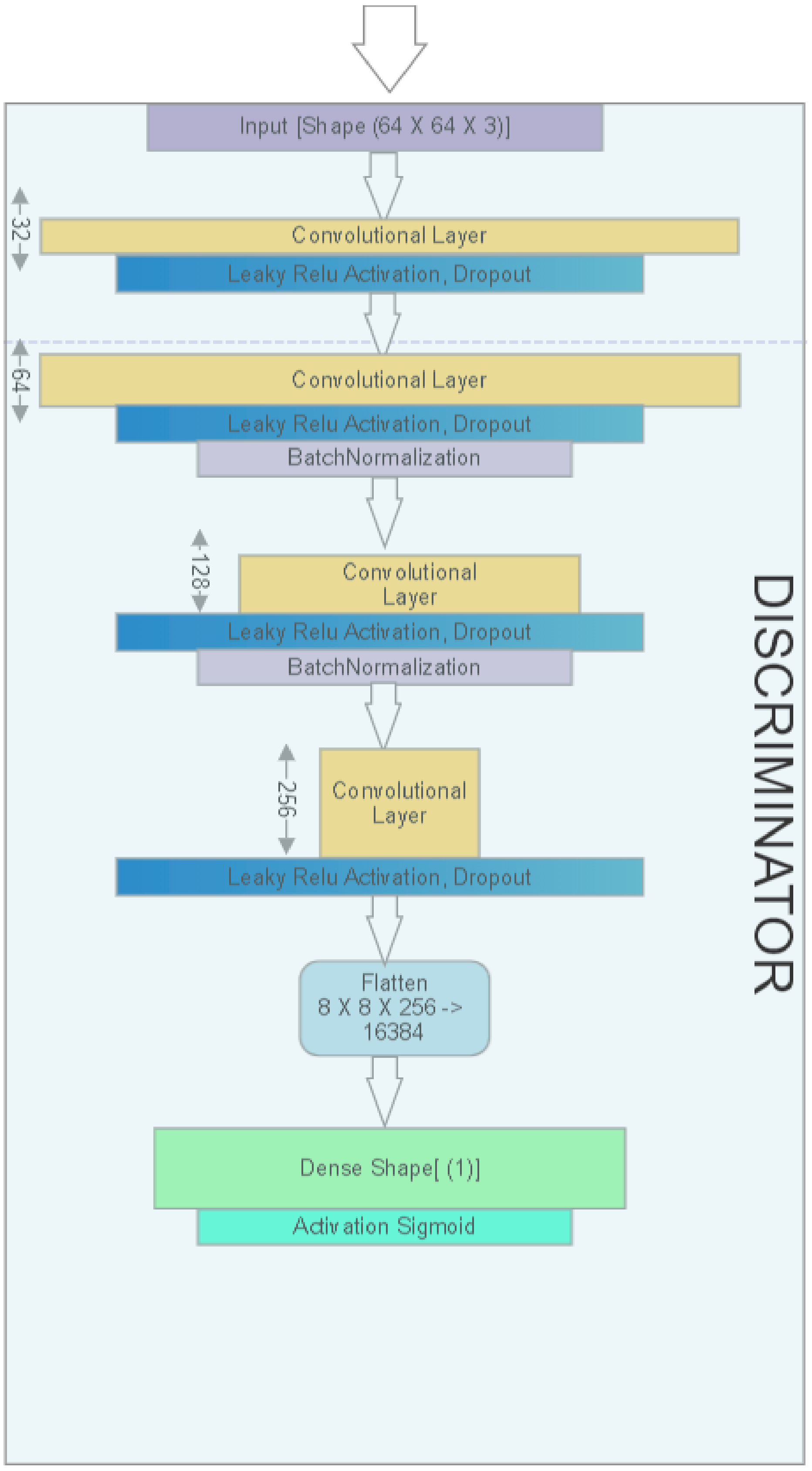
The problems that I saw during the training was that the discriminator loss decreased immediately after 20-30 epochs and would get accuracy near to 100 due to which the generator loss went up quickly and it took quite some time for it again decreased to a smaller value. This was clearly due to the overfitting of the discriminator on the generated images.

Refinement

To solve the overfitting problem I added several dropout layers in between convolutional layers and this seemed to solve our problem. I also applied batch normalization in the model to prevent arbitrary large weights in the intermediate layers as the batch normalization normalizes the intermediate layers thus helping to converge well. The final model for the DCGAN was as follows :



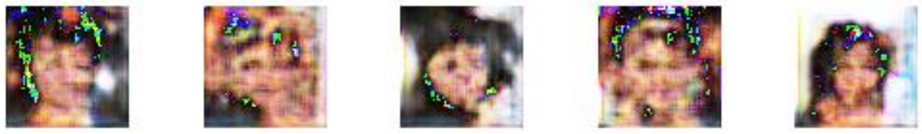
Noise [Shape (100 X 1)]
Dense Shape[512 X 16 (image width / 4) X 16 (image height / 4)]
Reshape (16, 16,256), BatchNormalization
Up Sampling
256
Convolutional Layer
Relu Activation, BatchNormalization
128
Convolutional Layer
Relu Activation, BatchNormalization
Up Sampling
64
Convolutional Layer
Relu Activation, BatchNormalization
3
Convolutional Layer
Tanh Activation
GENERATOR

