IMAGE GENERATION USING TEXT

Project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering/Information Technology

by

Prajwal Thakur (161616)

Under the supervision of

Mr. Rizwan Ur Rehman

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

Candidate’s Declaration

We hereby declare that the work presented in this report entitled **“Image Generation Using Text”** in partial fulfilment of the requirements for the award of the degree of **Bachelor of TechnologyinInformation Technology** submitted in the department of Computer Science and Engineering and Information Technology**,** Jaypee University of Information TechnologyWaknaghat is an authentic record of my own work carried out over a period from August 2019 to May 2020 under the supervision of **(Rizwan Ur Rehman)** (Assistant Professor, Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

[illegible]

Prajwal Thakur (161616)

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

[illegible]

Rizwan Ur Rehman

Assistant Professor

Department of Computer Science & Engineering and Information Technology, JaypeeUniversity of Information Technology

Dated:

ACKNOWLEDGEMENT

Foremost, we would like to express our sincere gratitude to our project guide **Rizwan Ur Rehman (Assistant Professor)** for the continuous support in our project, for their patience, motivation, enthusiasm, and immense knowledge. Their guidance has helped us in all the time of this study and writing of this report. We could not have imagined having a better advisor and mentor for our project. We would also like to thank them for lending us their precious time.

We are also very thankful to all the faculty members of the department, for their constant encouragement during the project.

Last but not least we would like to thank our parents, who taught us the value of hard work by their own example.

Date: …………….

Prajwal Thakur (161616)

Contents

List of Figures

List of Abbreviations

GAN	Generative Adversarial Networks
CNN	Convolutional Neural Network
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
LSTM	Long Short-Term Memory
RNN	Recurrent Neural Network

ABSTRACT

Synthetic Content Generation Using Machines is a very trending topic in the field of DeepLearning and it is an extremely difficult task even for the state-of-the-Art ML algorithms. Theupside of Using Deep Learning to do this is that it can generate Content that does not existyet. In the recent past Generative Adversarial Networks (GAN) have shown great promise when it comes to generating images but they are difficult to train and condition on any particular input which acts as a downside for them. However, they have tremendous applications in generating content in an unsupervised learning approach like generating video, Increasing the resolution of Images or Generating Images from Text. In this project we look at generating 64*64 Images on the fly using a text as an Input.The images generated will be unique in terms that they do not already exist and in doing that we will improve upon already existing Architecture models and try to reduce the difficulties that come with training GAN Models like Reduced Training Time and Better Convergence of The Model.

The Final Project will be a WebApp,where, you can Input a Text and a Synthetic Image Will be generated Based on the Description of the Text.

CHAPTER 1: INTRODUCTION

1.1 Introduction

For a human mind it is very easily to thin of new content. what if someone asks you to “draw a flower with blue petals”. It is very easy for us to do that. but machines process information very differently. Just understanding the structure of the above sentence is a difficult task for them let alone generate something based on that description.

Automatic synthetic content generation is a field that has been explored in the past and was discredited because at that time neither the algorithms existed nor enough processing power that could help solve the problem. However, the advent of deep learning started changing the earlier beliefs. The tremendous power of neural networks to capture the features even in the humongous of datasets makes them a very viable candidate for automatic content generation. another milestone was achieved when Ian Good Fellow proposed generative adversarial networks in 2014.

GANs are a kind of architecture in Deep learning that can produce content from random noise. What is even more unique about GANs is that the content they create represents the dataset on which they are being trained upon but it is totally unique in some way or the other.

Generating an image from a text-based description is one aspect of generative adversarial networks that we will focus upon. Since the GANs follow unsupervised learning approach we have modified them to take am input as a condition and generate based on the input condition. This can form base for a large number of things like synthetic audio generation like the ones used in Siri or assistant, video content generation from just scripts. imagine entire movies made out of just the script. These are some uses that many companies are researching about. modifying GANs and applying conditions on them isn’t limited to just generating images, we can use it to create passwords that are very hard to crack and numerous similar applications like this

Deep Learning and Content Generation

Deep Learning is a field that utilises and relies completely on Various Flavours of Neural Networks to Extract Insights from the data and find patterns among that data. While it has been shown to be very successful in things like Image Classification (In some datasets even beating human level accuracy by a large margin) and Time Series Analysis(There are so many factors involved that it even becomes difficult for a human to take all those into account), A completely different Aspect of it has been started to explore.

The big Question Being

"Can We use Deep Learning to Generate Content?"

As we know Neural Networks can extract features of a dataset that they have been trained upon, the goal becomes using those features to create new data points that do not belong in the dataset itself.

Generative Adversarial Networks

Generative Adversarial Networks (GAN’s) were created by Ian Good Fellow in 2014 in an attempt to generate content instead of just representing it in a compact form and they are the most successful kind ofDeep Learning Models that are even remotely close to the task.

What does GAN do?

Basically, it can be trained to generate data from a scratch or random noise.

It consists of two building blocks:

1) Generator:

The task of the Generator is to take in some input and generate something out of it. In cases ofImages it might take in some noise asinput and generate an image which might not mean anything Initially. It is simply the reverse of what a standard Convolutional Neural Network (CNN) is.A CNN takes in input as an image and down samples it along the height and width dimensions while increasing it along the channel dimension which acts as our features essentially. What a Generator Does is it takes in a down sampled input and through various Up sampling Operations Generates an Image.

By comparing the real images and the images that is generated by generator, GAN builds a discriminator that helps us to learn the differences that makes that image real and then after it will provide feedback to generator about the image that is to be generated next.

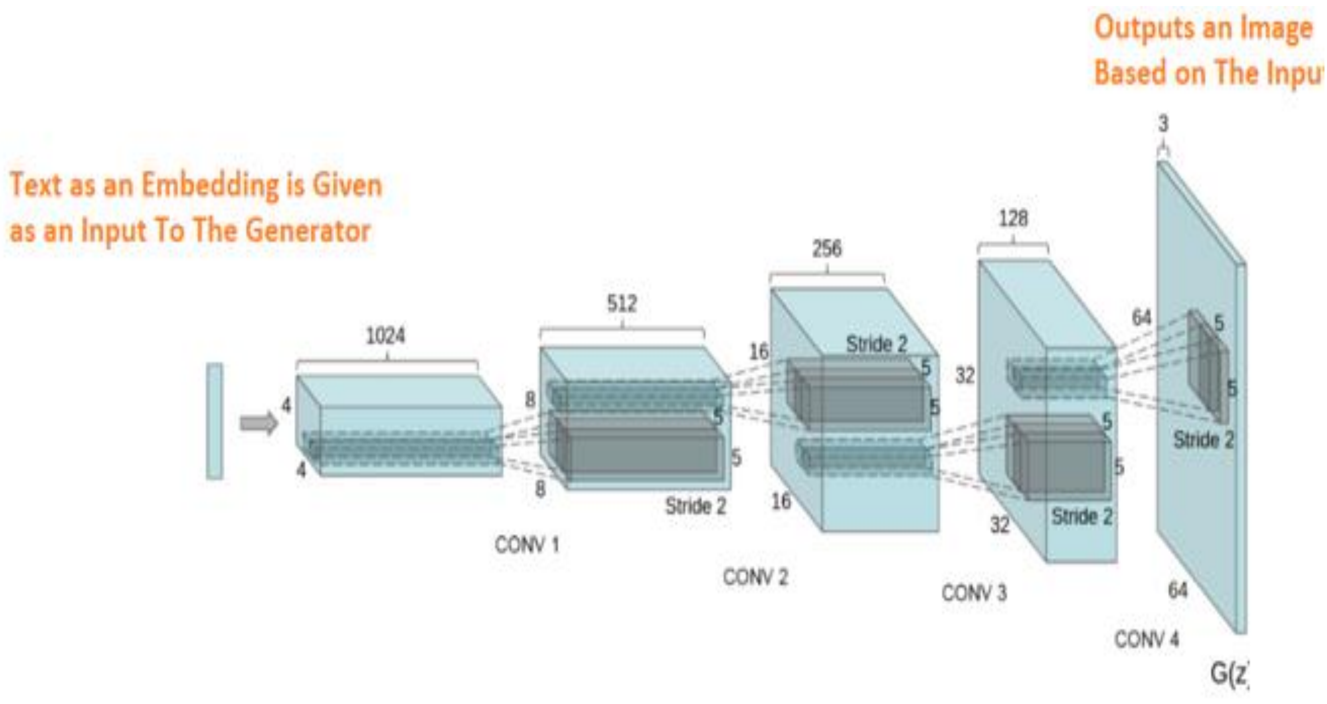
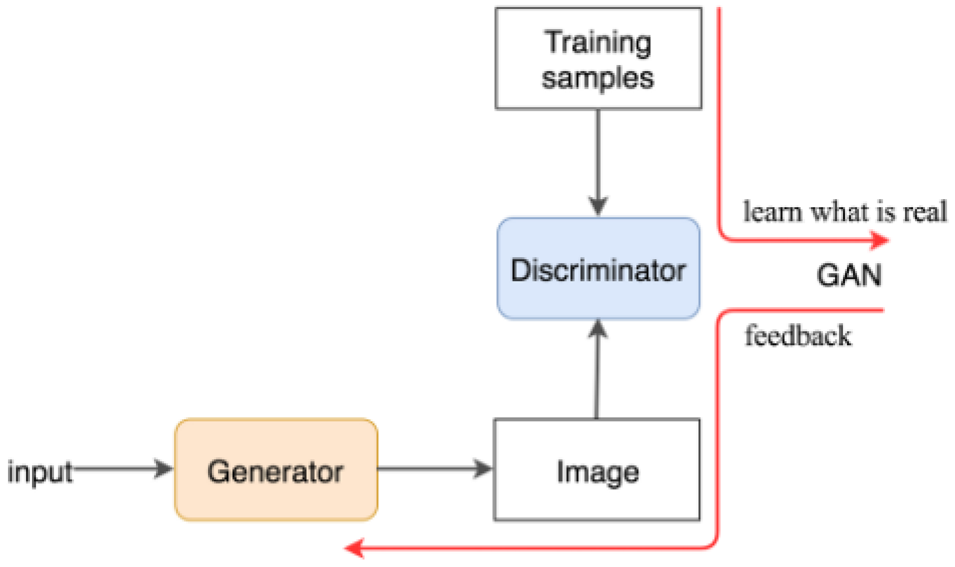


Figure 1 A standard Generator Architecture for our task

2) Discriminator:

Generator alone will just generate something random, so basically discriminator will give guidance to generator on what images it should create. Discriminator is nothing more than a simple convolutional neural network that takes in an image as an input and determines whether the image came from the original dataset or is it an image generated by the generator. Simply taking in an image as an input it determines whether it is real or fake (Synthetically Generated by Generator).



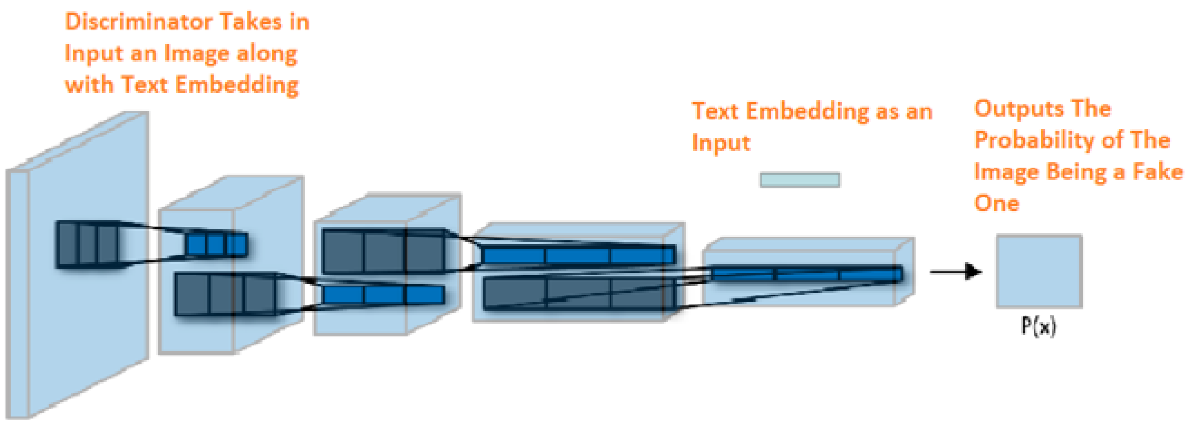

Discriminator Takes in Input an Image along with Text Embedding
Text Embedding as an Input
Outputs The Probability of The Image Being a Fake One
P(x)

