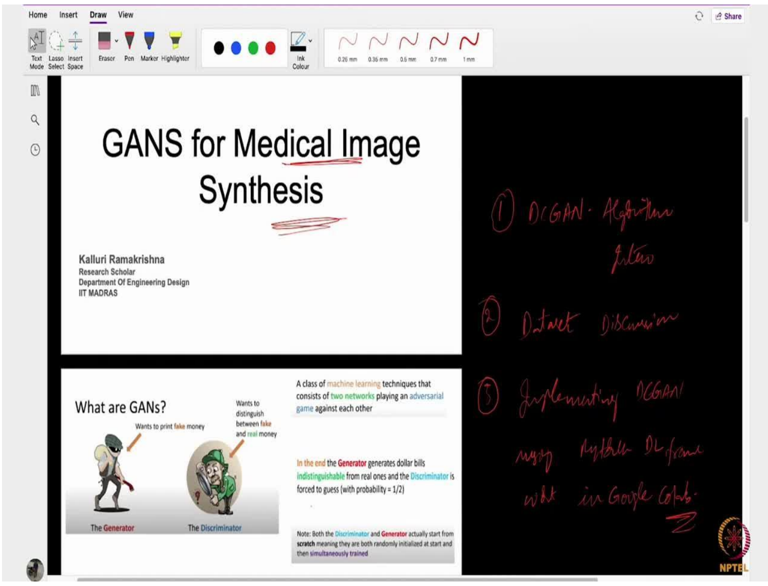
Medical Image Analysis
Professor Kalluri Ramkrishna
Department of Engineering Design
Indian Institute of Technology, Madras
Lecture 50

GAN Final Demo

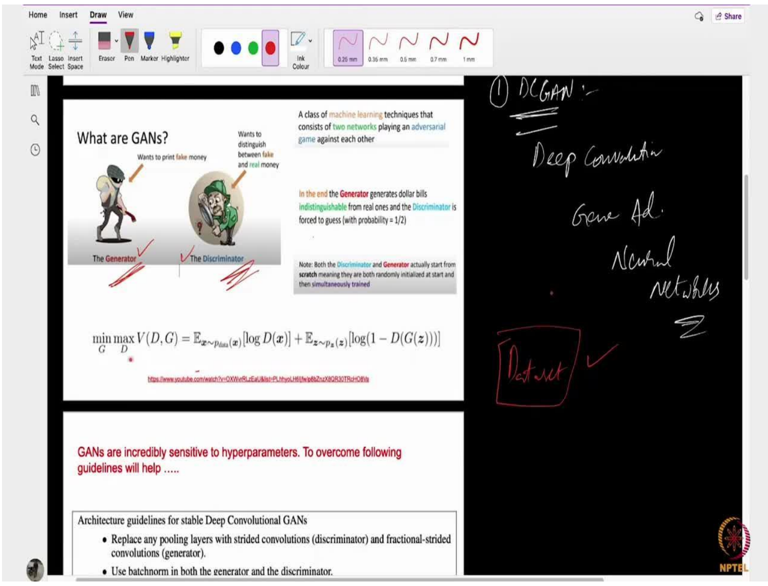
(Refer Slide Time: 00:15)



Hi everyone, I am Ramakrishna Kalluri one of the TAS of Medical Image Analysis course I am research scholar under Doctor Ramkrishnamurthi in the Department of Engineering Design at IIT Madras. By the end of this lecture, you will get to know how to use GANS for medical image synthesis. Mostly I will discuss on implementing DCGAN algorithm or generating images having data distribution almost close to the given training data distribution mainly I have divided the entire lecture into three parts.

First one is DCGAN algorithm introduction. Second one is data set discussion that we are going to use to implement this algorithm, dataset discussion and the third part is implementing DCGAN algorithm using PyTorch deep learning framework in Google colab.

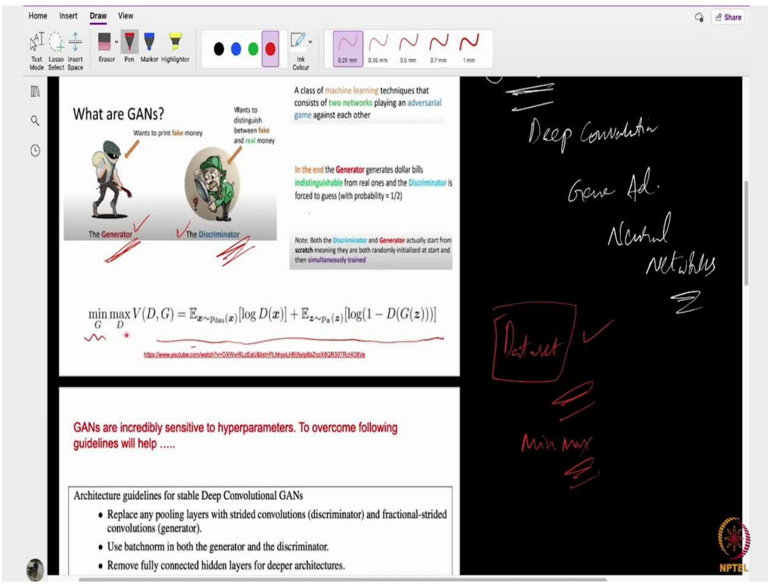
(Refer Slide Time: 01:36)



Let us take the first part that is DCGAN Introduction this you can call this Deep Convolutional Generative Adversarial Neural Networks. GANS mainly consists of two networks first one is generator and the second one is discriminator usually, in many machine learning or deep learning problems, we will be given some data set.

Let us take such a case and discuss the importance of generator and discriminator in GANS. Generator always try to generate fake or artificial images that are almost indistinguishable from the given real data training data that means, the probability distribution of the generated or fake images is almost close to the probability distribution of the real data distribution.

(Refer Slide Time: 02:48)

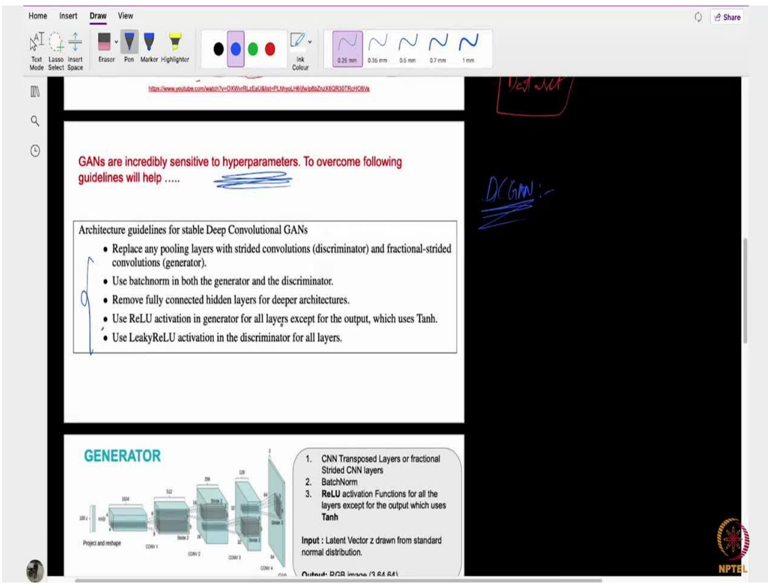


Now, coming to discriminator, the discriminator's task is to just distinguish the real and fake images whatever the images generated by the generator are to be rejected by the discriminator continuously. So, likewise they compete each other and will reach to a point where discriminator will not be in a position to distinguish between real and fake, that means at that point fake images generated by the generator are almost close to the real images.

This is the loss function or you can call objective function that is used by the GANS with respect to generator this function will be minimized whereas, with respect to the discriminator this function will be maximize. That is why people call this one as min max algorithm. Sorry, min max objective function that is the basic idea of GANS.

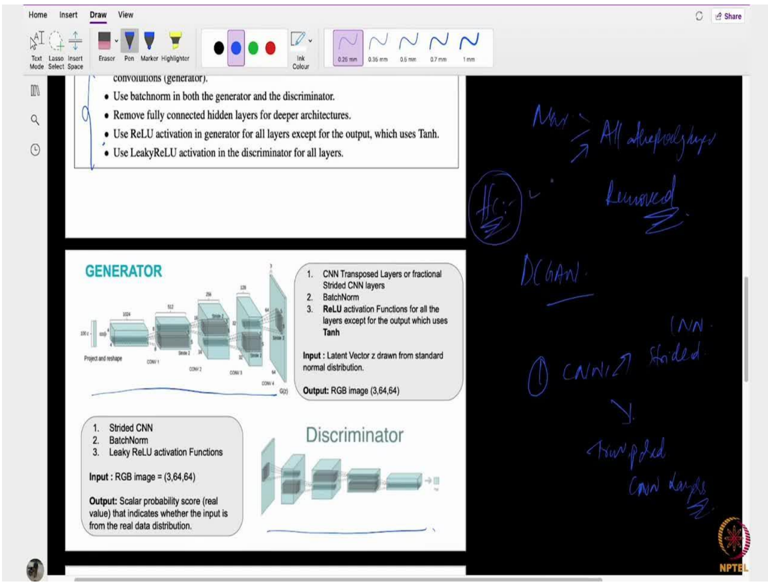
In summary, generator always try to generate the fake images that are almost similar to the data that is given in the data set or we can say training data. Discriminator's task is to always to reject the images generated by the generator whereas, to accept the images that are generally present in the real images or given data set.

(Refer Slide Time: 04:28)



GANS are incredibly sensitive to hyper parameters to overcome following guidelines will help in implementing the DCGAN. These are the 5 guidelines suggested by the others in the paper deep convolutional generative adversarial neural networks.

(Refer Slide Time: 04:48)

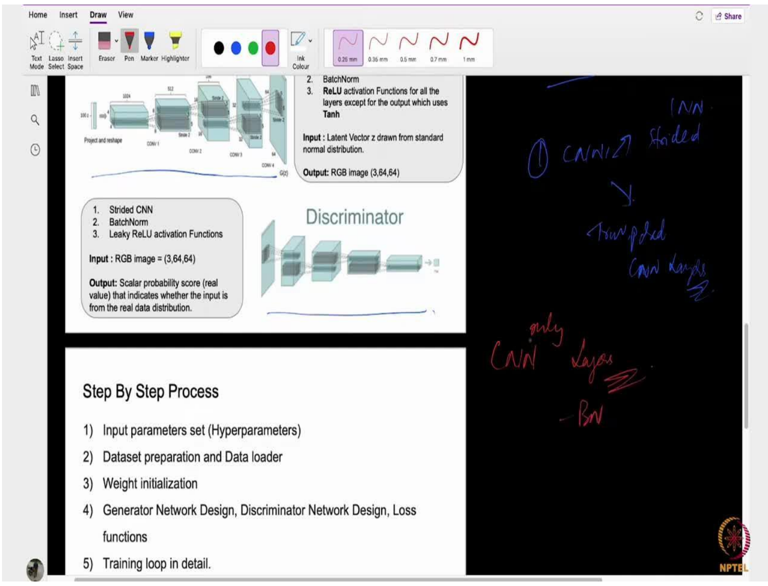


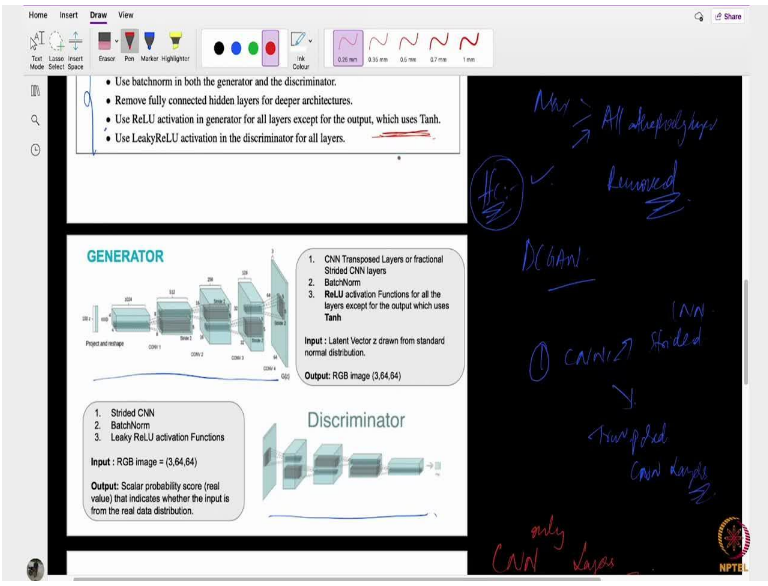
Now, if you observe clearly in this entire DCGAN algorithm, generator will be like this, whereas discriminator will be like this. If you observe clearly go to the first point replace any pooling layers with strided convolutions and fractional strided convolutions with generator.