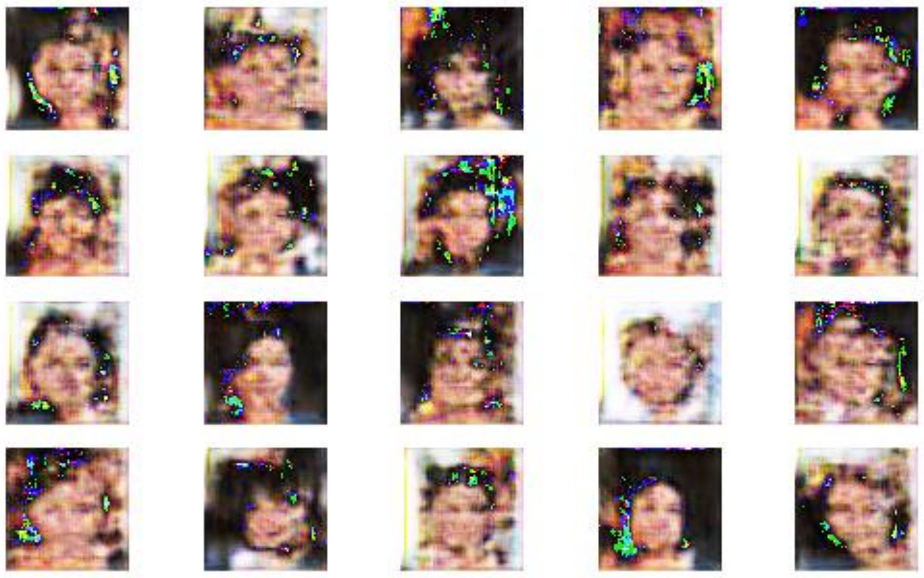


b. Final DCGAN architecture

The generator consists of first the dense layer with noise as the input of size 100 x 1. The output size of the dense layer is 65536. The next is a reshape layer to reshape the output vector of the dense layer in form of a 3d array the reshape layer reshapes the input vector of size 65536 into a 3d array of size 16 x 16 x 256. After the reshape layer is the upsampling layer the output of the upsampling layer is an array of size 32 x 32 x 256. The next layer is a convolutional layer with 256 filters the output shape remains the same after the convolutional layer follows activation layer with relu activation function and batch normalization. This type of layer continues with decreasing filter size the output shape from the convolutional layer with 128 filters is 32 x 32 x 128 next is again an upsampling layer which increases the size to 64 x 64 x 128. The next follows convolutional layer with the same structure having 64 filters the output shape from this layer is 64 x 64x 64. The final layer is also a convolutional layer with only 3 filters which corresponds to the RGB color channels the output shape from this layer is 64 x 64 x 3 which is our required image that is finally passed onto an activation layer with tanh activation function.

The architecture discriminator is completely opposite to that of the generator the first layer is an input layer to take in the input image the next is a convolutional layer with 32 filters the output shape from this layer is 32 x 32 x 32 the next is an activation layer with leakyRelu activation followed by a Droupout layer. The next layer is a batch normalization layer. A similar structure follows later on with convolutional layers with increasing filter size of 64, 128, 256 the output shape from the last convolutional layer i.e. layer with 256 filters is 8 x 8 x 256 the next two layers are the leakyRelu activation layer and a Dropout layer. The output from this is given to a flatten layer that flattens the 3d array into a vector of size 16384 this output is given to a dense layer with the size of 1. The output from this dense layer is a single value probability. Result obtained after refining the model.





c. Results obtained after refining the model for 500 epochs.

Hyperparameters set for the generator :

1. Kernel size: 4, To set the filter size to 4,4 and to perform fast convolutions on the image.
2. Momentum: 0.8, To avoid the problem of falling in local minima whenever one of the models falls to its minimum value.

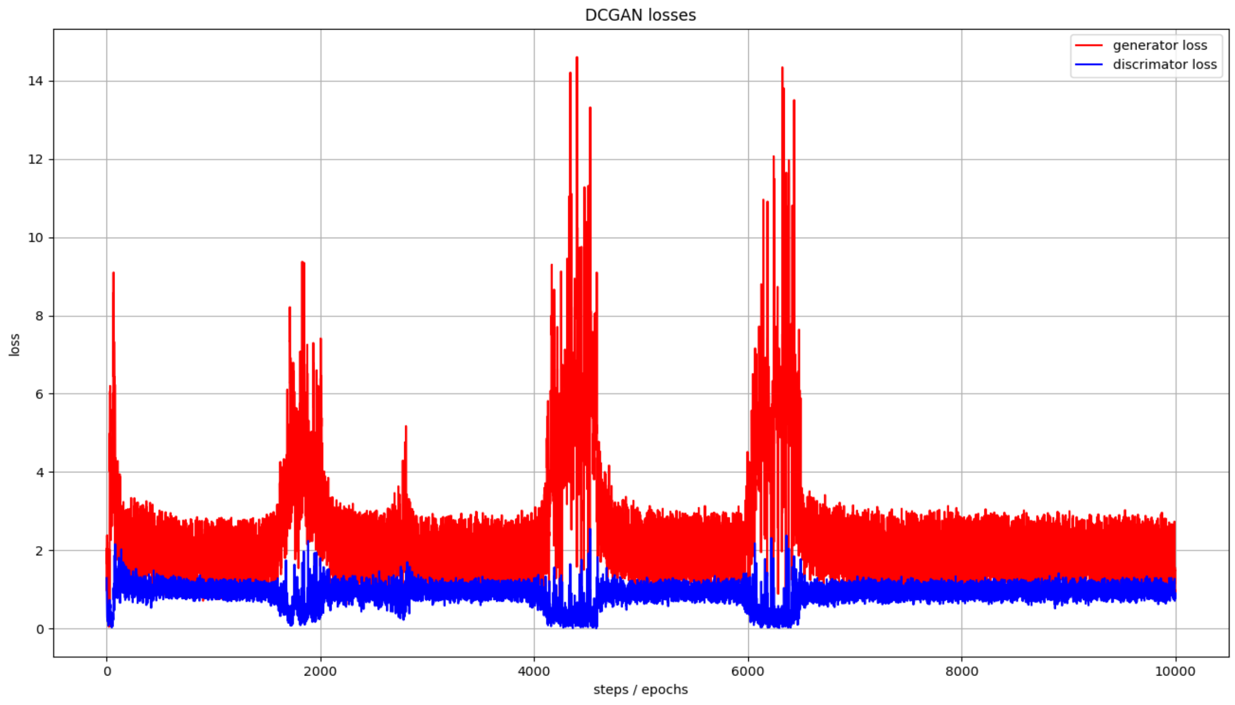
Hyperparameters set for the discriminator:

1. Kernel size: 4, To set the filter size to 4,4 and to perform fast convolutions on the image.
2. Strides: 2, To jump 2 pixels at a time.
3. Droupout: 0.25 between first two layers and 0.4 between the rest, To prevent the model from overfitting.

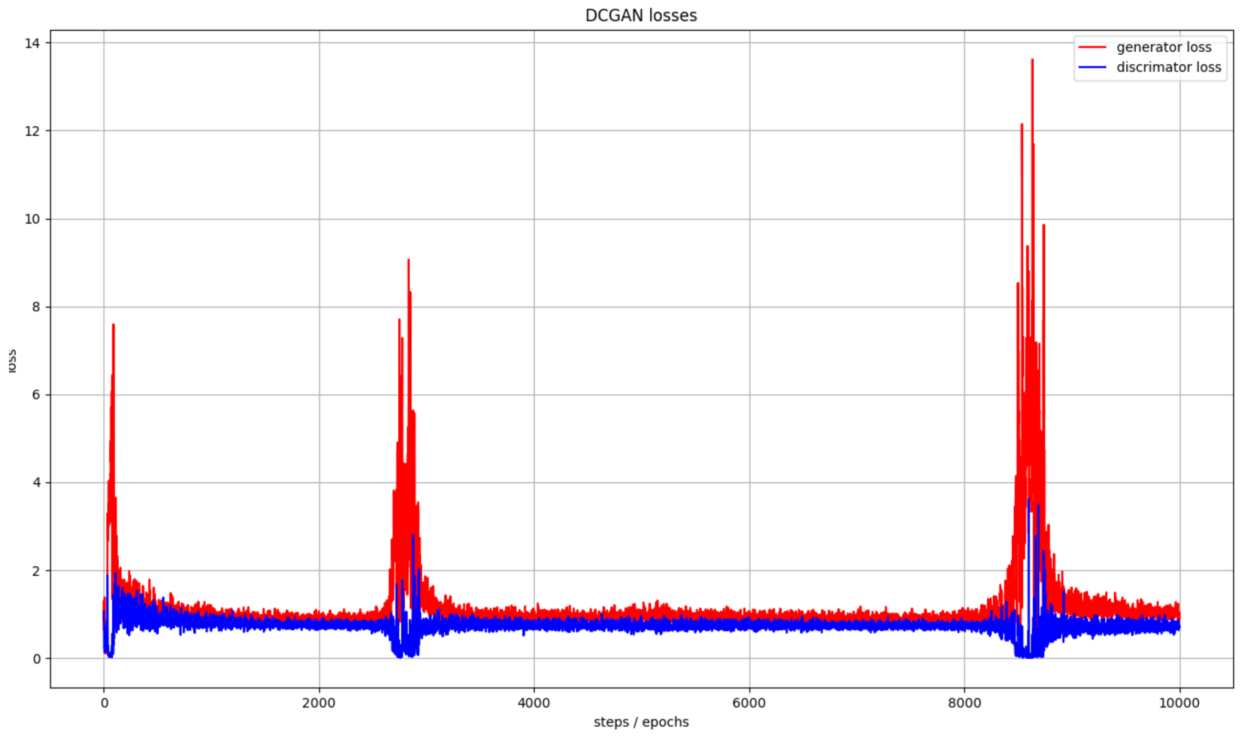
Results

Model Evaluation and Validation

The model discussed in the refinement section was used for the final implementation the model was initially trained for 10,000 steps with a learning rate of 0.02 but generator loss would stay around 2.4. The mean generator loss obtained was 3.43 and the mean discriminator loss obtained was 0.89.