Figure 2A Standard Discriminator for our Task

Why “Adversarial” and How Do they Even Learn

It has been already established that a generator creates an image and a discriminator identifies whether it is fake or real. But the bigger question is “How do these two model architectures learn to do that” and in answering that we will exactly know what “Adversarial” mean in our Context. So, models in deep learning optimise themselves by using something known as a **“Loss Function”.** A loss function is simply a cost function that the model tries to minimize and in doing so it starts representing the dataset better by fine-tuning its internal parameters. So, a generator takes in an input and generates an image which doesn’t mean anything in the beginning. Now the image is being passed on to a discriminator which can easily identify the difference because essentially it is very different from what the dataset represents after a few iterations. now we create two loss functions, one for the generator and other for the discriminator

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

Equation 1The loss function for the Generator

Initially the loss is very High for The Generator and in trying to minimize it, The Generator Starts Creating Images that are closer and closer to the dataset it is being Trained Upon

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$$

Equation 2The loss function for the Discriminator

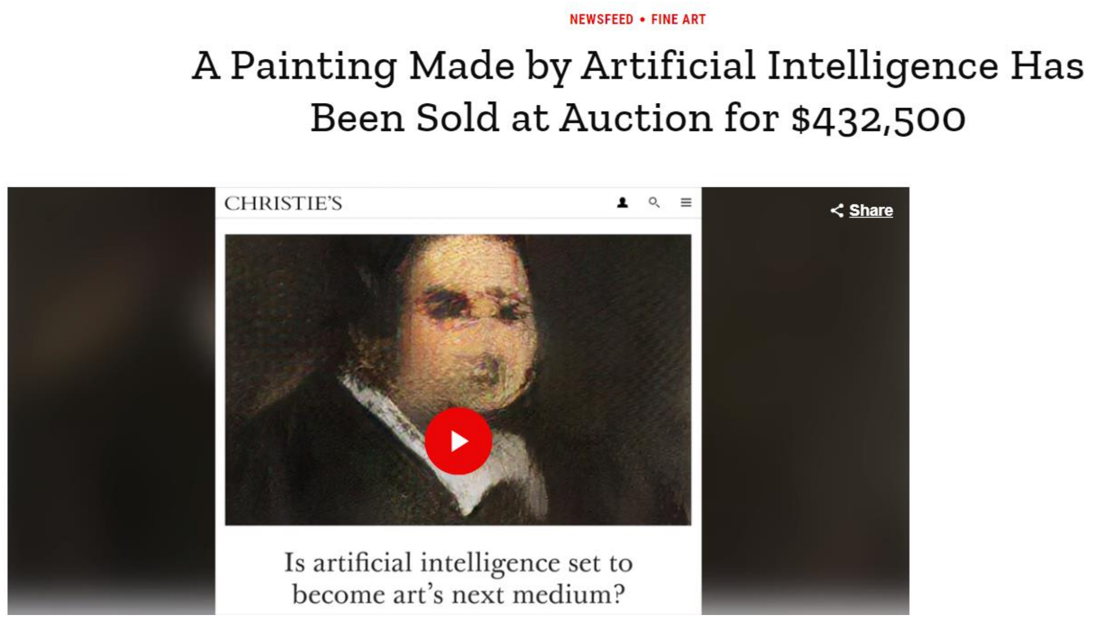
The Discriminator started off easy but it becomes more difficult for it to identify the real Images from the fake ones. An Ideal Equilibrium is Reached When the Generator gets so good at creating images that a Discriminator has no other choice but to take a random guess and all that is achieved by the two loss functions. If a generator produces an image that is meaningless or drastically different from the dataset. The loss Function comes out to be high and in trying to reduce that it generates better images that more accurately represent the features of the dataset**.**

s

USES

1) Advertising:

It has huge commercial applications. Imagine advertisers Just typing in the kind of content they want (Scenes) and have the model automatically build it for them instead of spending thousands of hours of work and saving them lakhs of rupees which they Otherwise have to Spend.


NEWSFEED • FINE ART

A Painting Made by Artificial Intelligence Has Been Sold at Auction for $432,500

Is artificial intelligence set to become art's next medium?

Figure 3The painting was created by a simple GAN

2) Creation of Dataset:

Deep Learning is a field that requires tremendous amount of data and most of the times that data isn't available GANs could be given the smaller dataset and can learn features from them and in turn create even more data points that are unique but represent the features of current dataset.

3) **Super Resolution:**

GANs Could be Used to turn a Low-Resolution Image into a High Resolution thereby Increasing the quality of the Image.

4) **Text to Speech Synthesis:**

Generation of Realistic Synthesized Audio from Text is another very important application of this kind of model where the user enters a text and a waveform is generated automatically. This is useful in applications like Assistant

5) **Image to Image translation:**

Transforming one Image into another is one use of Generative Adversarial Network that has become very popular. Examples include Style Transfer and Scene Conversion. Style Transfer means Painting One Image in the style of Others. Imagine modern day painting in style of Picasso or a day scene automatically converted to a night scene

6) They can be used in medical field, due to their training, they can be used for image analysis and even in making of new drugs.

7) It can be used in technology, imagine someone is saying something about his work and based on his words this will generate a image of what he is explaining and it will save a lot of time, money and work, they have to go through.

1.2) PROBLEM STATEMENT

Generating Images from Text is a very difficult problem that can be approached by using Generative Adversarial Networks and will be extremely useful for content creators wherein they can type a description and have the type of content generated automatically saving them a lot of money and work. Imagine Thinking about a Description and having to draw something that matches the description in a meaningful way. It’s even a difficult task for humans. But Deep Learning Can Understand the Underlying Structure of The Content and might be able to generate that automatically. Thereby eliminating the need of domain expertise.

GANs despite having all the upside for content generation are very difficult to train and Take a lot of time to converge and are unstable doing the training process and in this project, we also try to tackle these problems by modifying the Underlying Structure of the GAN Model

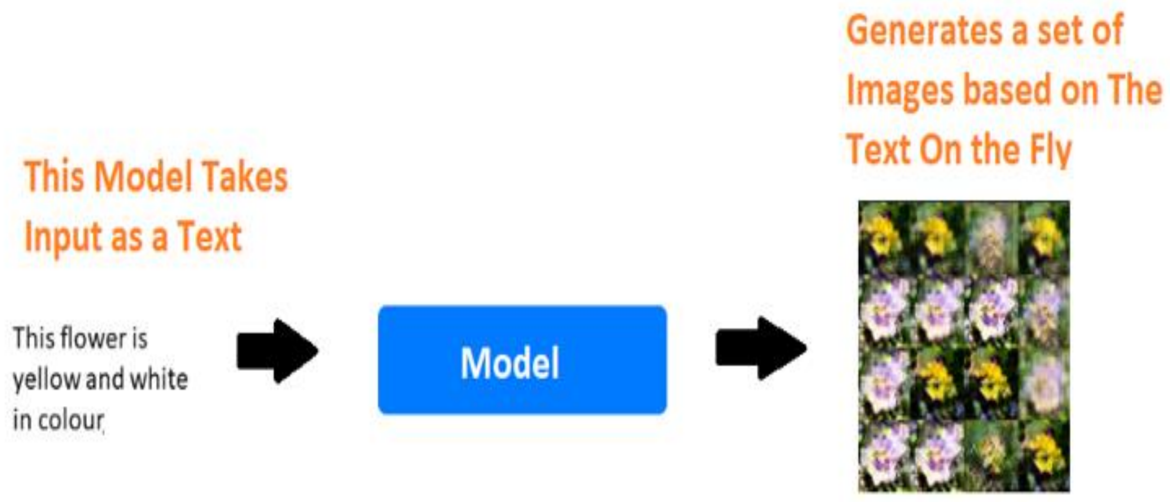

This Model Takes Input as a Text
This flower is yellow and white in colour
Model
Generates a set of Images based on The Text On the Fly

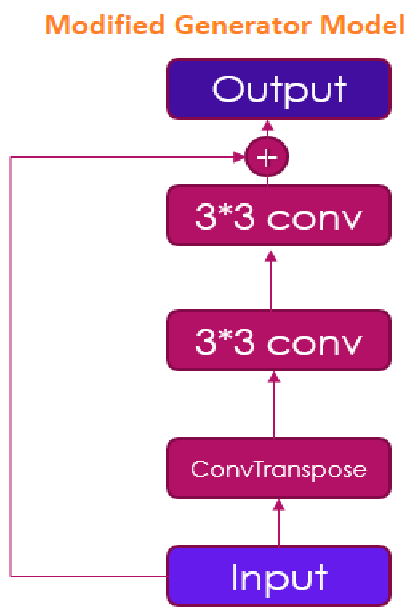
Figure 4 Gan Model

1.3) OBJECTIVE

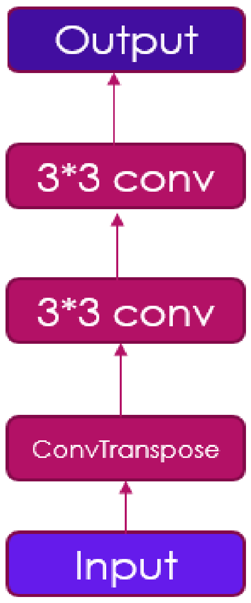
The main objective of this project is to develop a web app in which a text can be inputted and it outputs an image matching the description of the text and in doing so try to improve upon the generator architecture of the Generative Adversarial Networks. By modifying the input to a generator and applying conditions on the input we can create a model that generates images not from noise but from a controlled input. In Our case the Controlled Input Being the text that is Embedded after passing onto another Neural Network

Figure 5

Figure 6



Standard Generator Model Architecture



1.4) METHODOLOGY

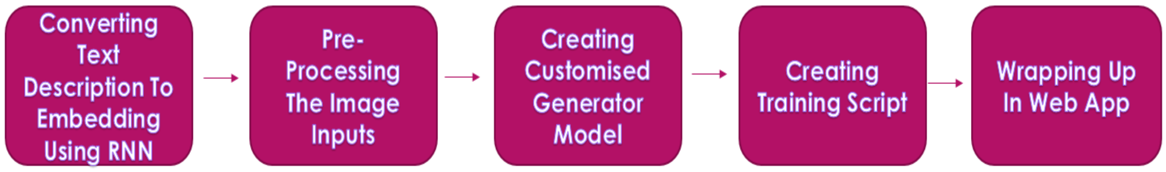
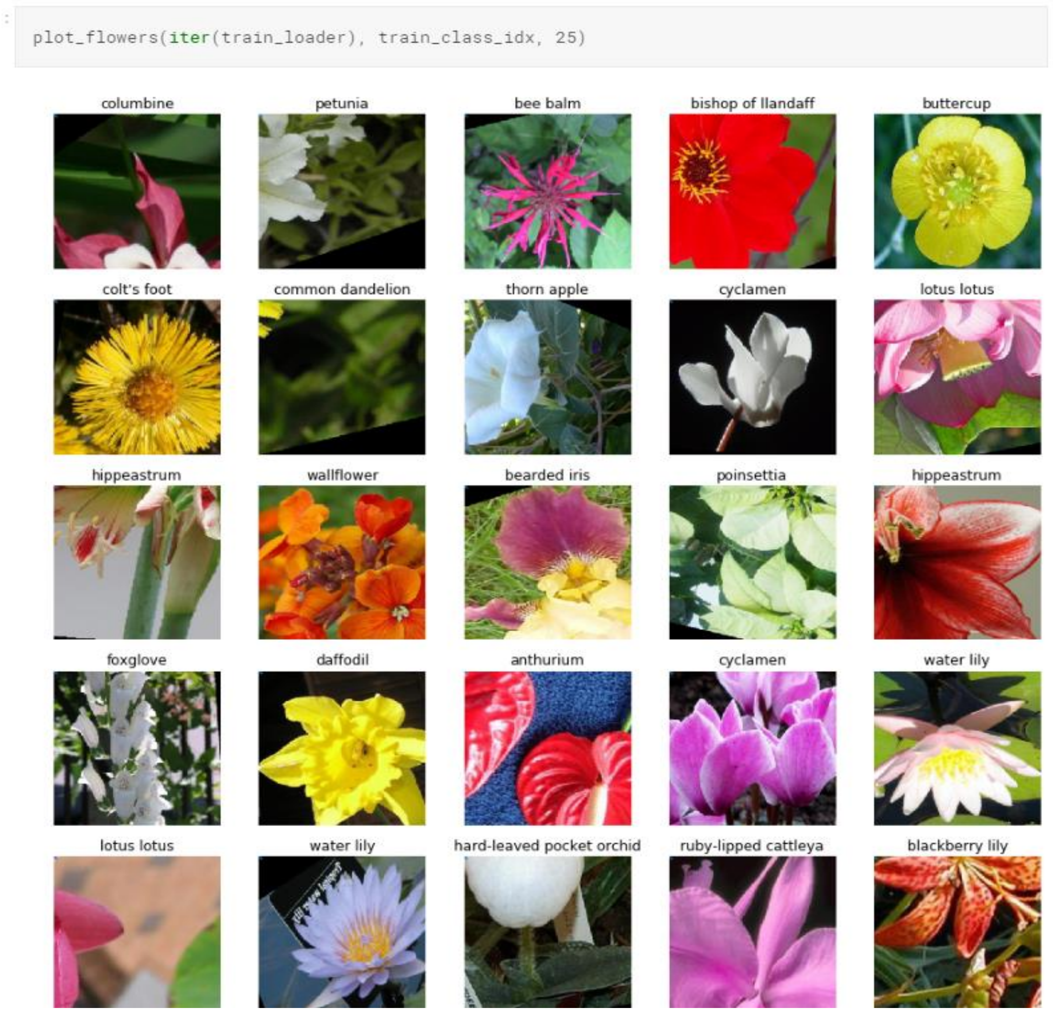

Converting Text Description To Embedding Using RNN
Pre-Processing The Image Inputs
Creating Customised Generator Model
Creating Training Script
Wrapping Up In Web App

Figure 7Methodology

We first start by downloading the **Oxford 102 Dataset** which contains 102 different Categories of flowers and also contains annotation for each image in the form of a text Description.



```
plot_flowers(iter(train_loader), train_class_idx, 25)
```

Figure 8

After this we download on more data set that is CUB dataset that contains 200 bird species with almost 11700 images.