Here if you observe generators and discriminators will only contain only CNN layers in discriminator, the CNN are strided convolutions only, strided CNN layers whereas in generator we will use fractional strided convolution or in other way transposed convolutions transposed CNN layers. That means all the max pooling layers or minimum pooling layers or average pooling if any of the all the pooling layers were removed all the pooling layers are removed all the pooling layers removed.

Now get back to using batch normalization in both generator and discriminator we will use batch normalization in both generator and discriminator. Next thing is remove fully connected layers also from deeper architectures that means, we will avoid using fully connected layers also.

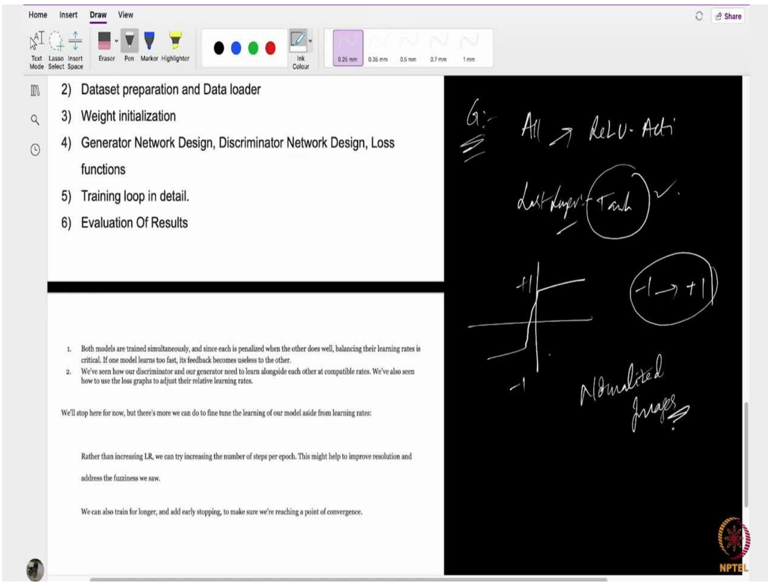
(Refer Slide Time: 06:26)





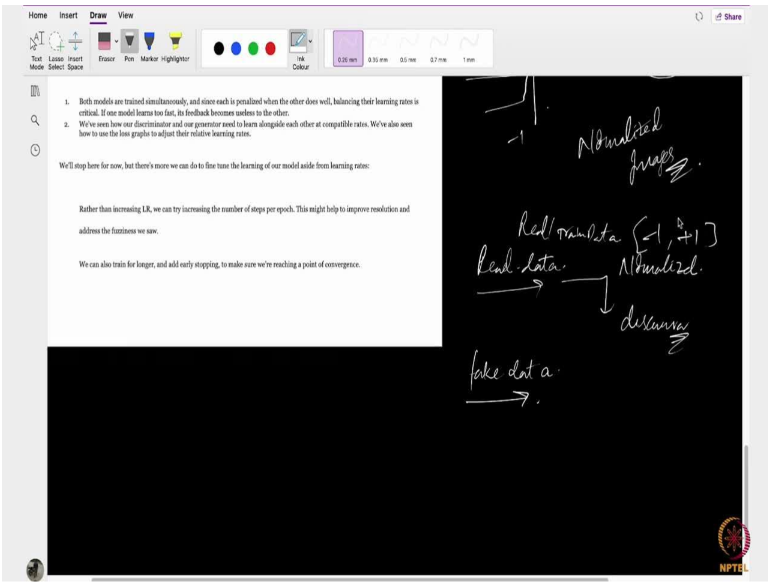
So, the summary is only CNN layers are present only CNN layers are present in the network and bachelor normalization is present everywhere both generator and discriminator and the last thing is we will use ReLU activation function in generator for all the layers except for the output which uses tan(h).

(Refer Slide Time: 06:47)



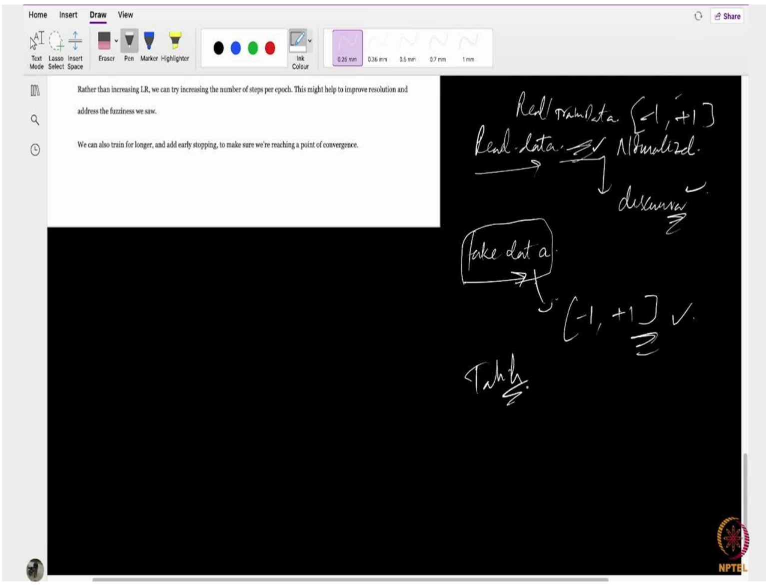
In generator all the other layers use ReLU activation function except for the lost layer where we use the Tan h function. The ReUL range of this tan(h) activation function is -1 to +1 that means, if the lost layer of the entire output at the last layer generated will consist of all the pixel values will in between -1 to +1 more or less like normalized images.

(Refer Slide Time: 07:29)



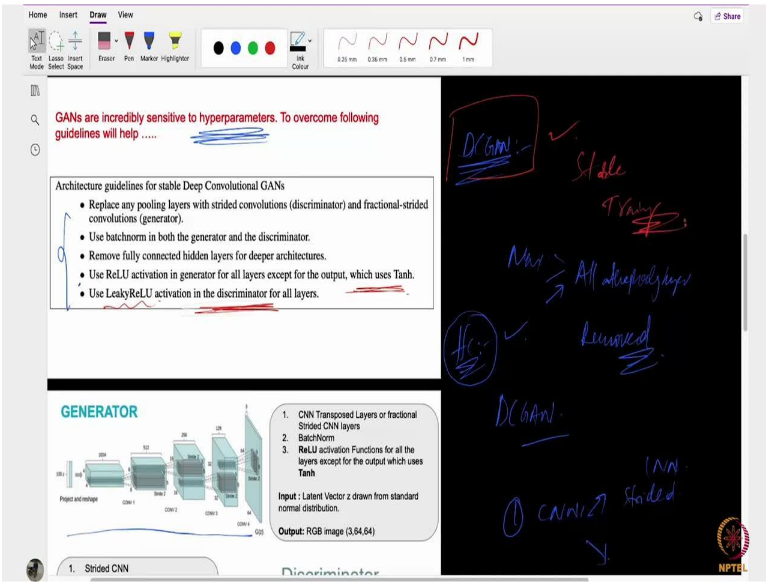
Why we generate normalized images because real data are the data set of the training data or the data that is generated by the this is generated by the generator and this is the original ReLU train data, before passing it before passing the real data to discriminator we have normalized all the training data images nominalized all the training data images to -1 to +1 range.

(Refer Slide Time: 08:04)



So, now, whatsoever the images that are generated by the generator also to be present in the same format as that of the data that is being fed into discriminator that means, the generator data by the generator also should be should contain all the all its pixel values should be in the range of -1 to +1 that is why we have used tan(h) function at the end of lost layer in the generator.

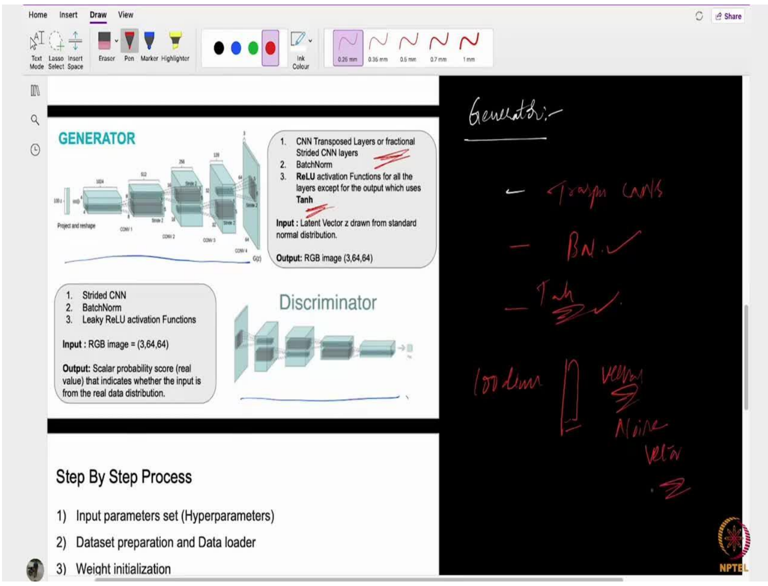
(Refer Slide Time: 08:41)



And the last point is use leaky ReLU function in the discriminator for all the layers that have been in discriminate section all the layers will contain leaky ReLU activation function.

So, these are the 5 guidelines one has to keep in mind while implementing this DCGAN algorithm. So, that the network will be somewhat less serious due to the hyper parameters so, that we can expect a stable training, stable training that is the summary.

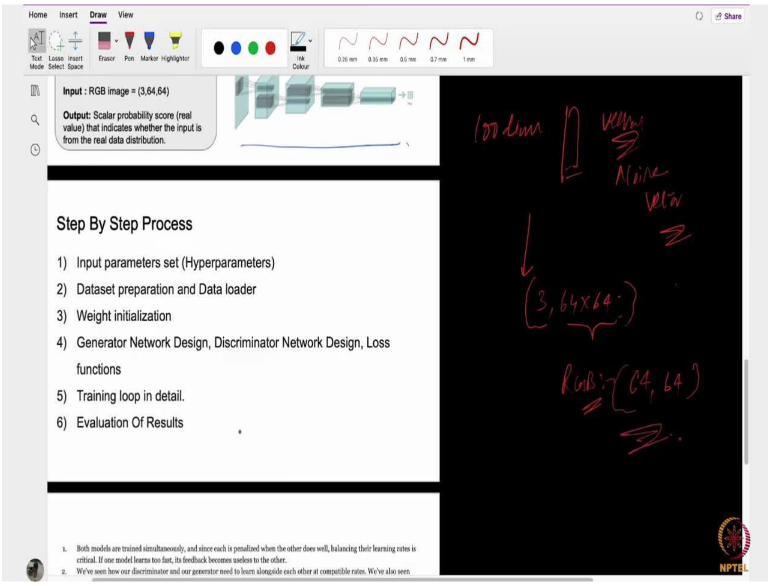
(Refer Slide Time: 09:19)



Now, let us speak about generator for some time. Generator contains only transpose CNNS transpose CNNS and the second thing is BatchNormalization is present and all ReLU activation functions for all the layer except for the output which uses tan(h) activation function.

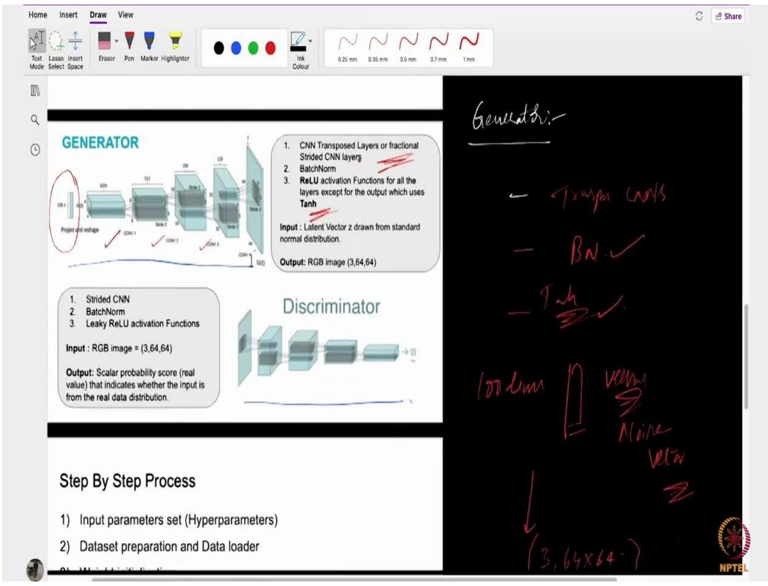
Now what is the input to the generator and what is the output of the generator let us see. For the generator in this case they have used 100 dimensional vector, 100 dimensional vector some random noise vector will be fed into a generator.