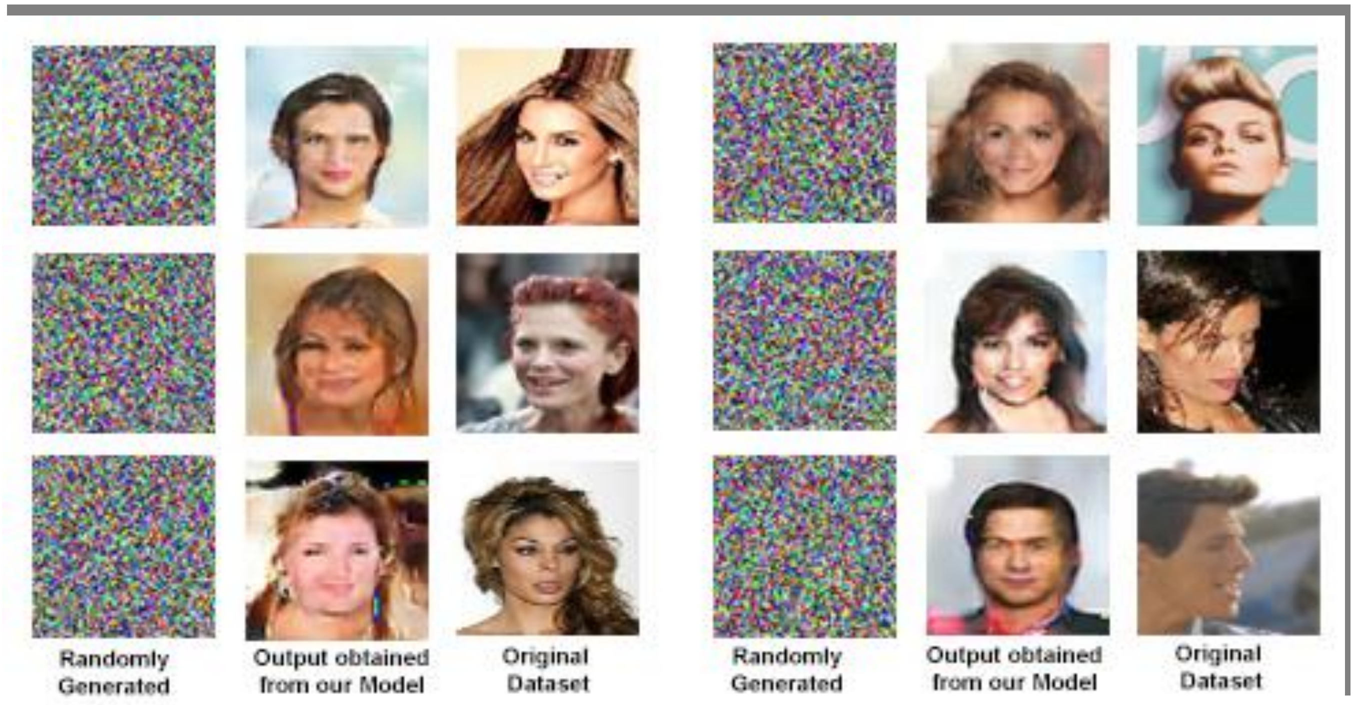
This was probably due to high learning rate. In order to decrease the generator loss, the model was again trained from start with a learning rate of 0.002 the losses of generator and



discriminator were much better with generator having an average loss of 1.169 and discriminator average loss of 0.728. The final model was trained for 100,000 epochs.

Justification

To compare the results obtained from the benchmark models and from our final model we will be visually comparing the outputs to the original dataset.