Figure 9 bird dataset

Next, We Begin importing all the packages and the sub packages and splitting it into the training, Validation and testing set. The following packages and libraries are being used to process The Dataset and build the architectures:

- Numpy
- Pytorch

- OpenCV
- Flask

We first start by downloading and pre-processing the dataset. During the pre-processing phase we convert text into embedding and normalize the images so they are ready to be passed onto respective models We then start to build our Customised Generator Model and use a standard Pre-trained Model as the Discriminator After the model Creation we create a training script and take in some best practices in the field of Deep Learning to train the model with stability using our customised Pytorch Trainer. The Final task is to wrap up the final trained model into a Flask Web App so that Testing becomes easy.

1.5 Organization

The steps required for the project is:

Chapter 1

In this we provide a brief introduction to all the work that we did in the project and what the End result looks like. Apart from that we also describe what are the exact steps followed to reach the results.

Chapter 2

In this chapter we did a literature survey of things already existing whether in academia or in various conferences. We got to know our basic Understanding from this.

Chapter 3

In this chapter we go through all the system development process that we did and describe the algorithms used in detail. From LSTM models to generators everything involved in the project has been described under this.

Chapter 4

In this chapter we presented the results of the working model with accuracy of the discriminator and various screenshots of the working model.

Chapter 5

In this we present whatever conclusions that came out of the project and future works that can be done.

Chapter 2: Literature survey

2.1 Books and publication

To understand how to generate content required an extensive study of Deep Learning and Unsupervised learning and to do that we used various Books and Publication along with Some Blogs, Talks and Conferences.

To finally be able to generate content we needed to study how to properly do the following Steps.

- Pre-process the words into embedding. Taking in input as a word and do proper tokenization

- Creating an LSTM Model and generating embedding using the model. Using a LSTM model, we need to convert the long sentence description into a word embedding to pass it into the generator.

- Pre-processing the image and using proper techniques to normalise them. Normalising and augmenting the images using own methods in numpy.

- Studying various Deep Learning Architectures and pros and cons of each. Studying about different architectures like auto encoders and GANs that already exist.

- Studying various methods and already existing architectures like the resnet.

Following books were studied to achieve the following:

1)Natural Language Processing with Python by Steven Bird

In this book I got familiarized with a lot of tools and techniques to process words and the Overall field of Natural Language Processing and the best practices to properly process my Input

2)Neural Networks and Deep Learning by Michael Nielsen

In this book I studied various architectures of Deep Learning that are mostcommonly used today along with proper techniques to train the models and steps to avoid overfitting and underfitting

3)Generative Adversarial Networks by Ian GoodFellow

Ian GoodFellow is the creator of Generative Adversarial Networks and this publication helped me understand what GANs are, how they function and more importantly how can they even do what they do. Also Introduced me to some Problems like Mode Collapse

Chapter 3: System Development

3.1 Text Description to Embedding

The veryfirst step involved in training our model is to convert the text to an embedding. Neural networks work on vectors and numbers and cannotessentially doanything if the input format is a text. So the very first thing we do is utilise a Long short Term Memory (LSTM) network which will take in the input as a pre-processed text afterremoving unnecessary space and improving semantics using standard text pre-processing libraries like spacy and converting the text description into a vector of numbers which is then given as an input to a pre-trained LSTM and the last layer is taken out which is essentially the word embedding that we are looking for.

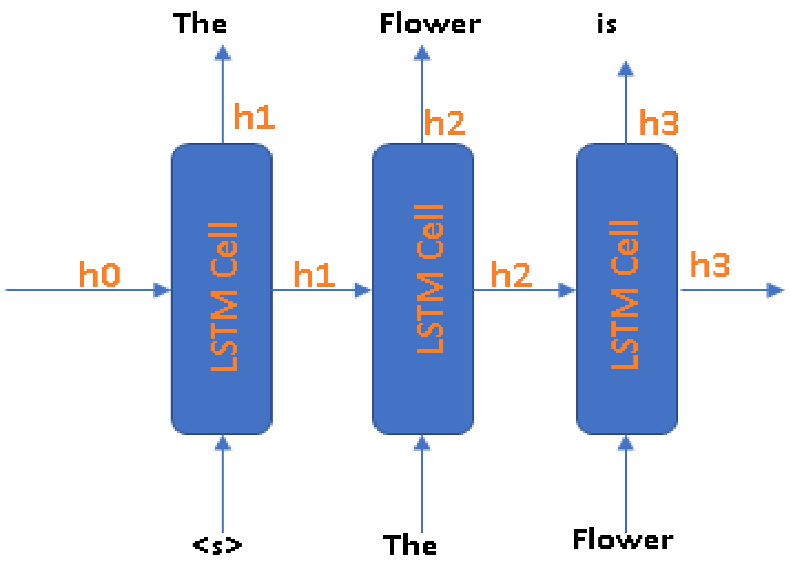


Figure 10Processing Sentences

Why Word Embedding

Why exactly do we need to convert our sentence into an Embedding and not just a one hot Encoded vector.

To Understand that let us take a very simple Example where in once we represent the Words as one hot encoded Vectors and in the other, we Use an Embedding Matrix

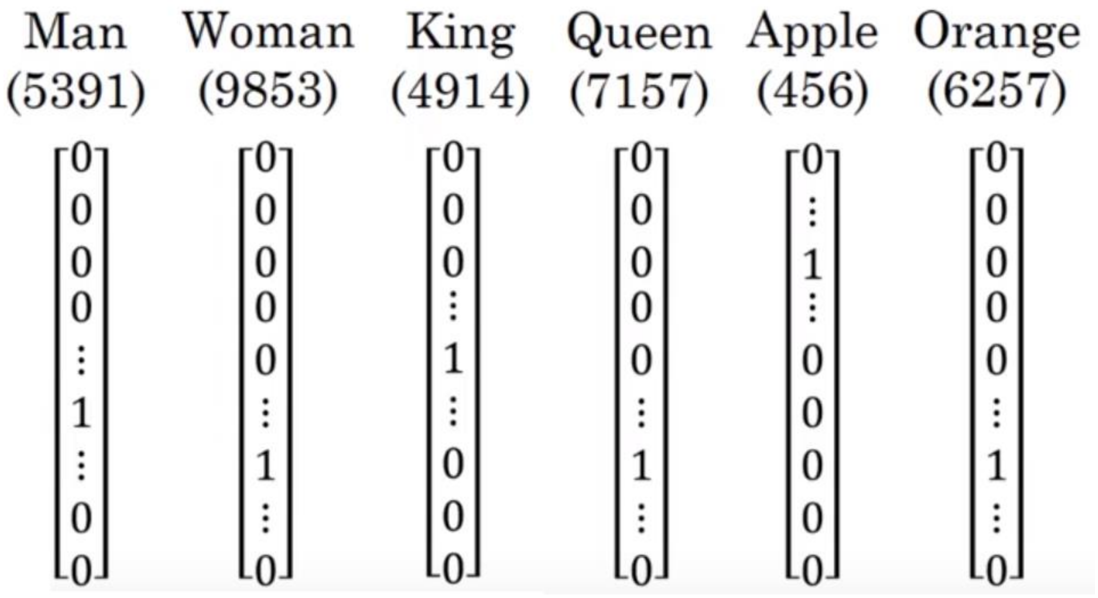


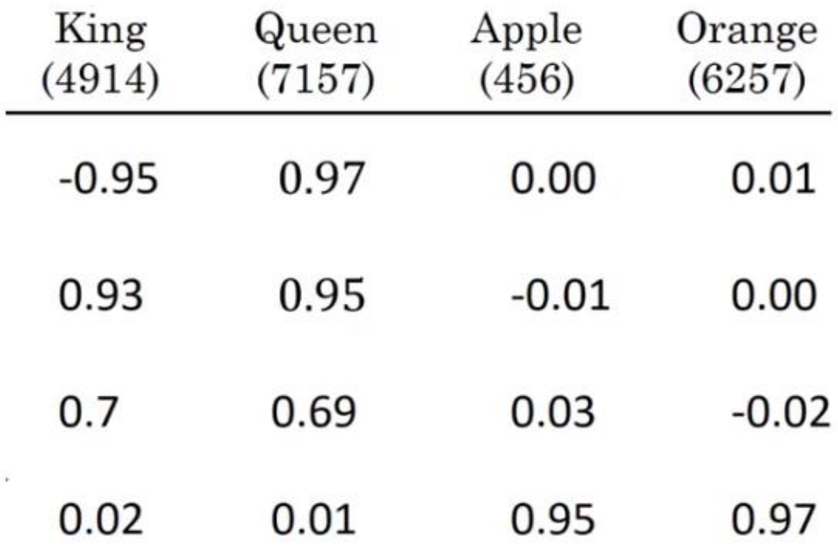
Figure 11One Hot Encoded Vectors

The issue with representing words like this is:

1. Each word is a very high dimensional vector

2. Those Vectors do not have any kind of relation among them that a model can Learn and it becomes very difficult for it to learn when it cannot even understand the relation between words

Now let us Represent them in an Embedding



King (4914)	Queen (7157)	Apple (456)	Orange (6257)
-0.95	0.97	0.00	0.01
0.93	0.95	-0.01	0.00
0.7	0.69	0.03	-0.02
0.02	0.01	0.95	0.97

Figure 12Embedding

When Represented like this the embedding for each vector has a meaning. When Representing these in Euclidean Space we will see that The Two Fruits are closer to each other while the King and Queen are very similar to each other in many respects except one which could be Gender.

It is not pre-decided on what features the model should learn but during the process model Itself decides the best values that reduce the loss and in process it **learns the embedding** That makes more sense to it.