(Refer Slide Time: 10:16)



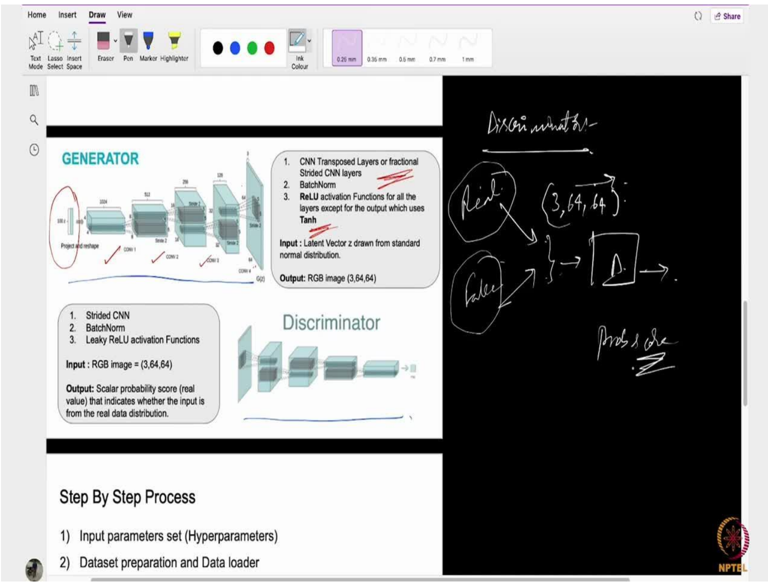
So that output generated by this generator will be a 3 channel, 64×64 image that is we can say RGB 64 height and 64 width.

(Refer Slide Time: 10:41)



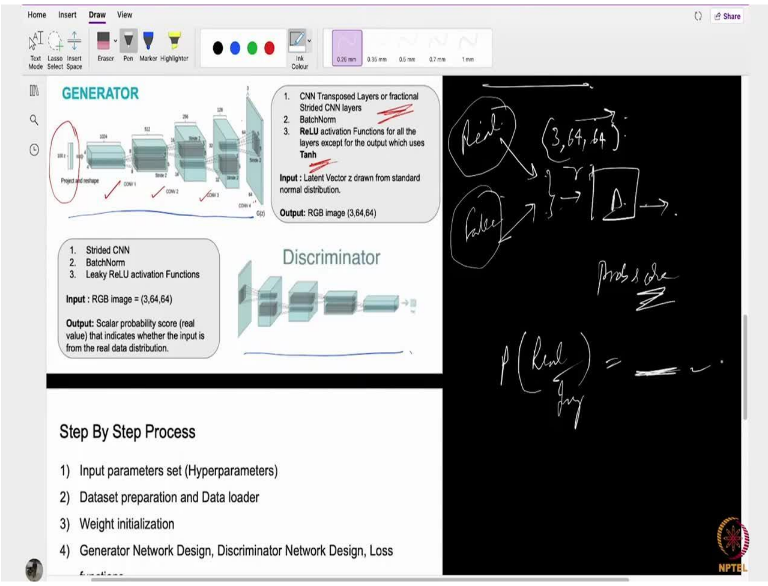
Now observe here in generator we have fed under dimensional noise vector and all the convolutional transpose convolution transpose convolution transpose layers are used at the end we will get 3 channel having height 64 similarly width also 64 image has been generated.

(Refer Slide Time: 11:04)



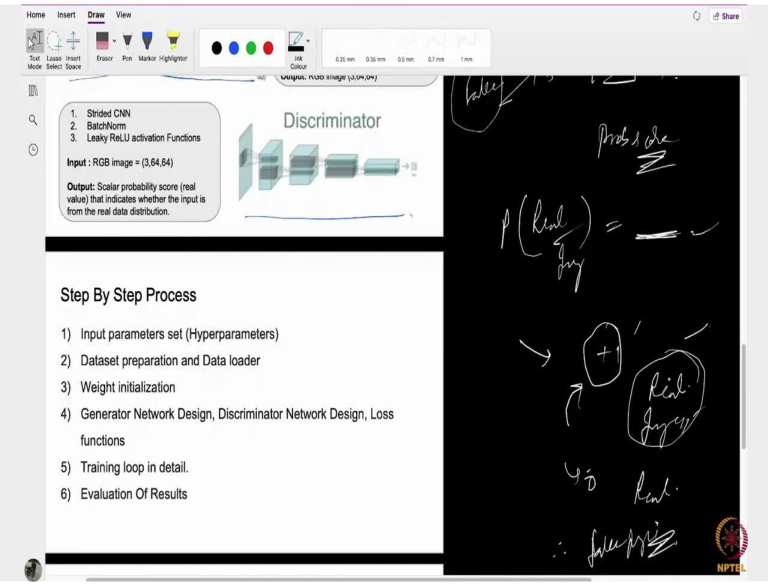
Now, let us speak up discriminator, now discriminator will take input inputs as either real images or fake images that means, the input to the discriminator will be a 3 channel 64 plus 64 image, that means, RGB any of these images either real or fake images, fake image generated by the generator or real images that are brought from original data set will be fed it to the discriminator and now, the discriminator will give the probability score, probability score.

(Refer Slide Time: 11:52)



The score indicates whether this particular any of these RGB belongs to real data that is given image being real that is what this score indicates.

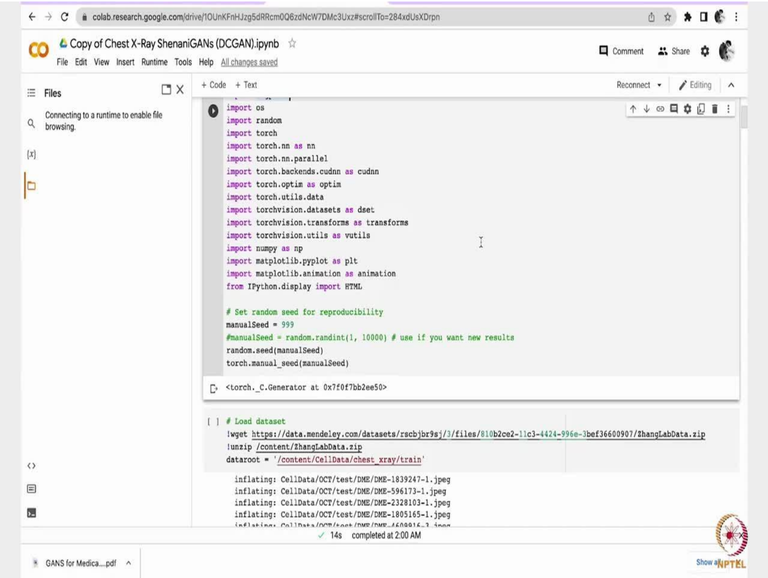
(Refer Slide Time: 12:06)



Suppose, the output score for any given image is very near to 1 what it means? It is a real image. There are more chances that the image that has been fed is can be considered as a real image.

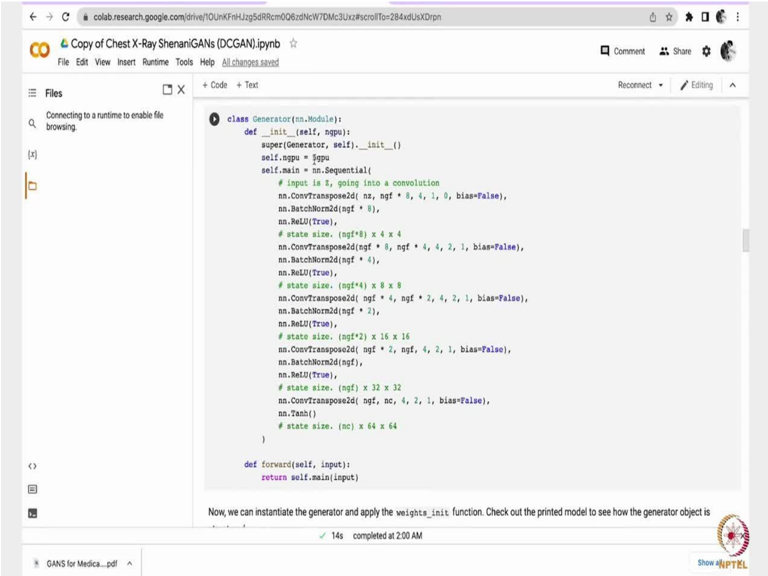
Suppose this value is very near to 0 that means, there are very less chances for the given image to be real what we can conclude now, maybe that image is fake images, fake image. This is how discriminator and generator will function.

(Refer Slide Time: 12:45)



Now, let me show you in what way generator and discriminator are designed in the network using these 5 guidelines. Suppose this code has been designed for this chest X ray image generation using the DCGAN algorithm. Here we have two parts like generator network similarly discriminator network let me go through.

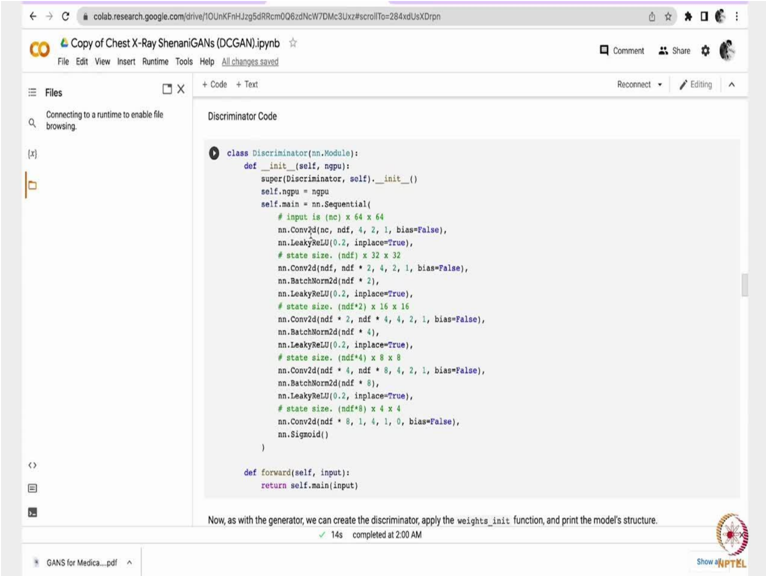
(Refer Slide Time: 13:16)



See here there here they are indicating generator network. In the generator network, if you observe clearly, they have used only convolutional transpose layers see here. Batch Normalization there, ReLU activation function, convolution 2D, transpose 2D, BatchNormalization ReLU convolution transpose 2D, BatchNormalization ReLU, if you observe clearly at every stage, they have ReLU activation functions only except at the lost layer at the lost layer, they have used Tan h function which I have discussed earlier.

So in generator almost all the convolutional 2D, transpose 2D networks are used at the same time BatchNormalization is used and ReLU activation functions are used at every layer except at the output layer where they have used tan(h) activation function. Here if you observe clearly I will get back to what is this noise vector and all the stuff later on just the basic idea this is or in another way we can see let me tell you.