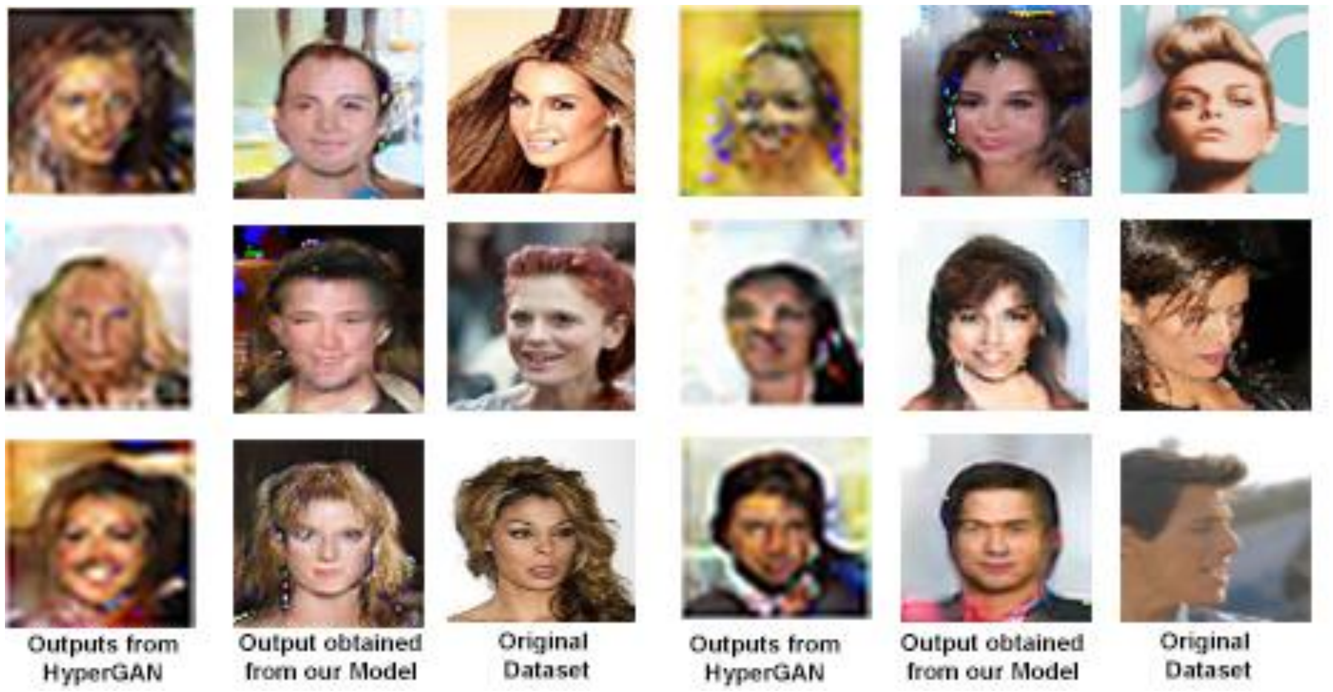


1. Comparing Outputs from DCGAN with random generator

The images generated from our model matches quite closely to the original data while the images from the random generator are not even close to look similar to resemble the original images. From this, we can safely conclude that the images generated from our GAN surpass the ones generated from the random generator.

The images produced from the HyperGAN seems to capture the characteristics of the faces from the real dataset but the produced images are not clear there is quite a lot of noise involved in these images also the images look more blurred as compared to ones generated from our

DCGAN, this can be improved through further training but from this observation we can clearly say that the implementation used and the results obtained are comparable to the existing models.



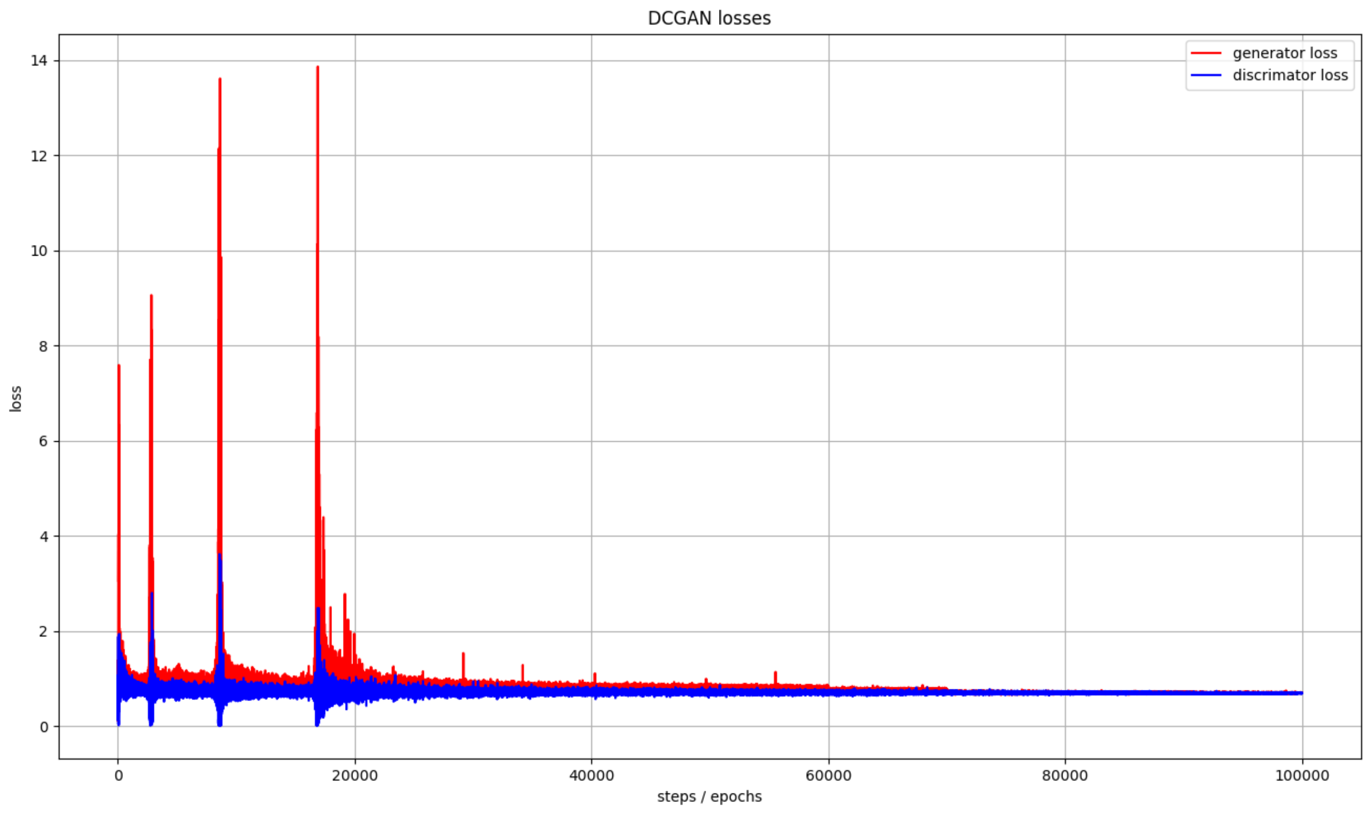
2. Comparing Outputs from DCGAN with HyperGAN