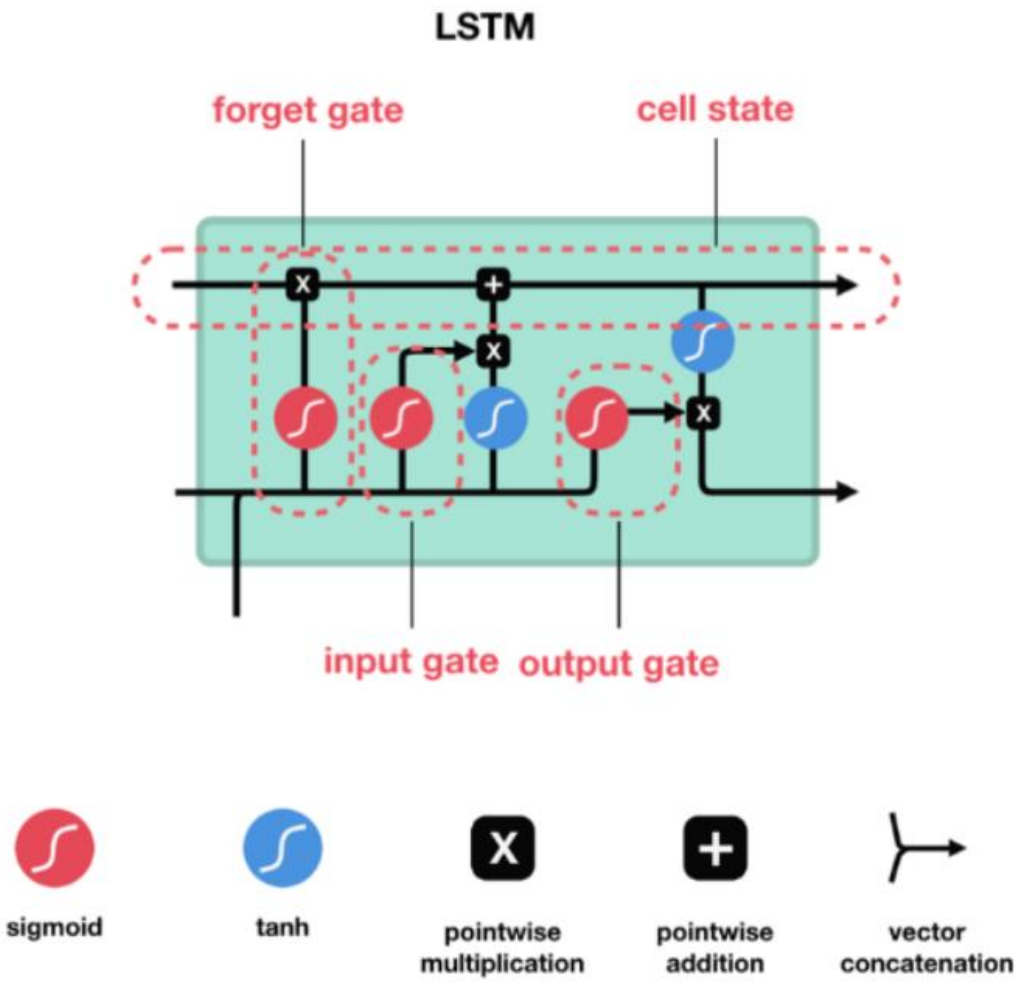


Figure 13LSTM

Long Short-Term Memory Network or LSTM is a type of Recurrent Neural Network that are very good for processing of long sentences because of its ability to learn long term dependencies within the text by modifying the weight of its gate cells.

RNN Typically suffer with a problem that they can’t remember the proper dependencies When processing text whose length is long. To illustrate that problem, we will demonstrate Using a very simple Series.Suppose you are being provided with a series and you have to tell the next number

Example 1) 2->4->6->8

Example 2) 2->4->8

Now in both the series three numbers are common and we know the first series is a Multiple of 2 while the second one is a **power of 2**. But when we pass the numbers to A Model the last input that it gets in both cases is **8** so **how should the modeldistinguish**

Between both the series. It should essentially have previous pattern information combinedwith the current input to output the correct result. But when the sequence gets longer in Length an RNN fails to factor the previous information properly as with no proper mechanism to deal with degrading gradients and at the end it is unable to do any kind of learning

This is the problem that LSTM were built to solve. An LSTM has additional gates that help It properly retains the information throughout the input. However Not all information is Important every time. As we go deeper into the sequence the chances that the next output Depends on a very old input is very less and that is where the **forget gate** of LSTM comes into action. At every Step of input in a sequence an LSTM remodifies the weight of the gates using backpropagation. In a very simple way, it helps it to determine what kind of inputs are important at the current step to predict the next word/element in a sequence. While the **forget gate** determines how much every input it has seen earlier in the sequence is important, **the input gate** helps to decide and update what information to keep and using combination of these it is able to retain information even in a long sentence and able to overcome the problems that arise with Recurrent Networks. The beauty of LSTM is that even a very shallow LSTM model can understand the structure of a sentence very well due to the large number of parameters that it has and its very uniqueconfiguration of the three gates.

3.2 Pre-Processing the Images

Mean and Standard Deviation for Proper Normalisation of Data:

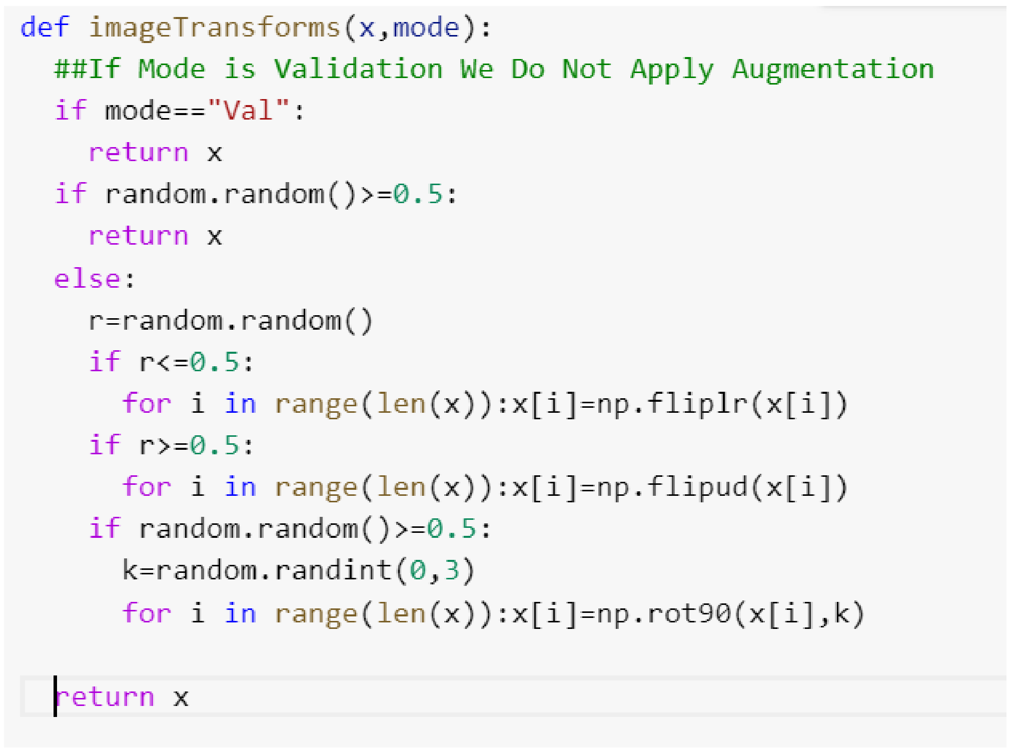
We need to properly process the data before passing to the model as this will Determine the level of accuracy that we can reach Instead of Using the 0 mean and standard Deviation of 1, we can compute the mean and standard deviation for each channel easily. for the current dataset the mean comes out to be [0.484,0.451,0.406] andthe standard deviation comes to be [0.231,0.244,0.412]

Data Augmentation

Data Augmentation will help us to create more data to feed in to the model and help it to generalise well by letting it see the data in various orientations.we create our own transformation using Numpy.Here are some of the Augmentation that we will be implementing

- Random Flip (Horizontal and Vertical)
- Random 90-degree Rotations
- Adding Lightening to the visual channels

Combining the random flip and random rotation we have come up with the 8 dihedral transformations that could be applied to any number of channels and on any kind of dataset as could be seen in the code snippet we first start by creating a function which takes in an input x as a tensor(Matrix Representation of Our Image) and a mode. We do not want to apply these image augmentations when we are in validation mode and testing the entire thing out in training mode, we need to randomly apply these transforms. We use the python's default random number generator to determine what kind of transformations would be randomly applied to the image.



```
def imageTransforms(x,mode):
  ##If Mode is Validation We Do Not Apply Augmentation
  if mode=="Val":
    return x
  if random.random()>=0.5:
    return x
  else:
    r=random.random()
    if r<=0.5:
      for i in range(len(x)):x[i]=np.fliplr(x[i])
    if r>=0.5:
      for i in range(len(x)):x[i]=np.flipud(x[i])
    if random.random()>=0.5:
      k=random.randint(0,3)
      for i in range(len(x)):x[i]=np.rot90(x[i],k)

  return x
```

To flip the image horizontally we first convert our tensor into a numpy array and then use the numpy fliplr function to flip the array horizontally and flipup to flip the array vertically. To rotate the image, we generate a random number k between 0 and 3 which determines how many 90-degree rotations of the array we will do. The following dihedral transformations could be formed after this step

- Horizontal Flip + Any of three 90-degree Rotations
- Horizontal Flip with No Rotations
- Vertical Flip + Any of three 90-degree Rotations
- Vertical Flip with No Rotations

3.3Creating Customised Generator Model

The way a standard Generator Model Works is that it takes in some input and by a series of Up sampling or Deconvolution operations, it creates the Image. The only issue with that is while generating the final output it takes into account is the Information from the previous layer which are very ideal for tasks like Classification and Bounding Box Regression. But when dealing with Image Generation we should also keep into account the original input constraints without much processing along with the Information in the last layer as it will not only help the gradient flow better but also help converge the model faster

```
class ModifiedGenerator(nn.Module):

  def __init__(self,up_in,n_out):

    super().__init__()
    ##Calling The Base Case Constructor Of nn.Module
    self.up_conv=nn.ConvTranspose2d(up_in,n_out)
    ##Operation to Double Along the Width and Height While Reducing
    ##Along The Channel Dimension
    self.drop=nn.Dropout(0.10)
    ##Dropout To Prevent OverFitting

    self.conv1=nn.Conv2d(n_out,n_out*2,kernel_size=3,stride=1,padding=1)
    ##Standard Convolution Operation That Increases
    ##Along the Channel Dimension While Retaining The Width and Height
    self.bn1=nn.BatchNorm2d(n_out*2)
    ##Batch Norm For Self Regularising
    self.conv2=nn.Conv2d(n_out*2,n_out,kernel_size=3,stride=1,padding=1)
    ##Another Convolutional Network To Reduce Along The Channel Dimension
    self.bn2=nn.BatchNorm2d(n_out)

    self.upsample= nn.Upsample(scale_factor=2, mode='nearest')

  def forward(self,up_p):
    identity=up_p
    x=self.drop(F.relu(self.bn1(self.conv1(up_p))))
    x=self.drop(F.relu(self.bn2(self.conv2(x))))
    identity=self.upsample(identity)
    return x+identity
```

In the code snippet above we create our customisedgenerator model from scratch using pytorch .We start off by declaring the class and then initialising the architecture within it.to properly use of pytorch's inbuilt neural network layers we need to use **super** to inherit the properties of the base class we start off by declaring a convtranspose2d which essentially takes in the input embedding and starts by doubling along the height and width and reducing along the channel direction we add a dropout to increase regularization which not only deals with overfitting the model on the training but also helps the model generalise on the input features well this is followed by two convolutional blocks, one doubling along the channel dimension and the other one taking in that input and again reducing it back to original channel dimensions without any change in any other dimensions. This was done as inour practical implementations this trick worked out well now comes the major step of producing the final image. As we stated earlier that we also need to add in the original embedding directly. But the issue with that is embedding has different dimensions altogether. To resolve that we use a simple up sampling operation to bring the embedding to proper dimension before adding it to the output of last layer. In terms of equations we can see it aslet the input be **x** and desired output be **h(x)**

$$\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}.$$

Equation 3

where **F(x)=Conv Blocks+Non Linearities**

Instead of hoping the function to fit to a desired mapping we can specify a residual mapping and let model reduce it and optimise it so as to bring it closer to our**desired output h(x)**