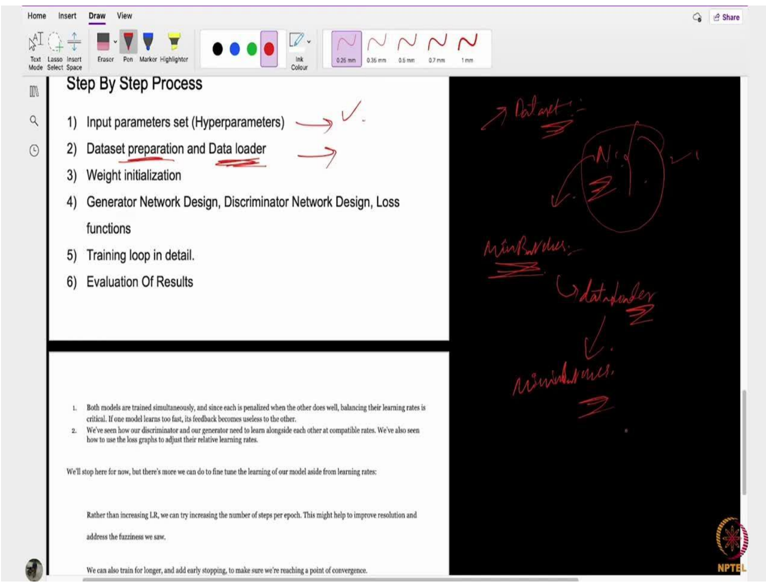
(Refer Slide Time: 14:26)



Now let us get back to discriminator code. Here if you observe they have used strided convolution only just normal CONV 2D layers and leaky ReLU, leaky ReLU, leaky ReLU, leaky ReLU and CONV 2D. Everywhere they have used leaky ReLU activation functions. But at the end, they have use sigmoid that is for outputting the probability score that is either near to 0 or near to 1.

That is will having sigma, since sigmoid activation functions, output values that lie between 0 to 1. Just as a basic idea discriminator has been designed like this. Similarly, generator has been designed like this, according to the 5 rules framed by the others in the DCGAN paper.

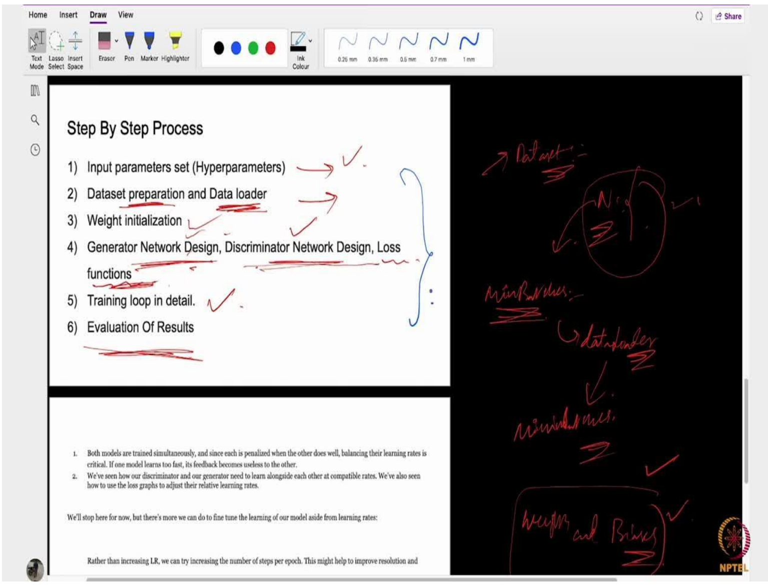
(Refer Slide Time: 15:15)

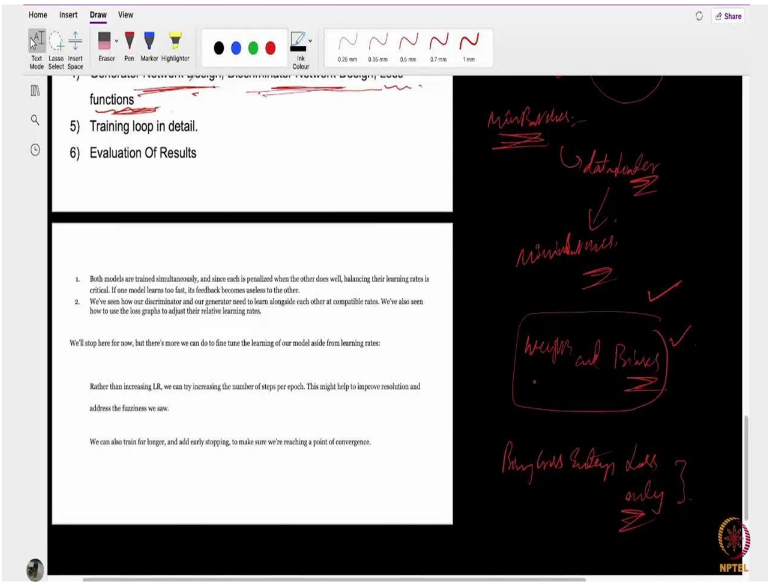


Now, let us see what exactly we are going to discuss later on. First of all, we will set some hyper parameter values next thing is creating the data set preparation and data loader preparation. In deep learning, we will first prepare the data set after downloading the data set whatsoever the necessary transformations are to be done please do it and later on, we will suppose the data set contains some N images.

Total N images are there in the data set you should have directly feeding these N total of N images into the model directly we prefer mini batches we prefer feeding mini batches rather than feeding the whole data set we will prefer feeding mini batches dataset. So, that is where this data loader part will come. That means, it will help you to load data in mini batches, mini batches from the entire data set.

(Refer Slide Time: 16:31)





After that, we will prepare the model after preparing this model we should initialize the weights and biases, weights and biases. Further in simple way, we call it as weight initialization step we will follow some criteria, to initialize these weights end of the day even weights are also just numbers only, but we will have we will put some restrictions like we will keep all these weights or these numbers we will take it from either some normal distribution or some particular probability distribution that is up to us.

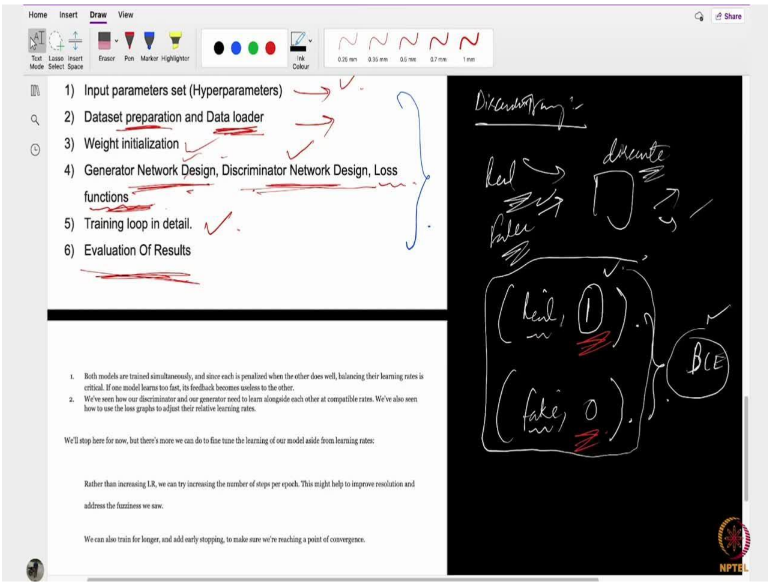
But in the paper they have used standard normal distribution only what the code whichever I show you also they have used the normal distribution only. Weight initialization and the next step is generator network design, discriminator network design, loss functions as I said earlier, they are just prepare a weight initialization function later on after designing these models after designing generator network and discriminator network, those initialization weight initialization function will be called on here so that the weights and biases of these two networks will be initialized using this function.

Later on they have designed the loss functions also end of the day here we are using the classification only be the particular image being real or fake that means, we just here using binary cross entropy loss only, binary cross entropy loss only. Later on when we go and see the code, I will explain you there I will show you what sort of loss function they have used binary cross entropy.

Next thing is training loop since the network is designed data loader, data everything is designed and the loss functions are also mentioned. Now, our target is to train start the training that is where this training loop in detail will be explained later on we will evaluation of the result.

At the end of the training what we will do is using the particular model we will generate some images from the GAN sorry generator and I will compare those images with the original real data images. This is what the entire process we are going to see while implementing the while going through the code.

(Refer Slide Time: 19:03)

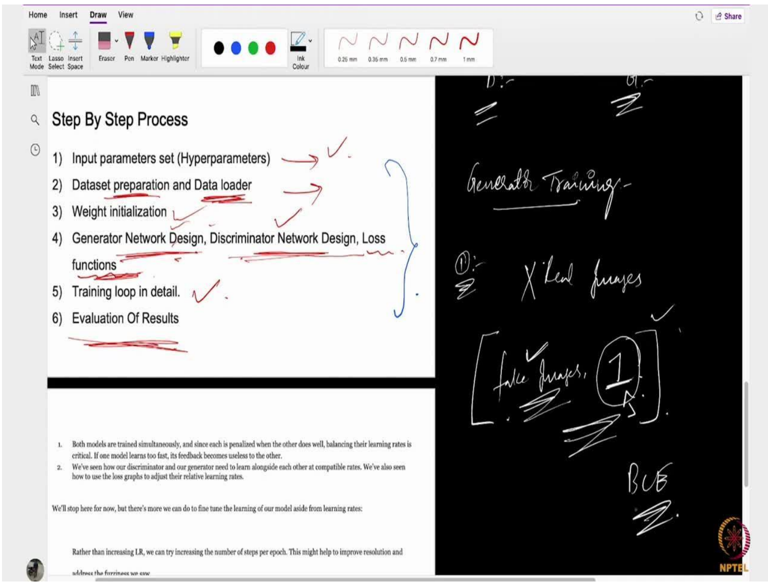


Before that let us focus on discriminator training, discriminator training where here real images and fake images both will be fed into the discriminator. Now its task is to classify. So, what it will do is? Here suppose the given input is real image, real image the output the target value will be kept as 1 whereas if you provide a fake image, I mean the image generated by the generator at the time the target value is 0. This is how the loss function is designed there. And end of the day we will calculate that corresponding binary cross entropy loss only.

But given these targets or maintaining this target is very important here. While training the discriminator we will feed both real images and fake images real images from the training data, fake images from the images generated by the generator.

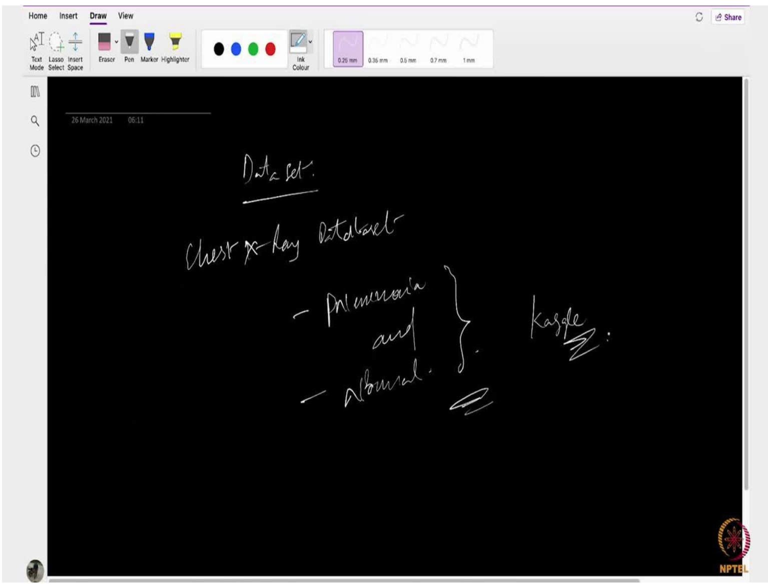
So, how it will so, how it will train leads? The lost will be suppose real input is given as input at the time keep the target values as 1 whereas, if you feed the fake images at the time keep them target value as 0 this is how visual prepare.

(Refer Slide Time: 20:40)



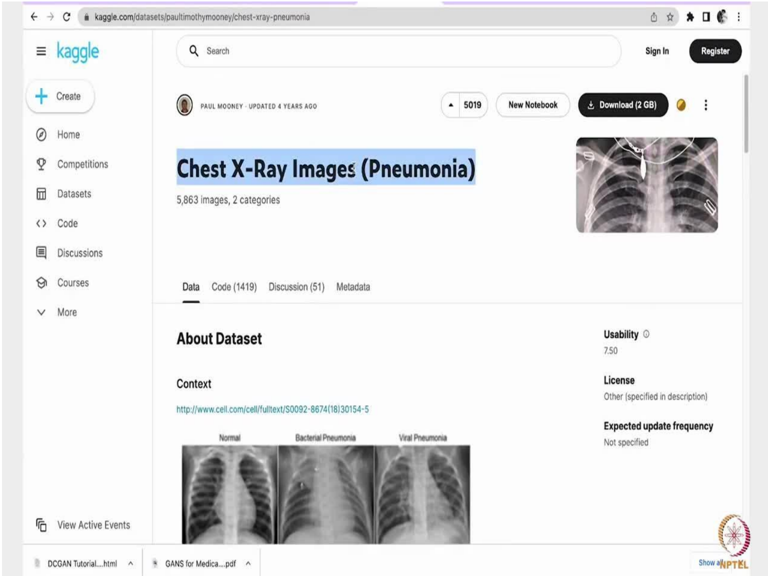
Now, coming back to generator training while training the generator, we will not consider real images, we will not consider real images. So, we only have fake images only, fake images. Here suppose, this is fake image, but here we will give its corresponding target value as 1 that means, we are while training the generator we are fooling the discriminator in such a way that whatsoever the images generated by the generator will their targets are kept as 1 that means, we are fooling the discriminator. This is how the target values are chosen and later on again they will use binary cross entropy loss only. Please keep these points in mind.