From the above observations, we can state that the final results obtained have solved the problem to a certain extent. The final images obtained looked quite similar to the real face images but it had issues like random background noise, morphing and lack of sharpness we will talk about this further in the improvement section

Conclusion

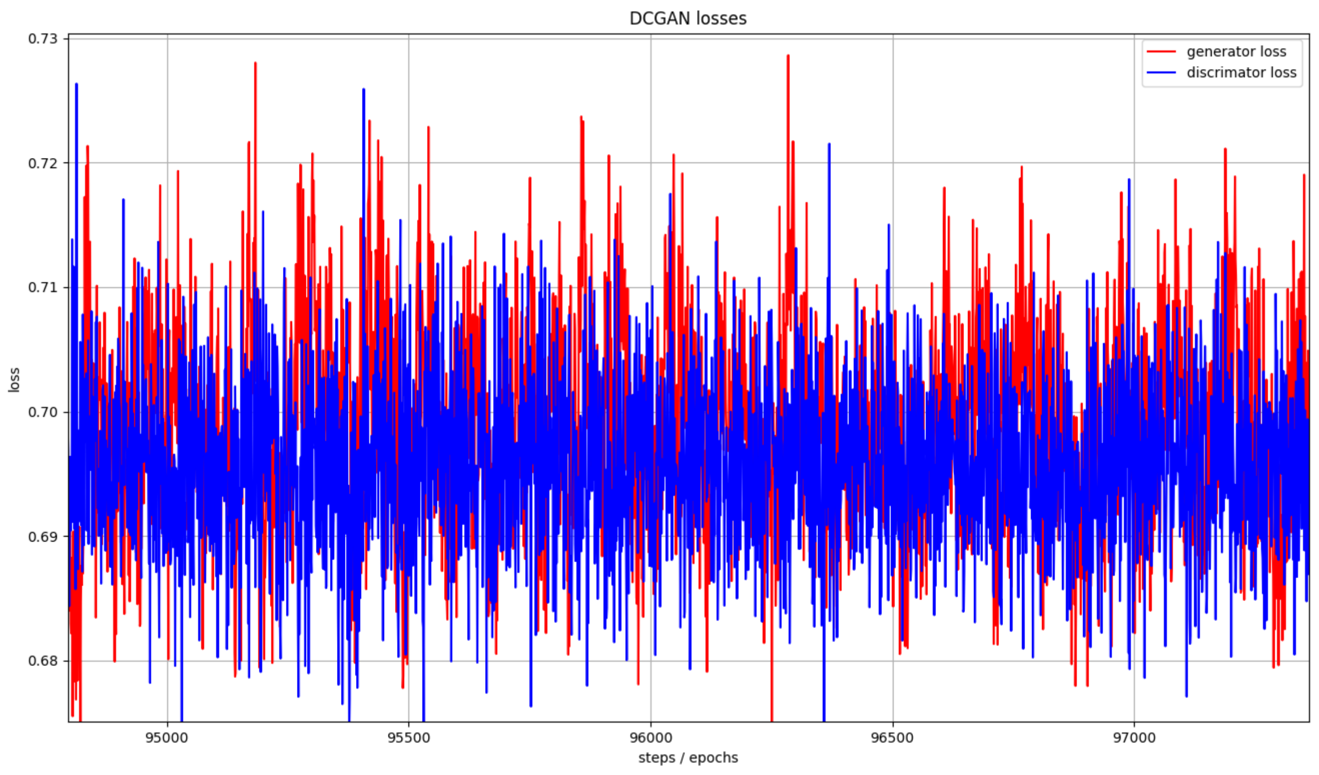
Free-Form Visualization

The final model was trained for 100,000 epochs the losses from generator and discriminator were tracked for each step. The losses for the generator and the discriminator are plotted below