$$\textbf{As } \mathbf{x \rightarrow 0} \quad \mathbf{F(x) \rightarrow H(x)}$$

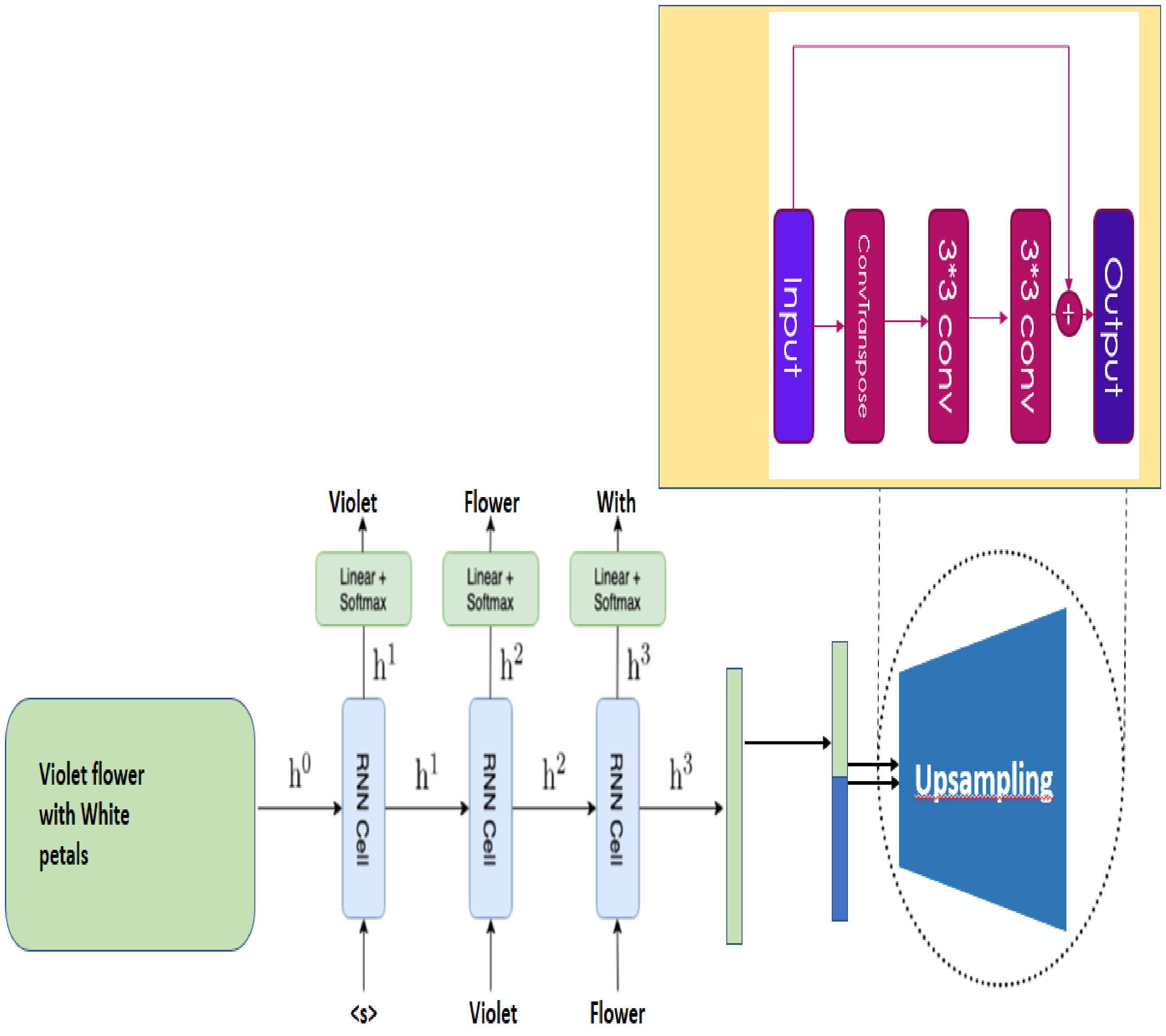


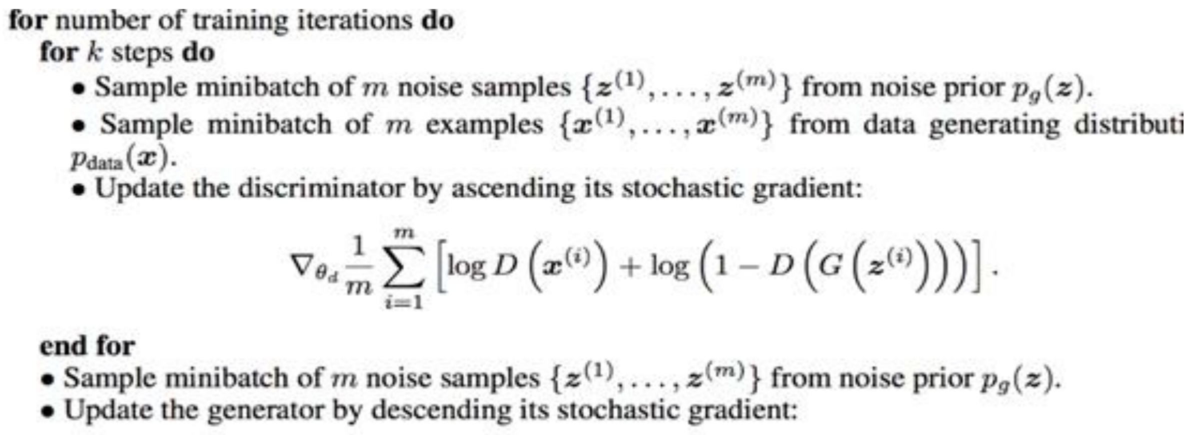
Figure 14 Generation process

3.4 Training the model

The training process of a Generative Adversarial Network is a bit complicated than training a normal Neural Network as it involves training the discriminator and the generator in an alternating fashion.

Step 1: Train the discriminator on original dataset and some random noise to get the discriminator an edge in identifying real images from random noise. This step is very important in the beginning as if the discriminator doesn't already know to some extentwhat the real dataset should look like.When we use the loss function to the generator it will give essentially a lower loss than it should which slows down the initial training. The training eventually stabilises if we do not train the discriminator first properlybut that takes a lot of time. by doing this we are decreasing the training time of the model.

Here is the algorithm:



```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
        • Sample minibatch of $m$ examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distributi
        $p_{\text{data}}(x)$.
        • Update the discriminator by ascending its stochastic gradient:
            $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].$
    end for
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Update the generator by descending its stochastic gradient:
```

Step 2: Afterthe discriminator has been initially trained for a while, we start by making a forward pass through the modified generator model and get in a random image initially and a high loss function, which is then backpropagated throughout the entire network in order to update and fine tune its internal parameters. The generated images are stored in a temporary variable and are passed on to the discriminator in its next phase.

There might be a chance where our gan is not finding the equilibrium between the discriminator and generator. This graph shows us the loss for the discriminator in the blue colour and loss for the generator in the orange colour that is both is heading towards zero in the initial phase of the training. It is possible when gan is not stable.

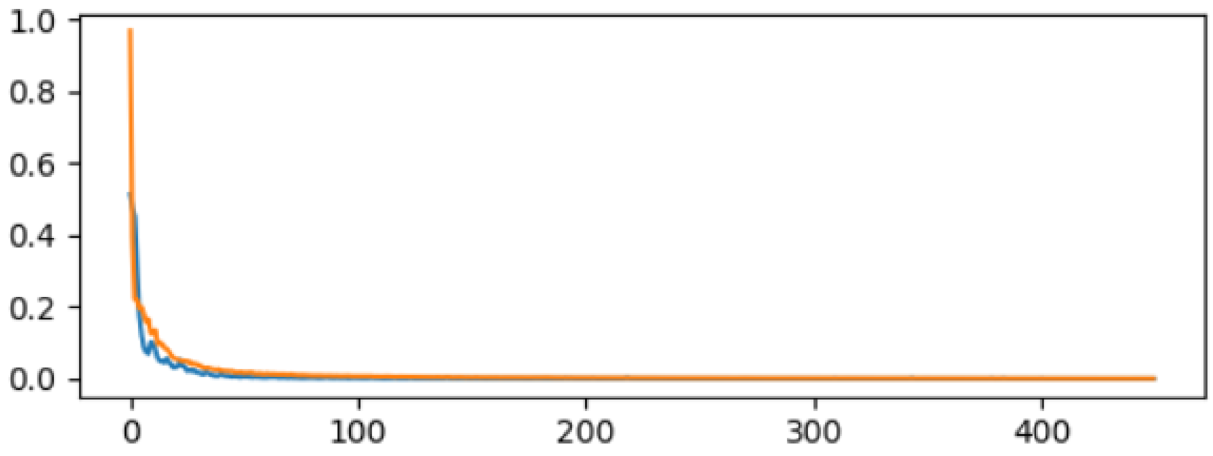


Figure 15convergence failure loss

This graph shows us the accuracy by the discriminator:

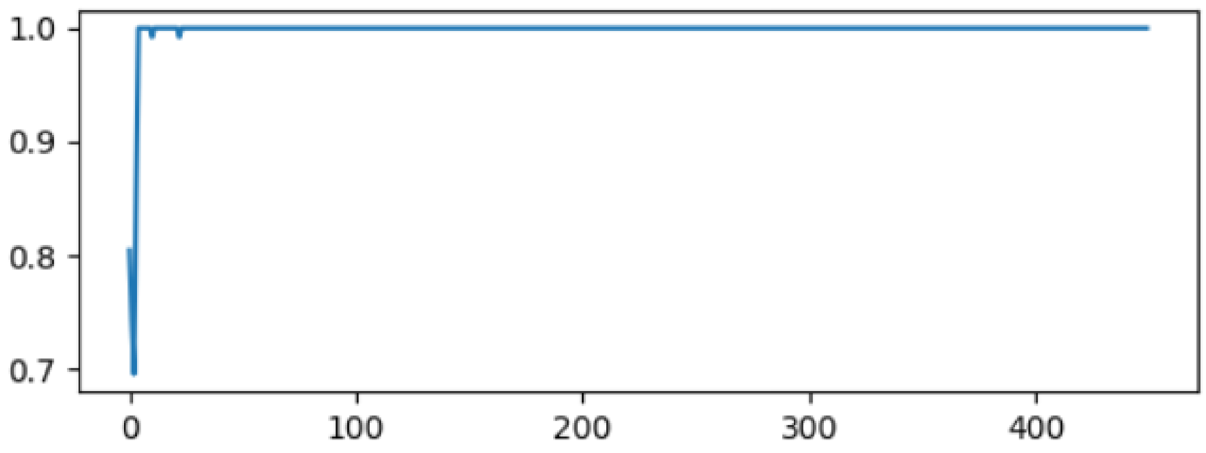
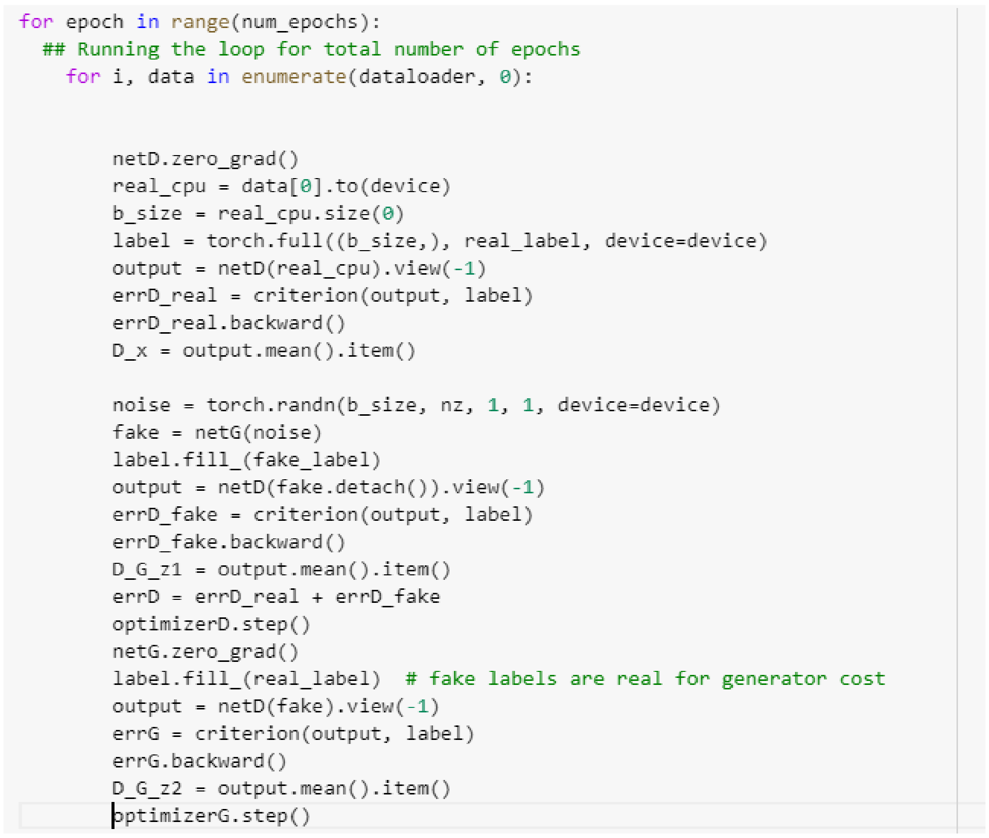


Figure 16 convergence failure accuracy

Here the accuracy of discriminator is 100% which means our gan is perfectly identifying that weather the image is real or fake. This is the most common failure and it is called convergence failure.



```
for epoch in range(num_epochs):
  ## Running the loop for total number of epochs
    for i, data in enumerate(dataloader, 0):


        netD.zero_grad()
        real_cpu = data[0].to(device)
        b_size = real_cpu.size(0)
        label = torch.full((b_size,), real_label, device=device)
        output = netD(real_cpu).view(-1)
        errD_real = criterion(output, label)
        errD_real.backward()
        D_x = output.mean().item()

        noise = torch.randn(b_size, nz, 1, 1, device=device)
        fake = netG(noise)
        label.fill_(fake_label)
        output = netD(fake.detach()).view(-1)
        errD_fake = criterion(output, label)
        errD_fake.backward()
        D_G_z1 = output.mean().item()
        errD = errD_real + errD_fake
        optimizerD.step()
        netG.zero_grad()
        label.fill_(real_label)  # fake labels are real for generator cost
        output = netD(fake).view(-1)
        errG = criterion(output, label)
        errG.backward()
        D_G_z2 = output.mean().item()
        optimizerG.step()
```