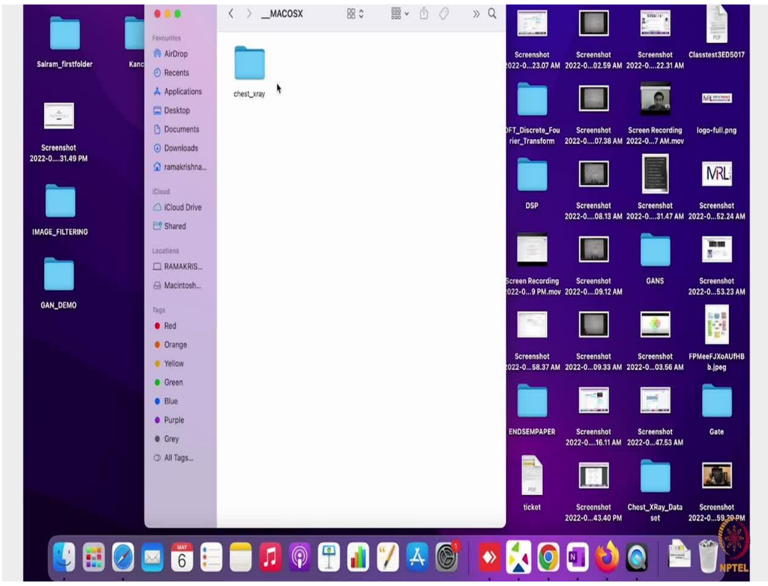
(Refer Slide Time: 21:41)

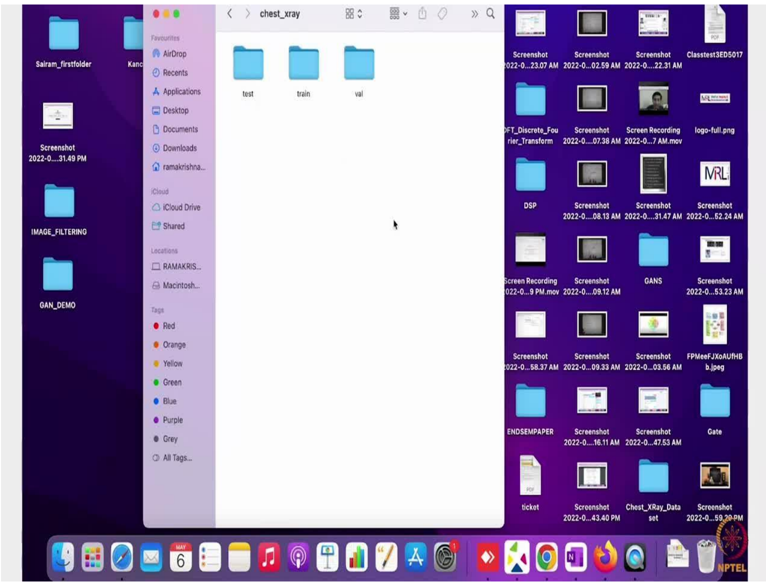


While going through the code, I will tell you I will show you now, let us discuss about the dataset that we are going to use to implement this algorithm data set that is chest X ray data set, chest X ray data set it contains pneumonia and normal of 2 classes of images out there and the data has been downloaded from Kaggle, data has been downloaded from Kaggle .

(Refer Slide Time: 22:19)

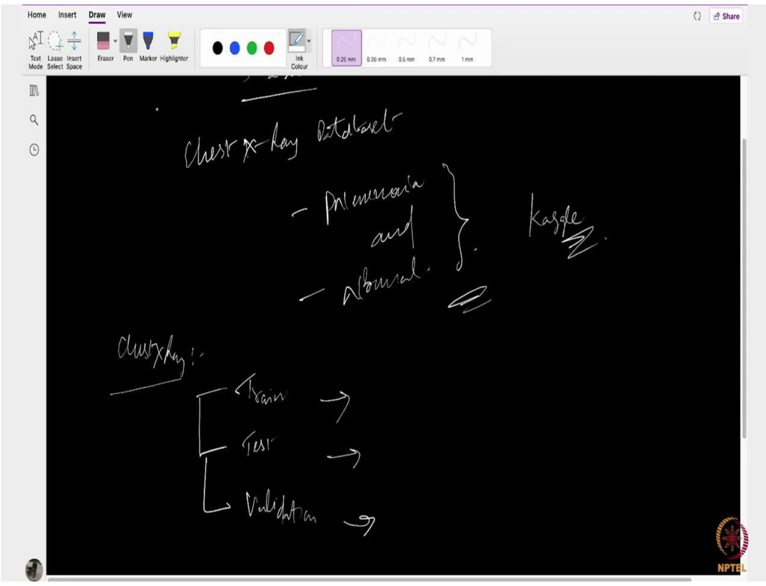


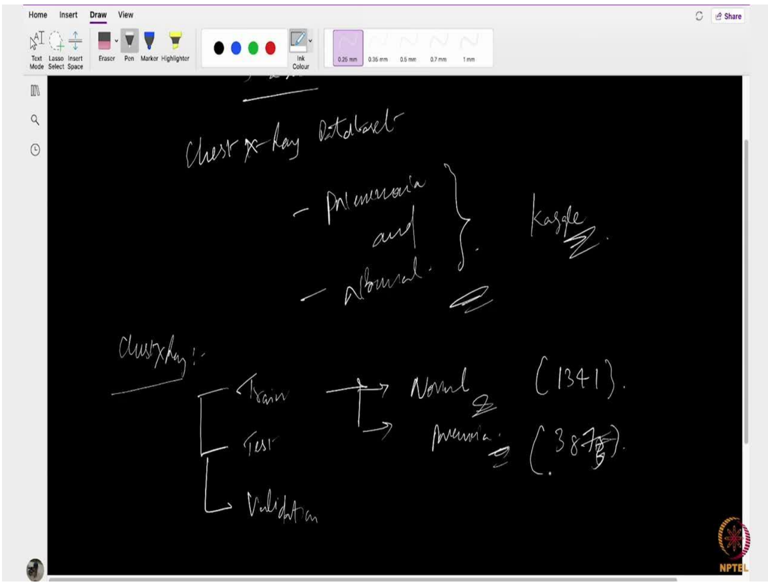


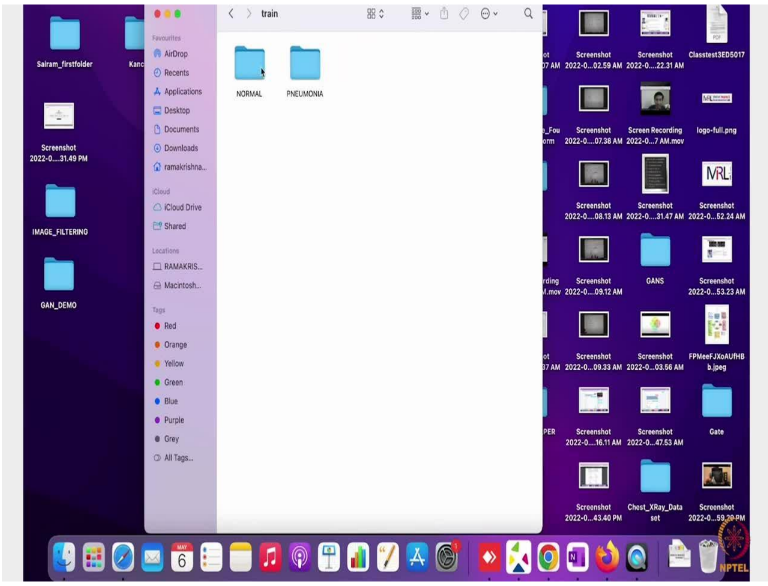


Now let me show you that Kaggle chest X ray images pneumonia just type this one in Kaggle you will get to know about this data there you can see that download 2GB of data, just click here that data gets downloaded to your local computer how it will be? I will show you once you download to your local computer see here, once you download the data from Kaggle in your particular directory wherever you have downloaded this file will be shown this folder that chest X ray and it contains 3 folders.

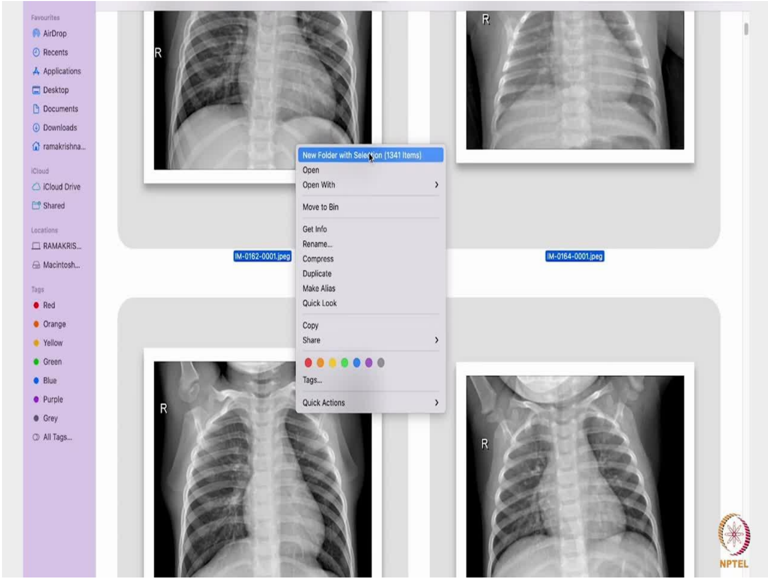
(Refer Slide Time: 23:00)

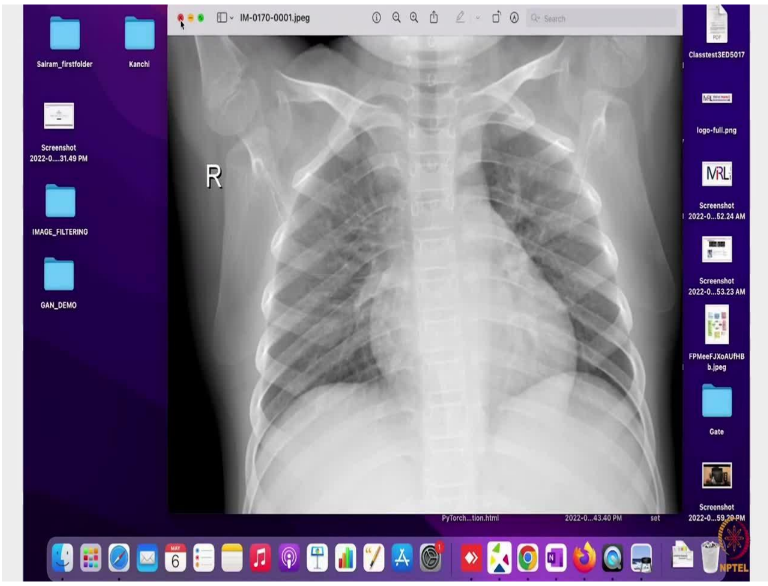


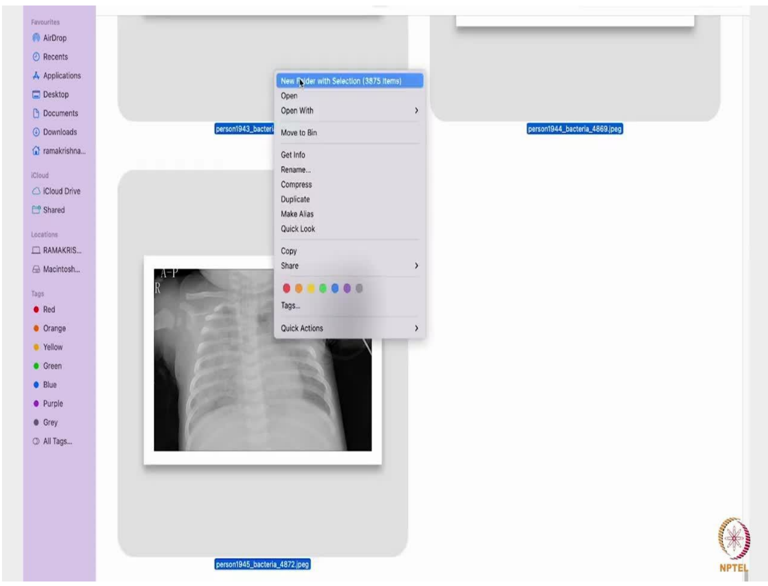

Home
Insert
Draw
View
Share
Chest X-Ray Dataset
- Pneumonia
and
- Abnormal
Kaggle
Chest X-ray:
Train
Test
Validation
Normal (1341)
Pneumonia (3875)


train
NORMAL
PNEUMONIA
AirDrop
Recents
Applications
Desktop
Documents
Downloads
iCloud Drive
Shared
Sairam_firstfolder
IMAGE_FILTERING
GAN_DEMO
GANS
Gate
Chest_XRay_Data set