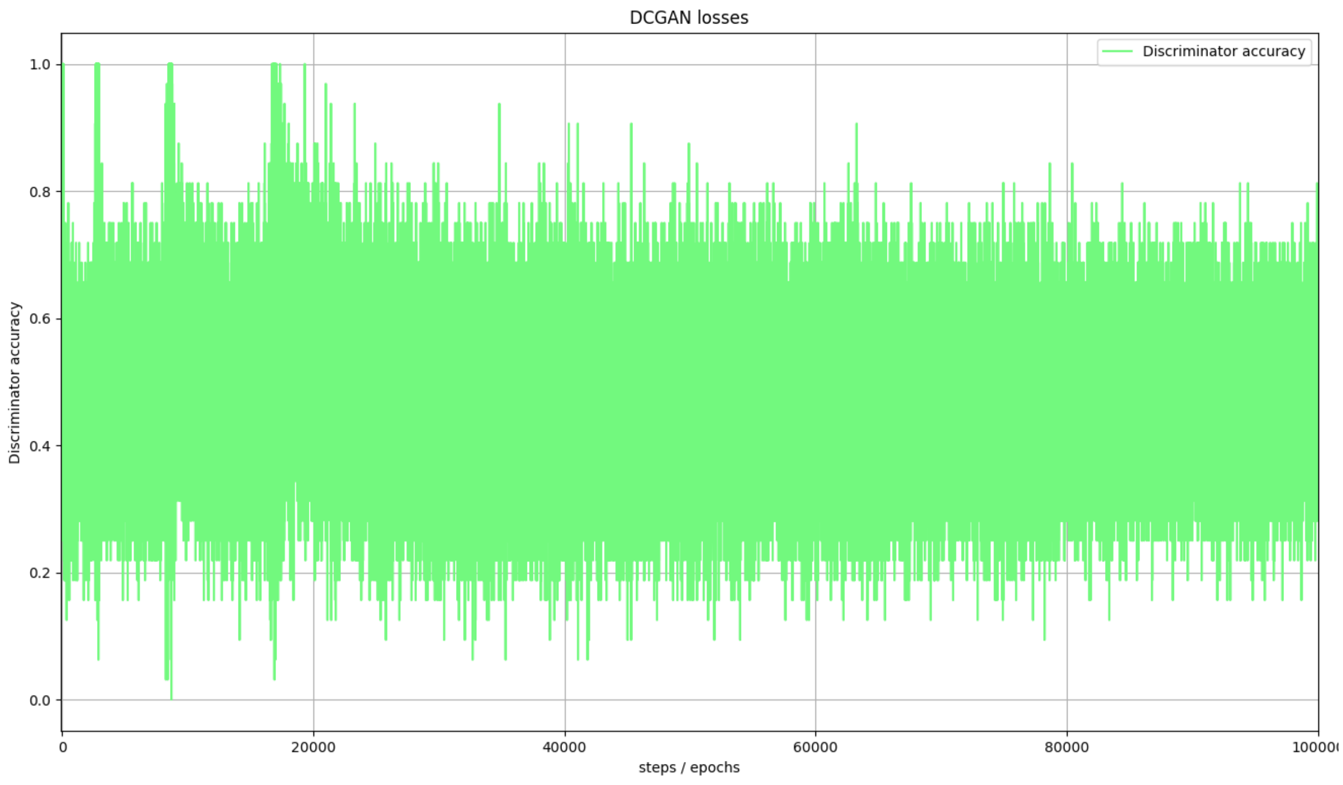


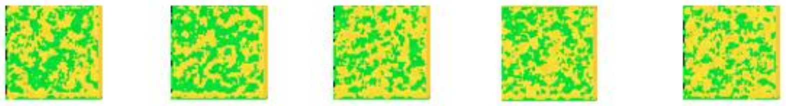
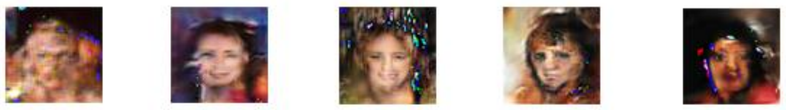
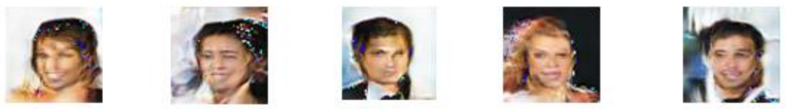
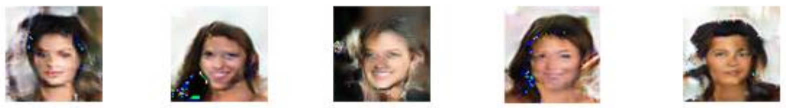
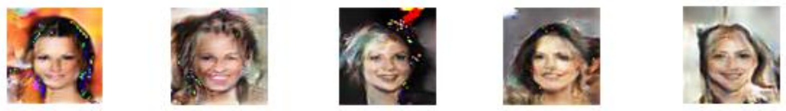
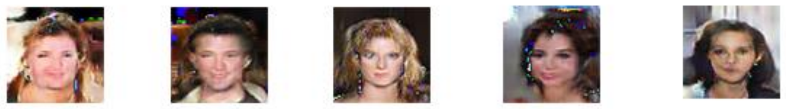
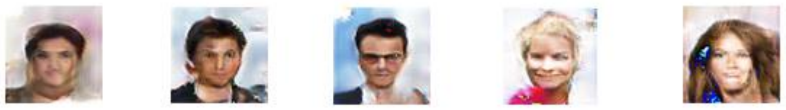
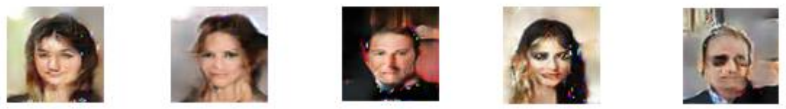
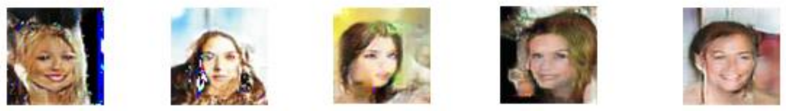
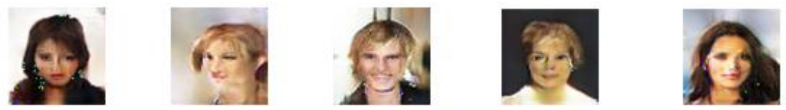
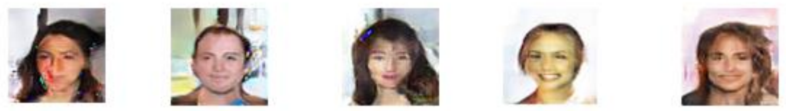
From the above graph, we can see the losses of generator and discriminator plotted on the Y-axis and the number of steps on X-axis. Initially, the loss from the generator is quite high averaging around 1.13 till 20,000 steps but it lowers down as the model trains for more time. On the other hand, the loss from the discriminator is quite low as compared to the generator. From the above graph, we can see an interesting relationship between generator and discriminator in GAN's which is whenever the loss of one model goes down the loss of other model goes up, this is the min-max relationship which is to be seen in GAN where one network tries to outperform the other this can be seen in the above graph near 20,000 epoch where discriminator loss is close to 0 and generator loss is at it's highest at around 14.