We can further Optimise our Training By following Some Simple Steps

1) Proper Learning Rate Scheduler:

A constant learning rate isn’t ideal when training our model as after some epochs we should decrease the learning rate by some amount to get closer to ourdesired value and fine tune the model better. This can be done with the help of a learning rate scheduler which takes in some input function and varies the learning rate by that. now decreasing the learning rate is a very common trend in machine learning. But initially we can increase the learning rate for a few epochs before starting to reduce it since it allows the model to jump high losses very quickly and then slowly fine tune as we can see in the graph it starts with a low learning rate and rises for a few iterations in an epoch and then it starts to fall down. We can set for how many iterations in an epoch will the learning rate rise and for how many iterations will it be decaying for

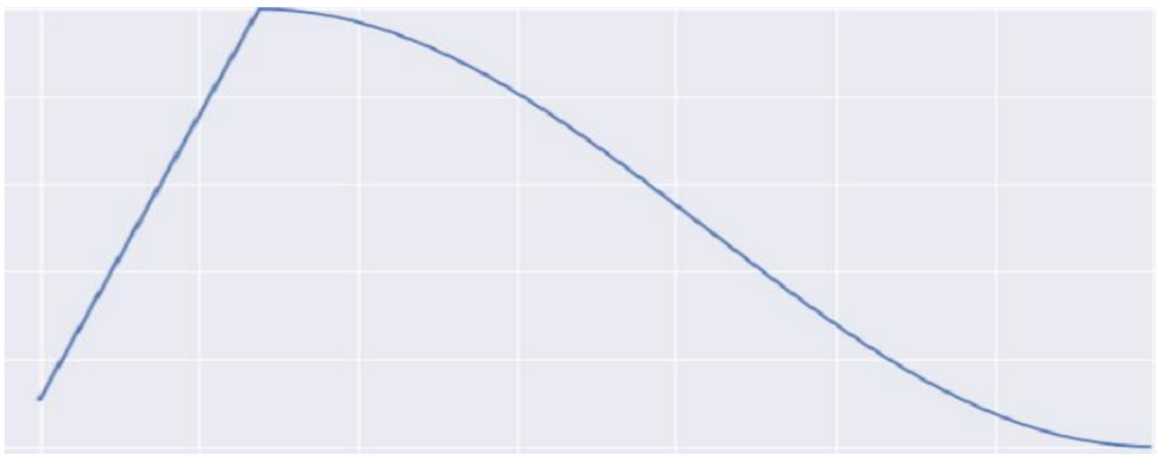


Figure 17Learning Rate Scheduler

2)Using Different Learning Rates for different Layers of the Discriminator:

Training all the layers with same learning Rate can affect the Training since theearlier Layers of The model(closer to the Input) are likely to have learned moregeneral features which we might not want to change that much as compared to later Layers so for earlier layers even a lower learning rate works while higher learning rate are suitable for later Layers

3)Using Weight Decay and Dropout to deal with overfitting:

Weight decay and dropout can be used to deal with overfitting. For our sample dataset a weight decay of 1e-4 and a dropout between 0.10 and 0.23 is working the best.

4)Finding an optimal Learning Rate for The Discriminator:

Instead of randomly starting with any learning rate we can train a subset of data between a learning rate range say 1e-7 to 1e-2 and observe the effect of each learning rate on the loss curve will help us determine the optimum rate. A graph could be plotted for learning rate vs training loss which could be used to determine the best learning rate to begin with. Going even further grouping up few layers we can find out the optimal learning rate for each layer groups and use that to further optimise the discriminator

3.5 Final Web-app Production

We Develop a Web App using flask that presents the user. The user with a web app and an option to input the text and choose a model from various inference Models thatwe have trained. On clicking generate Image, the request is processed in the backend in python and a resultant image using the model is created and we hit another flask endpoint where the generated Image is displayed. The following end points have been created in our existing flask application**:**

1)Home Route:

At the home endpoint or route, we redirect to the page where the user can provide with an input if the model has been loaded successfully without any error. If the chosen model cannot be loaded properly, we redirect to a route describing the error.

2)Generate Route:

After the user successfully enters a text it is pre-processed into a vector and passed on to our LSTM model that generates the word embedding. The embedded vector is then passed to the loaded generator model and is saved onto a location using timestamp as the file name.

3)Result Route:

After the Image has been successfully generated, we redirect the application to a page which displays the generated image.

4)Error Route:

The default route in case any error exists.

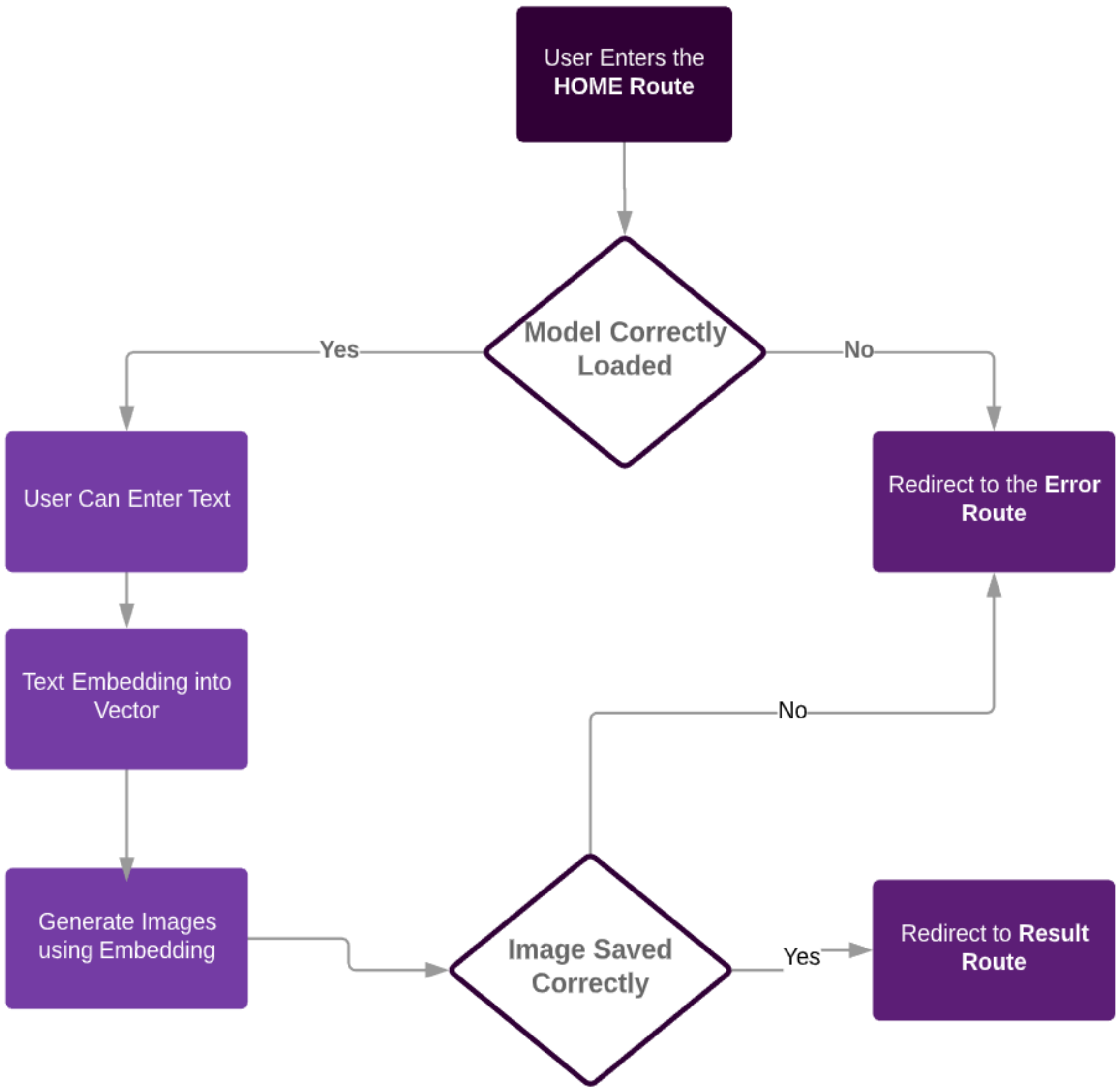

User Enters the **HOME Route**
Model Correctly Loaded
Yes
No
User Can Enter Text
Redirect to the **Error Route**
Text Embedding into Vector
No
Generate Images using Embedding
Image Saved Correctly
Yes
Redirect to **Result Route**

Figure 18Flow graph for The Web App

A sample of the WebApp Could be seen in the Images Below:

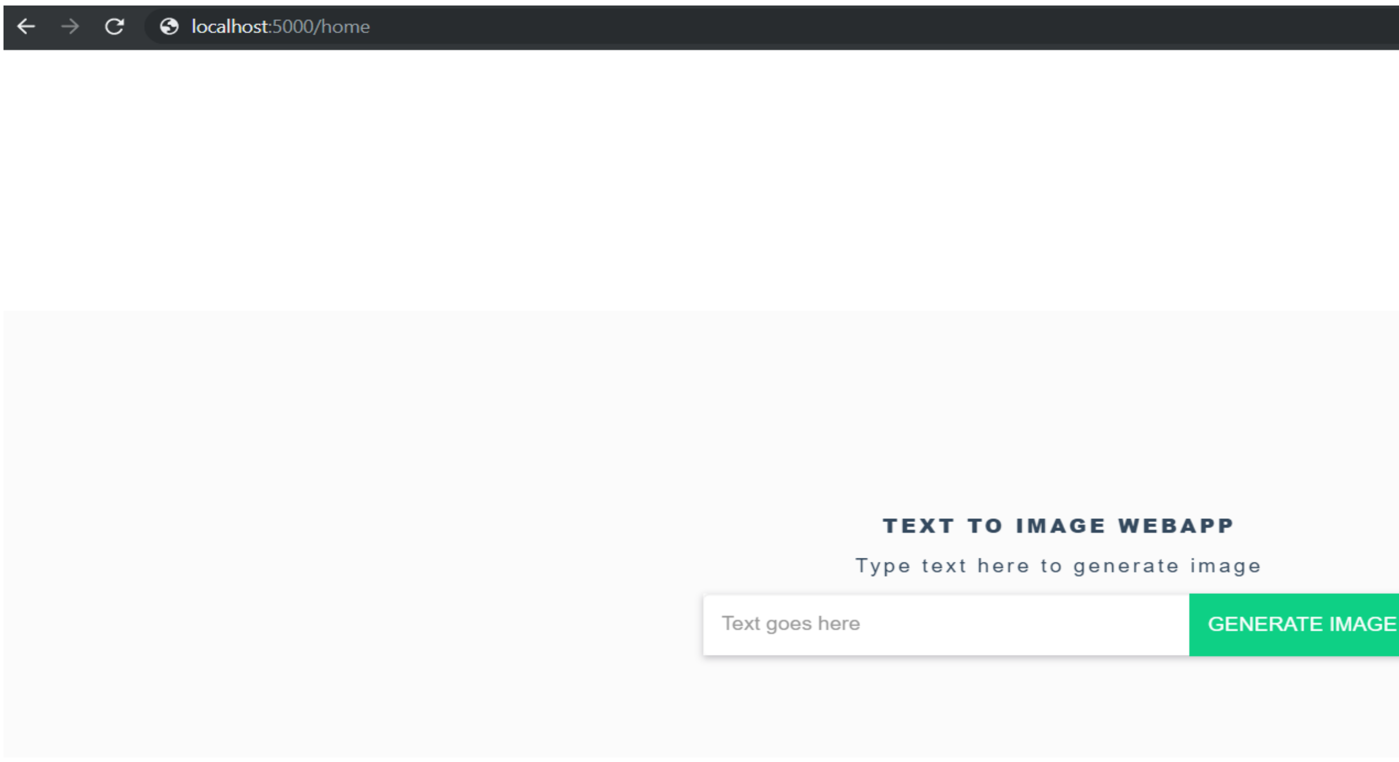


Figure 19 sample of web app

Chapter4: Performance Analysis

4.1 Evaluation:

It is not easy to evaluate the performance of a generative model based on any metric and mostly humans at the end have to decide whether the generated content is good or not and whether or not it holds any particular meaning. However, we can judge the discriminator model in its ability to distinguish real images from fake ones. now compared to ordinary convolutional models where high accuracy means better results it isn't true in the case of generative models. If the discriminator has very high accuracy in distinguishing real images from fake ones then that implies generator hasn't done a very good job in creating images that represent the dataset well. In the situation of a perfect equilibrium the discriminator should have an accuracy of 50% that is it has to take a random guess to determine whether the generated image is fake or not implying the generator has created images so good that are indistinguishable from the original images. The closer an accuracy is to 50% the better task the generator has done in creating images.