IM-0115-0001.jpeg
IM-0117-0001.jpeg


Favourites
AirDrop
Recents
Applications
Desktop
Documents
Downloads
ramakrishna...
iCloud
iCloud Drive
Shared
Locations
RAMAKRIS...
Macintosh...
Tags
Red
Orange
Yellow
Green
Blue
Purple
Grey
All Tags...
New Folder with Selection (1341 Items)
Open
Open With
Move to Bin
Get Info
Rename...
Compress
Duplicate
Make Alias
Quick Look
Copy
Share
Tags...
Quick Actions
IM-0162-0001.jpeg
IM-0164-0001.jpeg
R
NPTEL


IM-0170-0001.jpeg
Search
R
Sairam_firstfolder
Kanchi
Screenshot 2022-0....31.49 PM
IMAGE_FILTERING
GAN_DEMO
Classtest3ED5017
logo-full.png
Screenshot 2022-0...52.24 AM
Screenshot 2022-0...53.23 AM
FPMeeFJXoAUfhB b.jpeg
Gate
Screenshot 2022-0...59.39 PM
PyTorch...tion.html
NPTEL

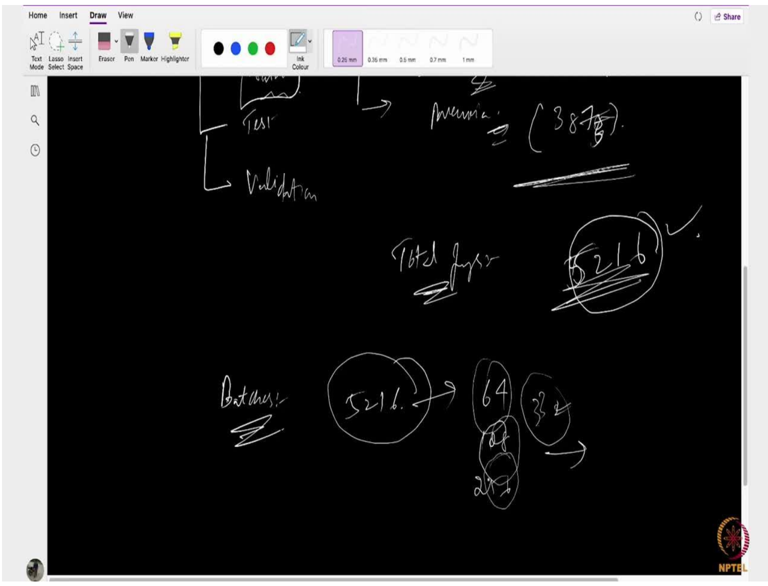

Favourites
AirDrop
Recents
Applications
Desktop
Documents
Downloads
ramakrishna...
iCloud
iCloud Drive
Shared
Locations
RAMAKRIS...
Macintosh...
Tags
Red
Orange
Yellow
Green
Blue
Purple
Grey
All Tags...
New Folder with Selection (3875 Items)
Open
Open With
Move to Bin
Get Info
Rename...
Compress
Duplicate
Make Alias
Quick Look
Copy
Share
Tags...
Quick Actions
person1943_bacteri
person1944_bacteria_4869.jpeg
person1945_bacteria_4872.jpeg
NPTEL

Train, test and validation, let us see what is there in each and every folder as a subdirectory? First we have chest X ray, chest X ray is the main directory and afterwards we have 3 folders now again subdirectory will be there. First one let us see, inside the train directory we have again normal and pneumonia, normal and pneumonia particular class of chest X rays.

Let us see how many files are there in training data even while implementing this algorithm also we mainly focus on this training data itself. Here we are not focusing on test data and validation. Let us see how many files are there instead the training folder, you can see normal and pneumonia

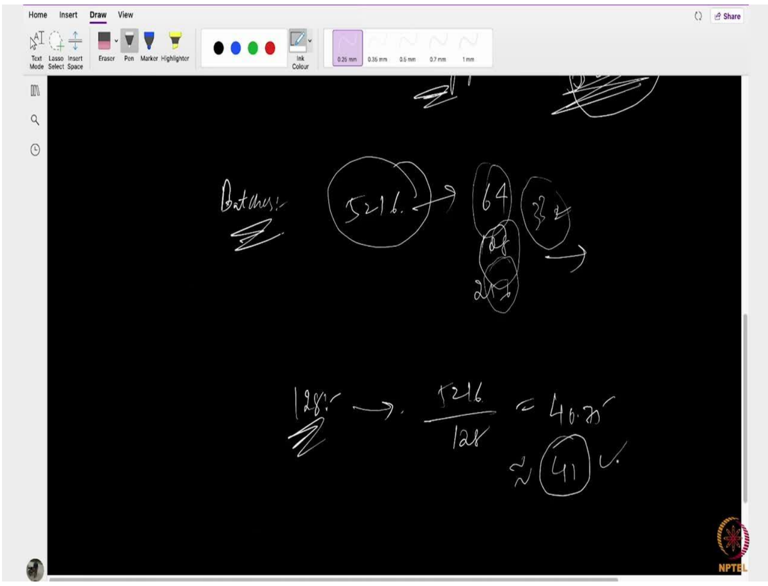
Let me open normal you can see, this is for the images we look like. And all these images are in dot jpeg format, all these images are in JPEG format. See here. Let me count how many images are there total 1341 items in normal folder. In normal folder 1341 chest X ray images similarly let me open one image and let me let us this is for the images look like. You can see clearly. Now in back the data looks like. Let us get back to pneumonia folder and set this pneumonia folder just count the total number of images 3875 the sum total of 3875.

(Refer Slide Time: 25:35)



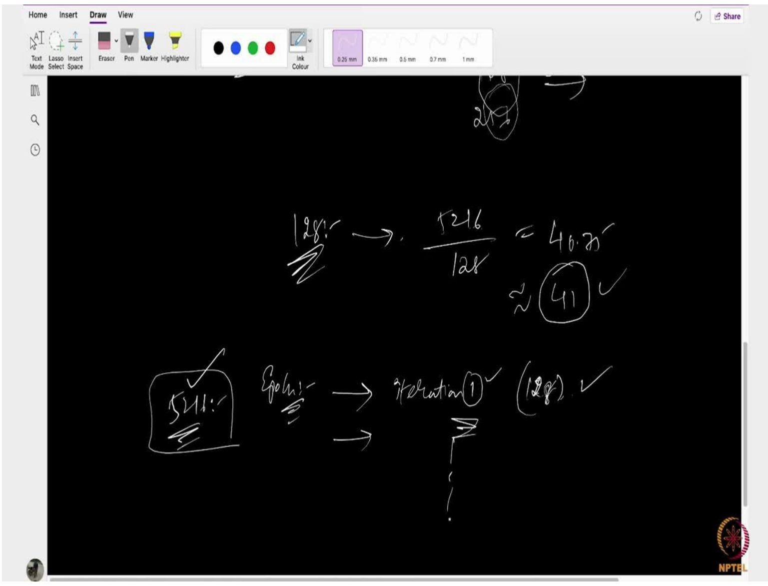
So, total images in the entire training data, training data folder is (1341+3875) that means sum of these two numbers 5216. Just remember this one, once we go and use this data, we need this number because usually for deep learning algorithms, we will give the feed the data in terms of batches, out of this 5216 data, we will keep batch size say some 64, 128, 256 or 32 likewise, we will give several batch sizes.

(Refer Slide Time: 26:24)



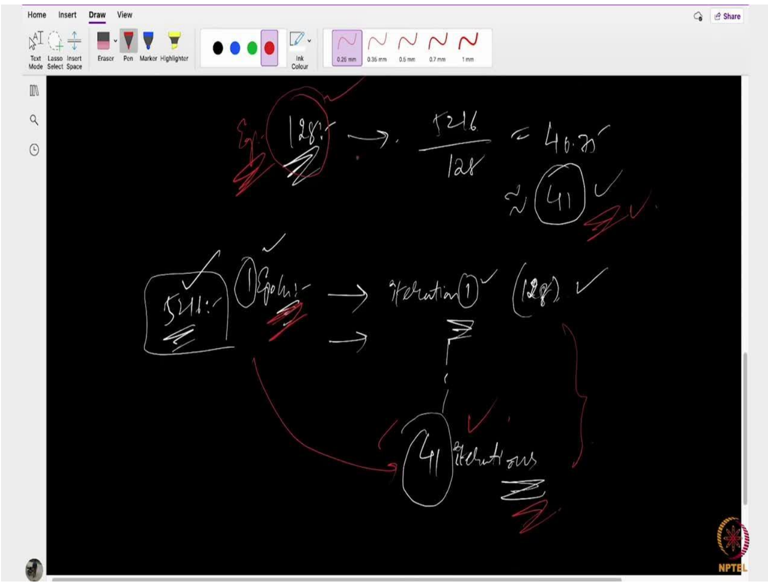
Suppose I take batch size as 128. Let us see, in a single epoch, how many iterations will be there. So, 5216 total images whole divided by 128. So, the answer will be 40.75, so, rounded up to 41.

(Refer Slide Time: 26:54)



That means, total images are 5216, that means one single epoch is, one single epoch means we have to go through entire all the batches of images. That means, on a single batch size, we will say one iteration, one iteration is nothing but one single batch size that means 128 out of these entire images 128 images will be taken and training will be done and we call that phases as one iteration.

(Refer Slide Time: 27:26)

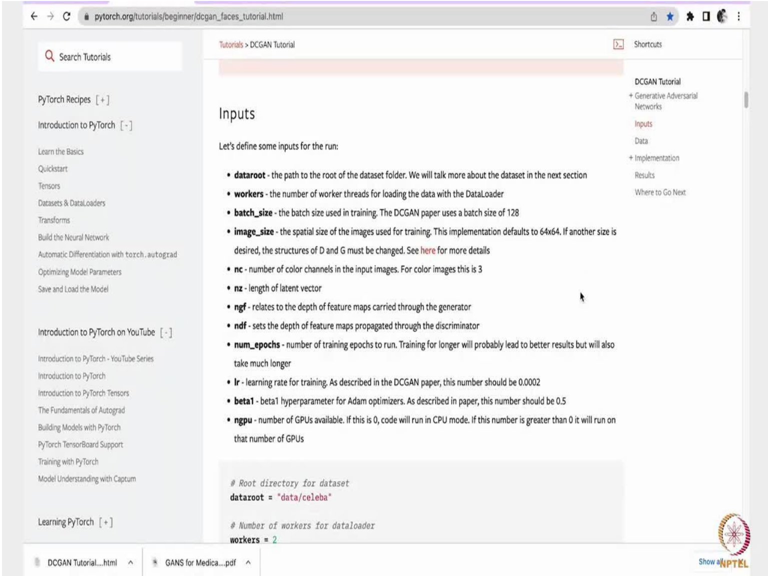


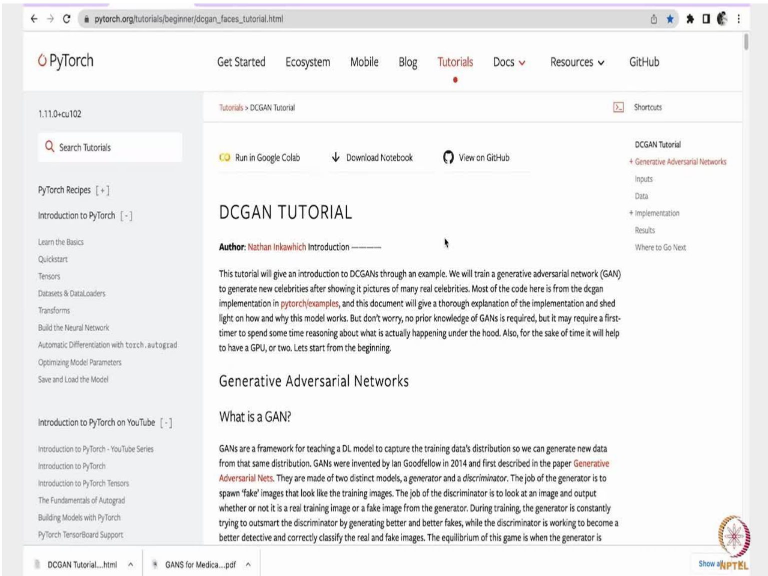
Similar likewise we have total of 41 iterations. Once you take the training data, sorry, once you take the batch size as 128. Now, we can say that in a single epoch, in one epoch, we have 42 iterations that is it. This is what the discussion of enter dataset summarize just X ray data has been downloaded from Kaggle. And we have mainly 2 sets of diseases pneumonia, and I am sorry, 2 classes pneumonia normal.