But as the training continuous we can see the two models begin to maintain an equilibrium this can be seen after 40,000 – 60,000 epoch and after 70,000 epoch we can see the losses of the generator and discriminator to nearly be the same value, but that's not true having a closer inspection we can see that still the two models are still trying to outperform each other where a decrease in one model's loss leads to increase in the other. This can be observed from the plot below.



The average loss of the generator during 100,000 epochs is 0.84404 and the average loss of discriminator is 0.707. The accuracy of the discriminator over time can be visualized from the plot below.



EPOCH	GENERATED IMAGES
0	
0 - 10,000	
10,000 - 20,000	
20,000 - 30,000	
30,000 - 40,000	
40,000 - 50,000	
50,000 - 60,000	
60,000 - 70,000	
70,000 - 80,000	
80,000 - 90,000	
90,000 - 100,000	

The average accuracy of the discriminator is 0.4819 or 48 % which seems to be an appropriate value for a well trained GAN.
The outputs obtained from the GAN can be summarized from the figure above. Over time we can see the improvement in the quality of images generated by the generator. We can also see that the generator is able to capture the variations that were observed in the face dataset and reflect these variations in the produced images, we can see a variety of different images having various hair color, face complexity, background, accessories (glasses, hats etc.).

Reflection

The process used for this project can be summarized using the following steps:-

1. An initial problem in the latest field of research was found
2. A relevant solution was documented and was made ready for implementation
3. A public dataset to be used for evaluating the problem was found
4. Algorithms to be used for solving the problem was devised.
5. Data was downloaded and pre-processed according to the requirement
6. Benchmarks were established to compare the implementation
7. An initial model was prepared and trained.
8. Improvements were made to the initial model and the final architecture for the model was devised
9. The model was trained and during the training, the results were collected.
10. The final trained model was compared to the benchmarks.

I found step 8 and 9 to be the most difficult steps , I had to make countless improvements so that the model could produce some decent outputs also during the training the main problem which I faced was overfitting of the discriminator, this happened a number of times which caused the loss of the generator to go to its peak value and stay there, how hard the generator tried to outperform the loss of the generator won’t go down due to which, I had to make numerous changes to the architecture of the discriminator to overcome the issue of overfitting. Also finding the point where the loss of generator and discriminator were lowest was difficult.

Improvement

As mentioned earlier there are several improvements that have to be made so that the images generated would look the same to the original dataset.

The problems with the images that were generated from the GAN are :

1. Images being blur and lacking sharpness
2. Noisy background seen in the images
3. Cases of morphing seen in the images i.e. some images generated look like a combination of several images.

To improvise the results the following measures can be taken :

1. Increasing the batch size and training time.
2. Increasing the size of input images and generated output images.
3. Adding more convolutional layers with a large number of filters to the network *. (This might improve the quality of generated images but the result has to be tested)
4. Using a different architecture for the model.

Initially, I considered comparing the results from the DCGAN to different generative models like Boltzmann Machine and Variational Auto-Encoders. But to implement them and to get the appropriate results would take a lot of time and research since they are advanced concepts in Deep Learning. So I excluded them from this project as of now,

The final solution obtained is quite good requiring some minor improvements but there definitely exists better solutions that have obtained better results than mine. One of the solutions can be seen on this blog http://torch.ch/blog/2015/11/13/gan.html they used GAN along with VAE to generate higher quality images.

References and Citations

Bruner, J., & Deshpande, A. (2017). *Generative Adversarial Networks for Beginners*. Retrieved from Oreilly: https://www.oreilly.com/learning/generative-adversarial-networks-for-beginners

D, Z. M., & Rob, F. (2013). Visualizing and Understanding Convolutional Networks. *Cornell University Library arxiv.org*, 1-2.

Goodfellow, I. (2016). NIPS 2016 Tutorial: Generative Adversarial Networks. *Cornell University Library*, 2-4.

Goodfellow, I. J., Pouget-Abadie, Jean, Mirza, M., Xu, B., Warde-Farley, . . . Bengio, Y. (2014). Generative Adversarial Networks. *https://arxiv.org/abs/1406.2661 Cornell University Library,* 1-5.

Johnson, J., & Karpathy, A. (2017). *CS231n Convolutional Neural Networks for Visual Recognition*. Retrieved from github.io: https://cs231n.github.io/neural-networks-1/

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *https://arxiv.org/abs/1511.06434 Cornell University Library*, 3-16.

SHAIKH, F. (2017, June 15). *Introductory guide to Generative Adversarial Networks* . Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2017/06/introductory-generative-adversarial-networks-gans/

Tang, Liu, Z., Luo, P., Wang, X., & Xiaoou. (2015). *Deep Learning Face Attributes in the Wild.*

Wikipedia. (2017). *Activation function*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Activation_function