Loss graph:

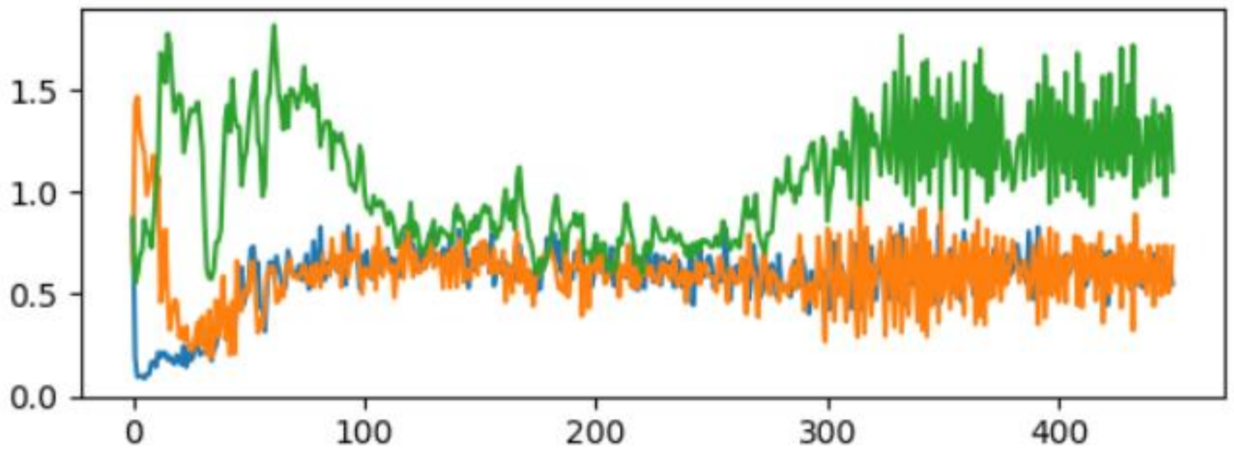


Figure 20loss graph

The above graph shows us the loss that is the discriminator loss for the real images in blue colour and discriminator loss for fake images in orange colour and generator loss for the generated images in the green colour.

This is an expected loss during this training and it will stabilize after around 100 to 300 epochs. The discriminator loss for the real and the fake images is approximately 50% and the generator loss for the generated images is between to 50% to 70 %.

Accuracy graph:

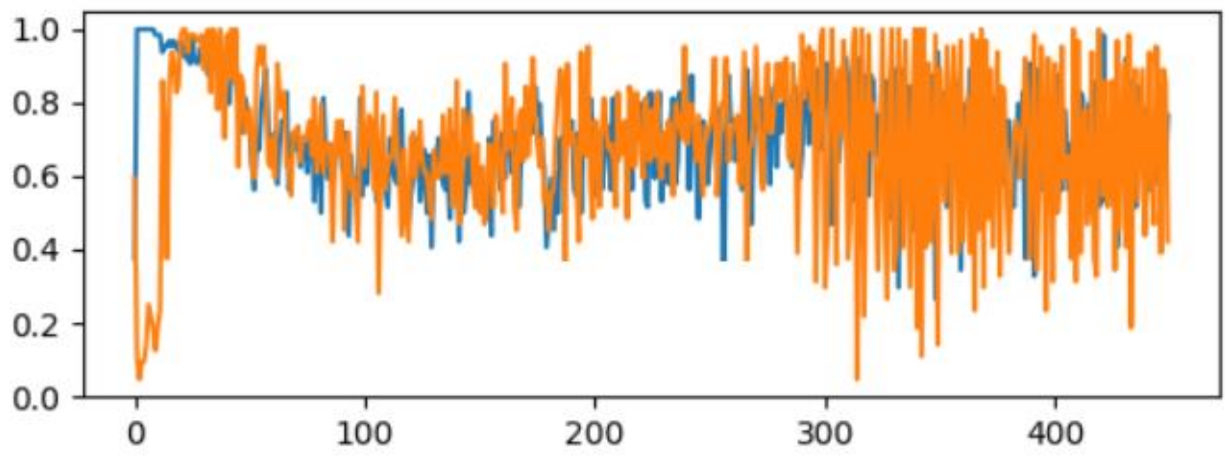


Figure 21 accuracy graph

This graph shows us the accuracy by the discriminator for the real images that is in blue color and for the fake images in orange color.

This GAN model will get stabilized in 100 to 300 epochs and after that it will give us an accuracy approximately in between 70% to 80% and it will remain stabilized after that.

4.2Training phase snapshots:

In the starting as we get those images that have more or less noise but without any meaning.

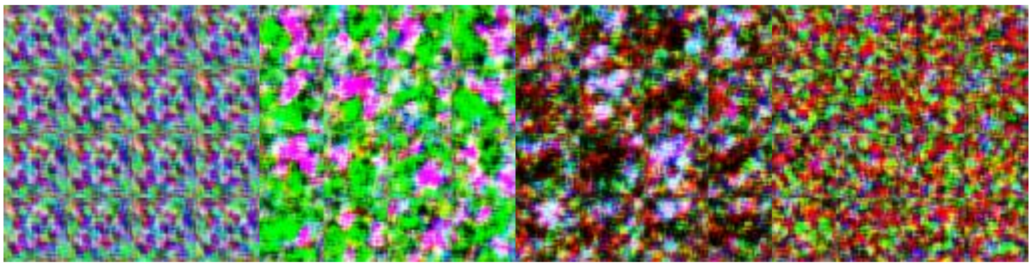


Figure 22

After about **55 epochs** the results starts improving a little bit and the noise starts making some sense.

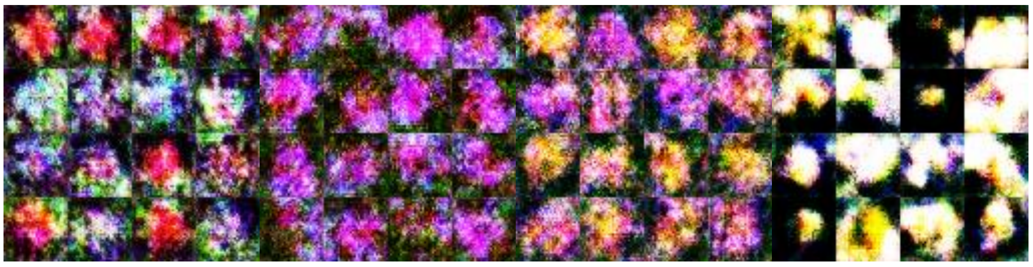


Figure 23

And slowly we start moving towards our final Model

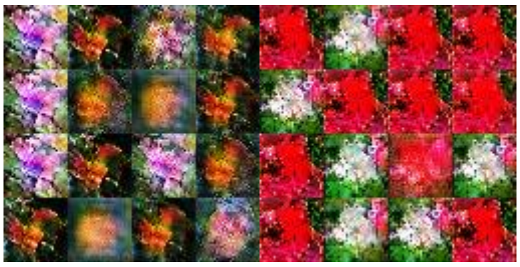


Figure 24

Here are some text descriptions with our first data set and their results.

A flower that is yellow and white in colour:

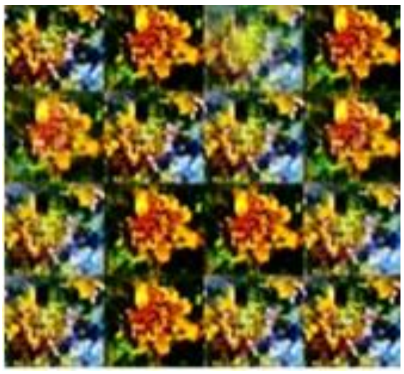


Figure 25

A flower having pink petals:

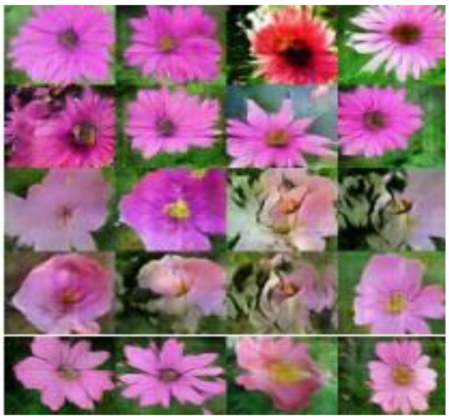


Figure 26

A flower with white petals and purple and white anthers:

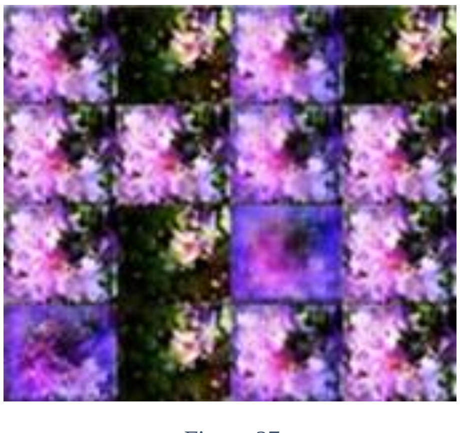


Figure 27

A flower with maroon petals and green leaves:

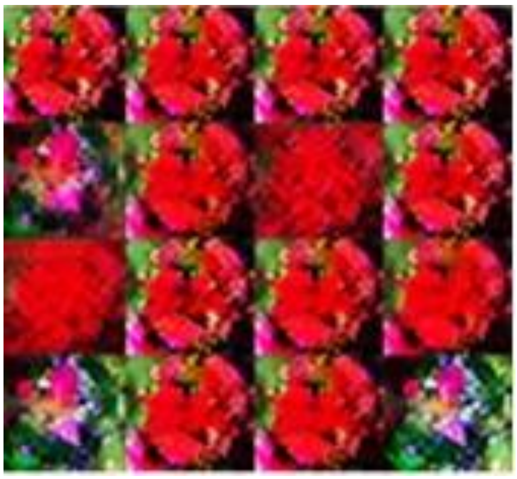


Figure 28

A flower having pink petals and they are curled upwards:

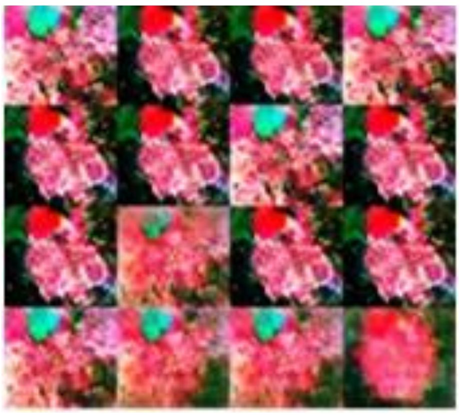


Figure 29

Finally, we stop the training and load up our Inference model and here are the results after using the web app.

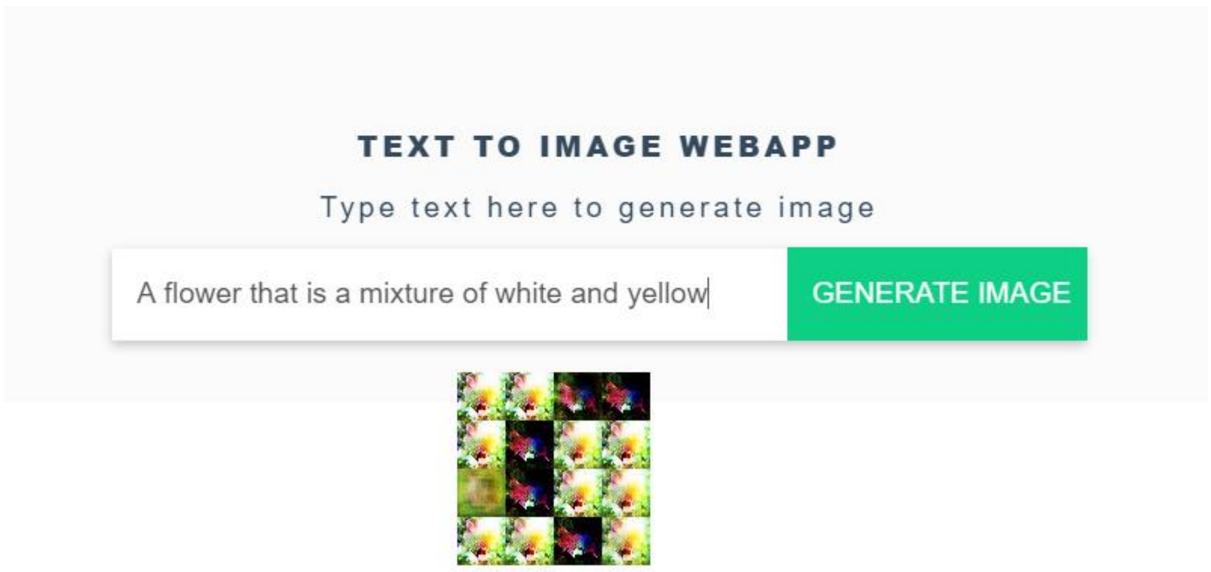
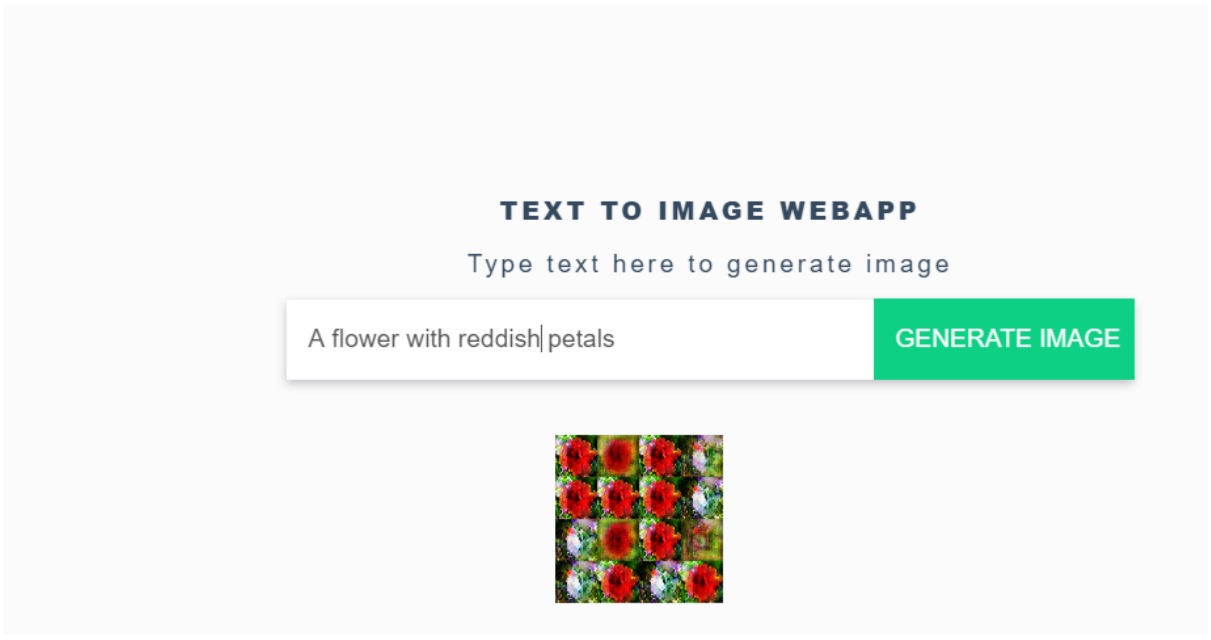

TEXT TO IMAGE WEBAPP
Type text here to generate image
A flower that is a mixture of white and yellow
GENERATE IMAGE

Figure 30 text to image


TEXT TO IMAGE WEBAPP
Type text here to generate image
A flower with reddish petals
GENERATE IMAGE

Now these are the results we get from our second data set that is of birds:

80 random pics of birds:

Figure 31

A bird with a red head:

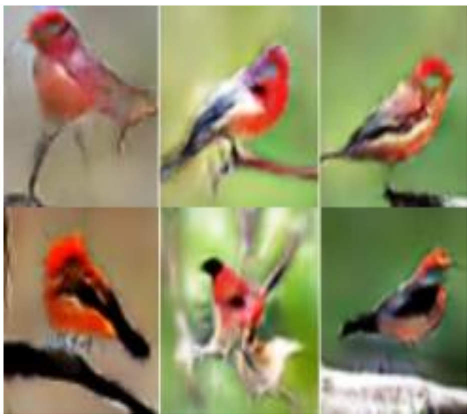


Figure 32

A bird that is yellow in colour:

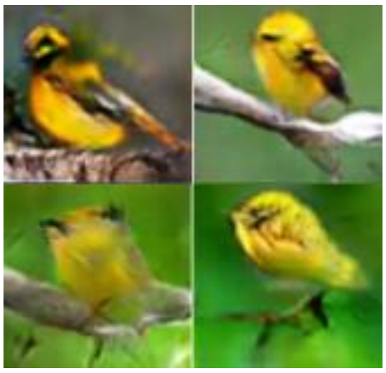


Figure 33

A bird which is flying and it has white wings:

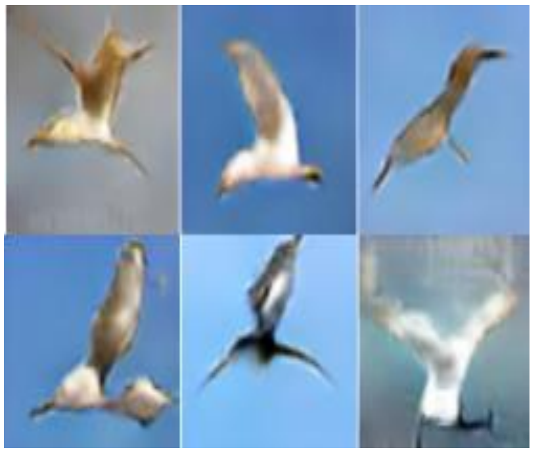


Figure 34

Chapter5: CONCLUSIONS

5.1 Conclusion:

In this project we have created a web app that can take in text description of a flower and bird and generate images based on that. And while doing that we have modified the generator architecture in such a way that we have reduced the training time of GAN.

52. Future Scopes:

For the Future Work we can take the 64*64 image that we have obtained through our results and using Super Resolution convert it into a 256*256 image which can again be accomplished with the help ofGAN. We can make two stages of the GAN that is the first stage will give us image that is 64*64 and second stage will give us 256*256 image.

References

[1] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. In ICLR, 2017.

[2] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Neural photo editing with introspective adversarial networks. In ICLR, 2017.

[3] T. Che, Y. Li, A. P. Jacob, Y. Bengio, and W. Li. Mode regularized generative adversarial networks. In ICLR, 2017.

[4] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In NIPS, 2016.

[5] E. L. Denton, S. Chintala, A. Szlam, and R. Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In NIPS, 2015.

[6] C. Doersch. Tutorial on variational autoencoders. arXiv:1606.05908, 2016.

[7] J. Gauthier. Conditional generative adversarial networks for convolutional face generation. Technical report, 2015.

[8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In NIPS, 2014.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In CVPR, 2016.

[10] X. Huang, Y. Li, O. Poursaeed, J. Hopcroft, and S. Belongie. Stacked generative adversarial networks. In CVPR, 2017.

[11] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML, 2015.

[12] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In CVPR, 2017.

[13] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In ICLR, 2014.

[14] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. In ICML, 2016.

[15] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. In CVPR, 2017.

[16] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In ECCV, 2014.

[17] E. Mansimov, E. Parisotto, L. J. Ba, and R. Salakhutdinov. Generating images from captions with attention. In ICLR, 2016.

[18] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. In ICLR, 2017.

[19] M. Mirza and S. Osindero. Conditional generative adversarial nets. arXiv:1411.1784, 2014.

[20] A. Nguyen, J. Yosinski, Y. Bengio, A. Dosovitskiy, and J. Clune. Plug & play generative networks: Conditional iterative generation of images in latent space. In CVPR, 2017.

[21] M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In ICCVGIP, 2008.

[22] A. Odena, C. Olah, and J. Shlens. Conditional image synthesis with auxiliary classifier gans. In ICML, 2017.

[23] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In ICLR, 2016.

[24] S. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, and H. Lee. Learning what and where to draw. In NIPS, 2016.

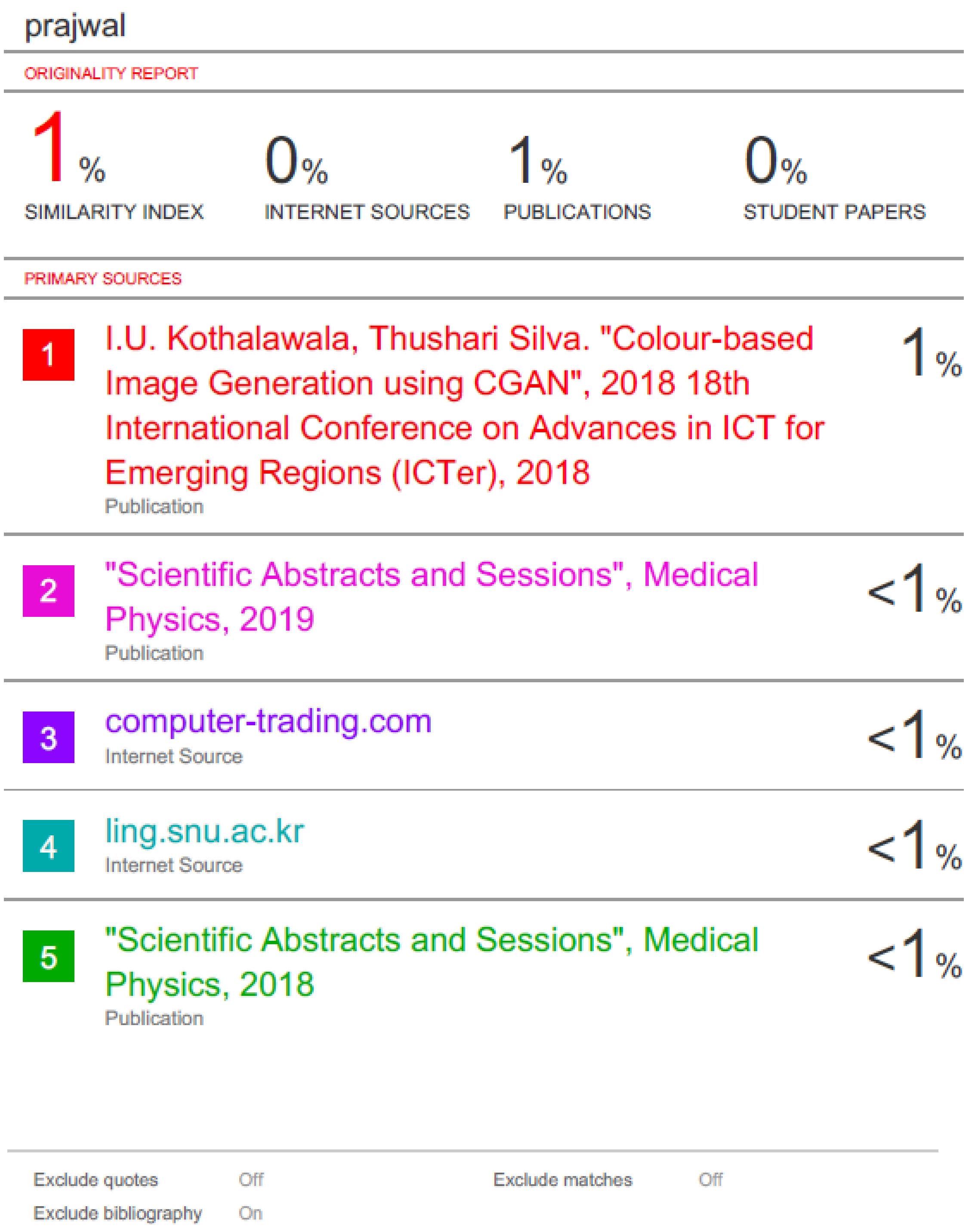
[25] S. Reed, Z. Akata, B. Schiele, and H. Lee. Learning deep representations of fine-grained visual descriptions. In CVPR, 2016.

[26] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text-to-image synthesis. In ICML, 2016.

[27] S. Reed, A. van den Oord, N. Kalchbrenner, V. Bapst, M. Botvinick, and N. de Freitas. Generating interpretable images with controllable structure. Technical report, 2016.

[28] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In ICML, 2014.

[29] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In NIPS, 2016



prajwal

ORIGINALITY REPORT

1%	0%	1%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	I.U. Kothalawala, Thushari Silva. "Colour-based Image Generation using CGAN", 2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer), 2018 Publication	1%
2	"Scientific Abstracts and Sessions", Medical Physics, 2019 Publication	<1%
3	computer-trading.com Internet Source	<1%
4	ling.snu.ac.kr Internet Source	<1%
5	"Scientific Abstracts and Sessions", Medical Physics, 2018 Publication	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		

.