Now, coming back to normal images, there are 1341 images and it comes to pneumonia, we have around 3875 so, in total 5216 number of images. So, feeding the data to depending model we just divide the data into several batches. So, for suppose, as an example, I have explained it here with respect to 128 as batch size. If you keep 128 batch size so, 5216 whole divided by 128 that means 41 batches in a single entire training data we can make 41 batches or in other way the deep learning training will be happen on each and every batch that means we have in total 41 iterations to go through the entire data set, entire data set or in other way we can say that in a single epoch we are going to have 41 iterations.

Where I explained this stuff means, explaining the while implementing this algorithm I have used batch size as around 128. This calculation will be helpful for you. This is the end of this data set discussion.

(Refer Slide Time: 29:10)





DCGAN Tutorial — PyTorch Tutorials 1.11.0+cu102 documentation

Friday, 6 May 2022 9:02 AM

DCGAN Tutorial —...

Table of Contents

Colab Notebook GitHub

DCGAN TUTORIAL

Author: Nathan Inkawhich Introduction ———

This tutorial will give an introduction to DCGANs through an example. We will train a generative adversarial network (GAN) to generate new celebrities after showing it pictures of many real celebrities. Most of the code here is from the dcgan implementation in pytorch/examples, and this document will give a thorough explanation of the implementation and shed light on how and why this model works. But don't worry, no prior knowledge of GANs is required, but it may require a first-timer to spend some time reasoning about what is actually happening under

So, $D(G(z))$ is the probability (scalar) that the output of the generator G is a real image. As described in Goodfellow's paper, D and G play a minimax game in which D tries to maximize the probability it correctly classifies reals and fakes ($logD(x)$), and G tries to minimize the probability that D will predict its outputs are fake ($log(1 - D(G(z)))$). From the paper, the GAN loss function is

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

In theory, the solution to this minimax game is where $p_g = p_{data}$, and the discriminator guesses randomly if the inputs are real or fake. However, the convergence theory of GANs is still being actively researched and in reality models do not always train to this point.

What is a DCGAN?

A DCGAN is a direct extension of the GAN described above, except that it explicitly uses convolutional and convolutional-transpose layers in the discriminator and generator, respectively. It was first described by Radford et. al. in the paper Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks. The discriminator is made up of strided convolution layers, batch norm layers, and LeakyReLU activations. The input is a 3x64x64 input image and the output is a scalar probability that the input is from the real data distribution. The generator is comprised of convolutional-transpose layers, batch norm layers, and ReLU activations. The input is a latent vector, z, that is drawn from a standard normal distribution and the output is a 3x64x64 RGB image. The strided conv-transpose layers allow the latent vector to be transformed into a volume with the same shape as an image. In the paper, the authors also give some tips about how to setup the optimizers, how to calculate the loss functions, and how to initialize the model weights, all of which will be explained in the coming sections.

```
from __future__ import print_function
#%matplotlib inline
import argparse
import os
import random
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.optim as optim
import torch.utils.data
import torchvision.datasets as dset
import torchvision.transforms as transforms
import torchvision.utils as vutils
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML

# Set random seed for reproducibility
```

being actively researched and in reality models do not always train to this point.

What is a DCGAN?

A DCGAN is a direct extension of the GAN described above, except that it explicitly uses convolutional and convolutional-transpose layers in the discriminator and generator, respectively. It was first described by Radford et. al. in the paper Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks. The discriminator is made up of strided convolution layers, batch norm layers, and LeakyReLU activations. The input is a 3x64x64 input image and the output is a scalar probability that the input is from the real data distribution. The generator is comprised of convolutional-transpose layers, batch norm layers, and ReLU activations. The input is a latent vector, z, that is drawn from a standard normal distribution and the output is a 3x64x64 RGB image. The strided conv-transpose layers allow the latent vector to be transformed into a volume with the same shape as an image. In the paper, the authors also give some tips about how to setup the optimizers, how to calculate the loss functions, and how to initialize the model weights, all of which will be explained in the coming sections.

```
from __future__ import print_function
#%matplotlib inline
import argparse
import os
import random
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.optim as optim
import torch.utils.data
import torchvision.datasets as dset
import torchvision.transforms as transforms
import torchvision.utils as vutils
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML

# Set random seed for reproducibility
manualSeed = 999
#manualSeed = random.randint(1, 10000) # use if you want new results
print("Random Seed: ", manualSeed)
random.seed(manualSeed)
torch.manual_seed(manualSeed)
```

Pytorch framework

Numpy

matplotlib

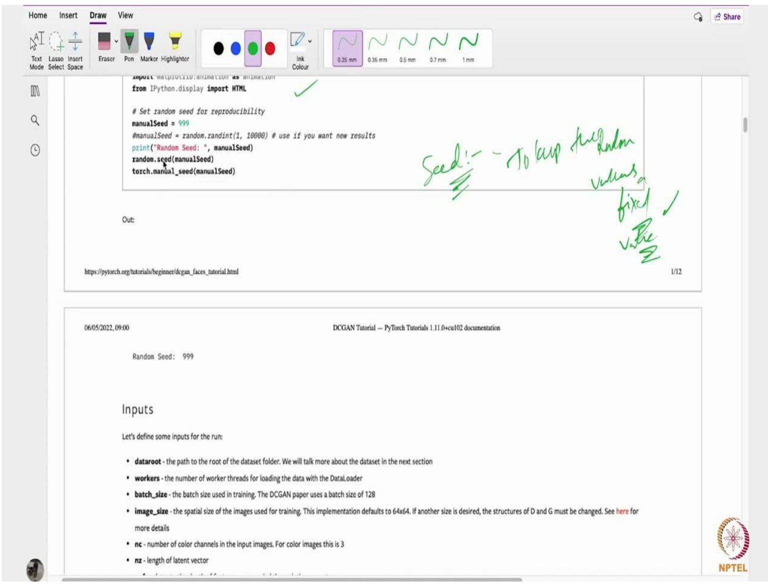
Out:

In this section, we will now see how to implement the code downloaded data set of chest X rays. So here if you go to PyTorch tutorials, here you can see DCGAN tutorial and you can directly go there and go through this, but let me explain you what is happening in each and every block. So that once it is done, we will directly go to our code, whatever I have will build here, see here, this is the entire code first relationship taken here.

See here DCGAN tutorial they just explain what exactly GANS and what we have already discussed in the first code min max algorithm now coming to coming back to DCGAN what they have done is first of all we have to here we are using PyTorch framework first of all PyTorch framework.

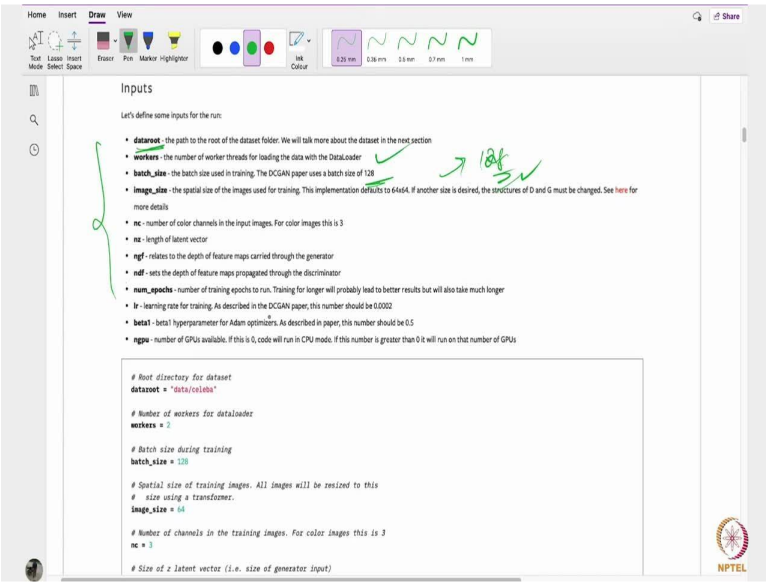
Next thing is they have imported all the module, requires modules, NumPy matplotlib apart from that random torch, here the deep learning library torch dot nn dot nn parallel all these modules are to be imported before going to write code.

(Refer Slide Time: 30:58)



Next thing is they have imported here manual seed and you know usually we use random values especially for initializing the weights or some other places to repeat the same values we always go for seed to keep the random value as a fixed value to keep the random value as fixed value throughout the programme. That means for a particular type of running, for a particular type of running the value should not change that is how we go our manual seeds similar in torch and even random seed.

(Refer Slide Time: 31:38)



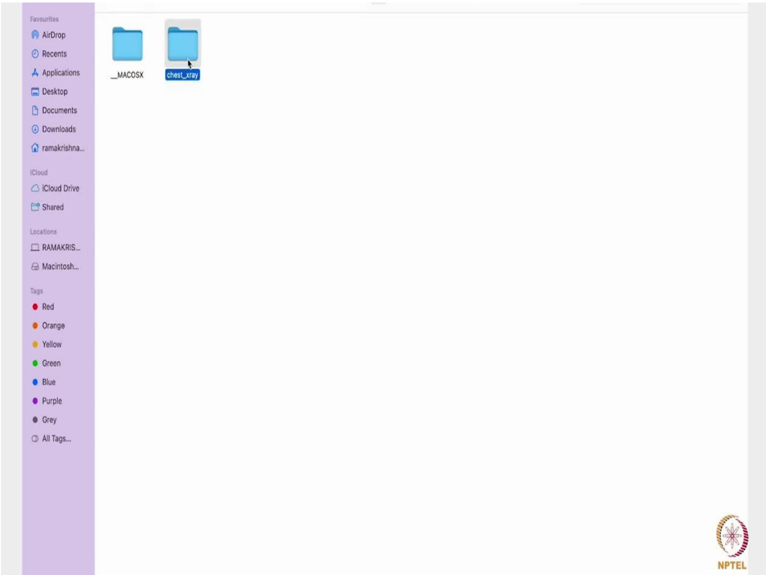
Inputs

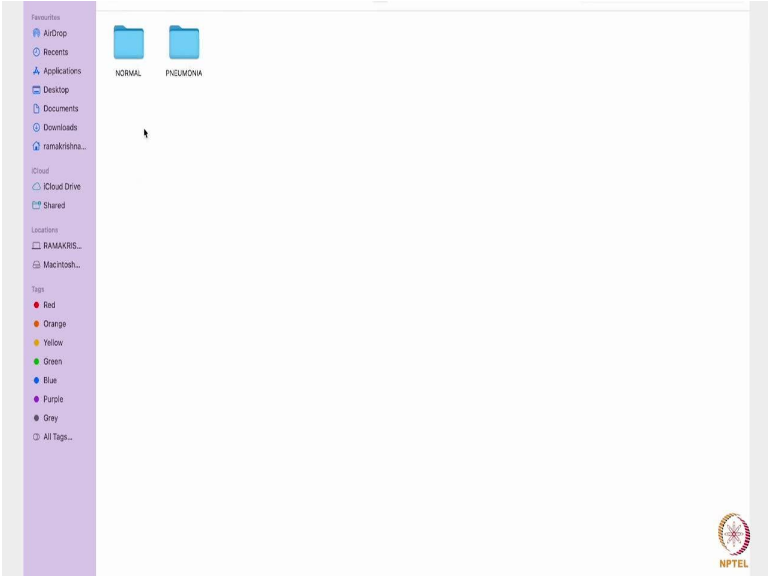
Let's define some inputs for the run:

- **dataroot** - the path to the root of the dataset folder. We will talk more about the dataset in the next section
- **workers** - the number of worker threads for loading the data with the DataLoader
- **batch_size** - the batch size used in training. The DCGAN paper uses a batch size of 128
- **image_size** - the spatial size of the images used for training. This implementation defaults to 64x64. If another size is desired, the structures of D and G must be changed. See here for more details
- **nc** - number of color channels in the input images. For color images this is 3
- **nz** - length of latent vector
- **ngf** - relates to the depth of feature maps carried through the generator
- **ndf** - sets the depth of feature maps propagated through the discriminator
- **num_epochs** - number of training epochs to run. Training for longer will probably lead to better results but will also take much longer
- **lr** - learning rate for training. As described in the DCGAN paper, this number should be 0.0002
- **beta1** - beta1 hyperparameter for Adam optimizers. As described in paper, this number should be 0.5
- **ngpu** - number of GPUs available. If this is 0, code will run in CPU mode. If this number is greater than 0 it will run on that number of GPUs

128

```
# Root directory for dataset
dataroot = "data/celeba"

# Number of workers for dataloader
workers = 2

# Batch size during training
batch_size = 128

# Spatial size of training images. All images will be resized to this
#   size using a transformer.
image_size = 64

# Number of channels in the training images. For color images this is 3
nc = 3

# Size of z latent vector (i.e. size of generator input)
```





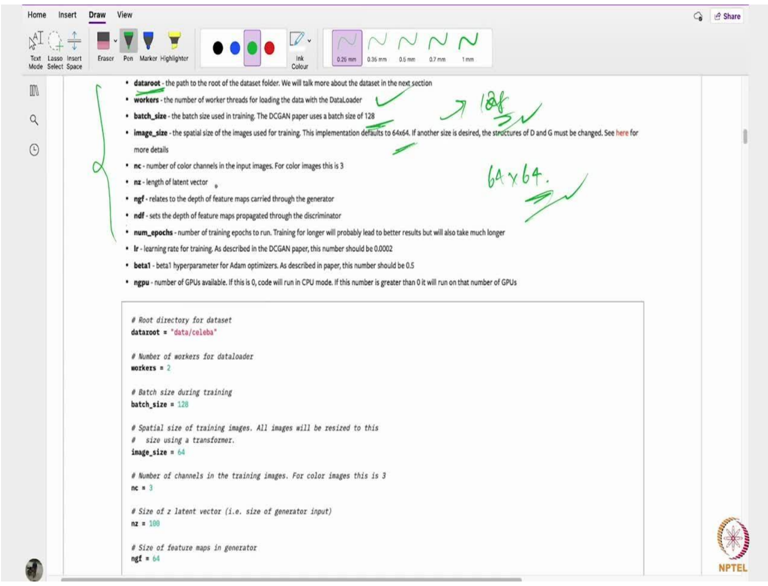
Now coming back to the inputs or hyper parameters whatever we are keeping here. Here you can see the dataroot, dataroot in the sense the root directory suppose here as I have shown you earlier here I am trying to use chest X rays this is the root chest X rays folder is the root folder inside this again I am going to track train data this is what my root folder, inside that root folder we have normal and pneumonia folders.

This is what we even for your may be for this kind of data set we are we have done like I suppose in case of your data set, you may have some root directory and you may have trained you should have chest X rays you might have some other data set inside this you will definitely have this division of test, train and validation.

Inside trained data you will have, according to a problem you will have N number of classes and I want you to keep this kind of directory structure, root directory main root directory, its sub directory and its subdirectories, this is how the files should be arranged folder should be arranged.

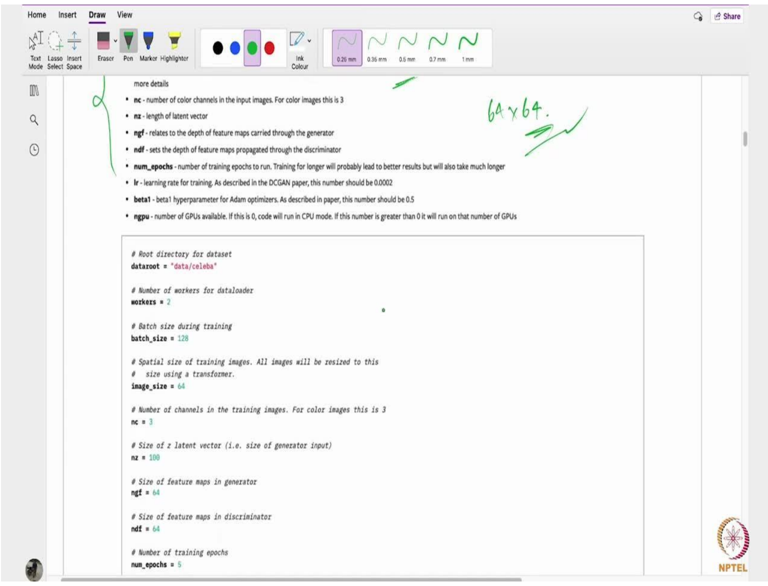
Next thing is workers. Next thing is batch size 128 they have taken 128. Later on you can change this one and you can play around with this one. So, we call this one as a hyper parameter.

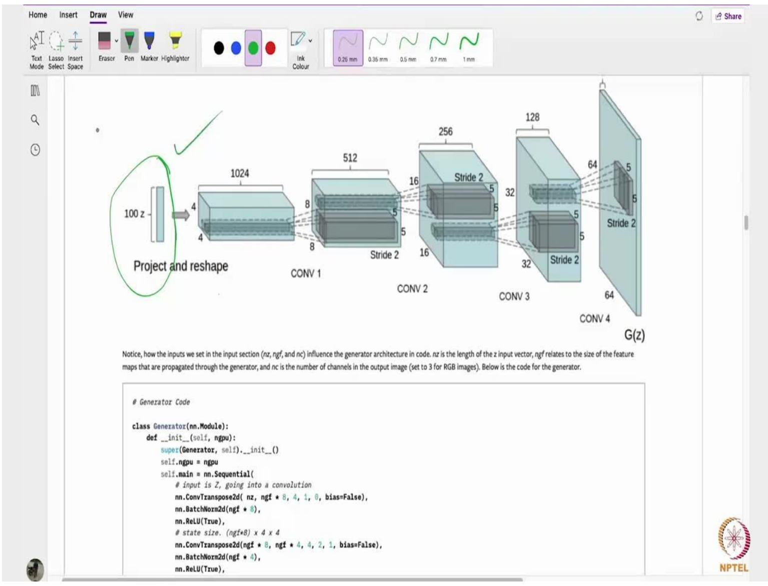
(Refer Slide Time: 33:03)

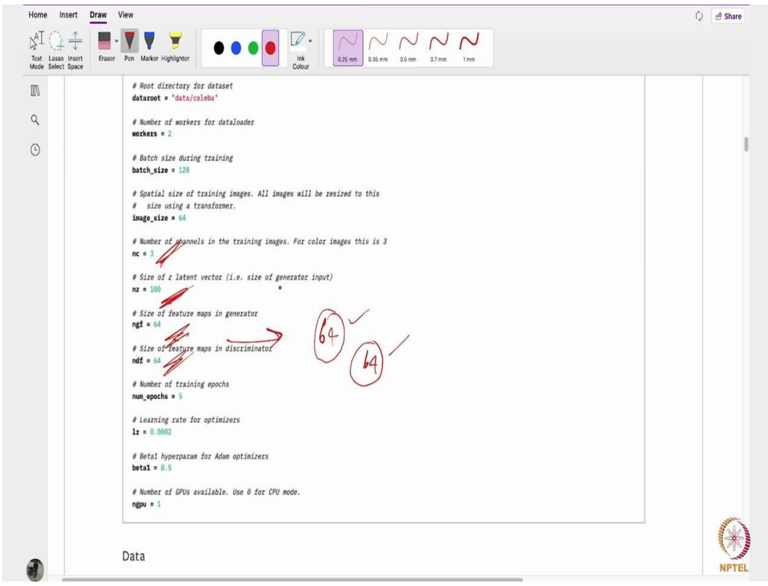


Next thing is image size here for the implementation sake they have used 64×64 I mean even you whatsoever data you are working on for the all the images in this particular data set has to be resized to 64×64 sample. Next number of color channels in the input image for color channel this is 3, number of channels of the particular data set of images you are having.

(Refer Slide Time: 33:33)



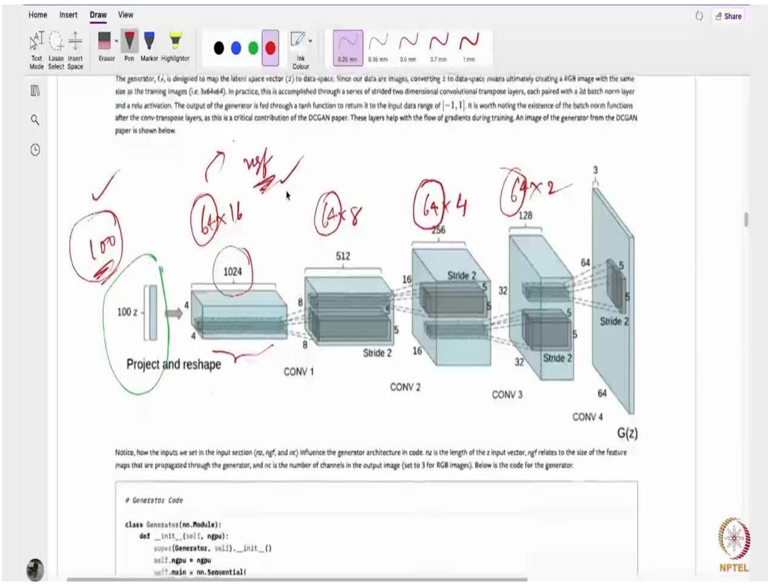




Next thing is the nz that is length of the latent vector it means the random noise vector so, I have already told you let us go to in the case of generator it takes 100 dimensional input vector here this is called they are saying as latent vector. You can here also they had kept it as nz equal to 100 this is the number of channels of the input images.

Next thing is size of feature maps in generator and number of size of feature maps in discriminator. Here you can keep it according to your problem at your hand you can keep these values but here they had kept as 64 and 64 for both discriminator and generator I will explain you in generator section.

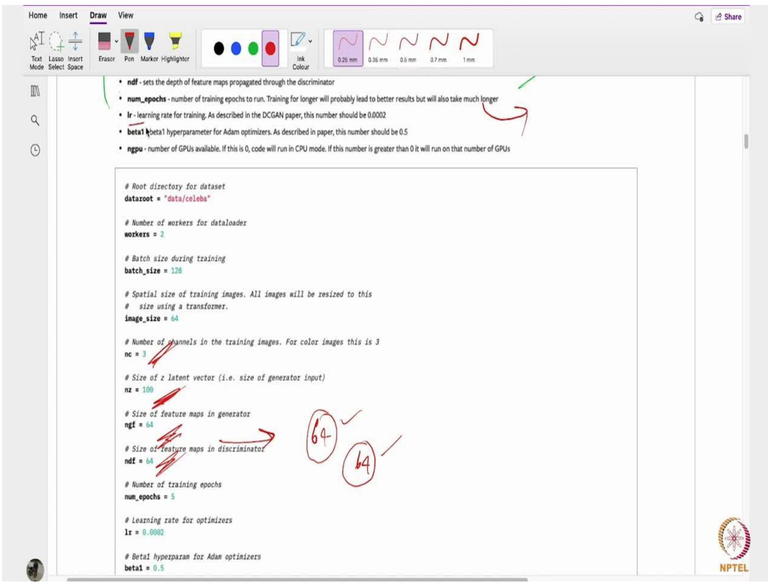
(Refer Slide Time: 34:16)

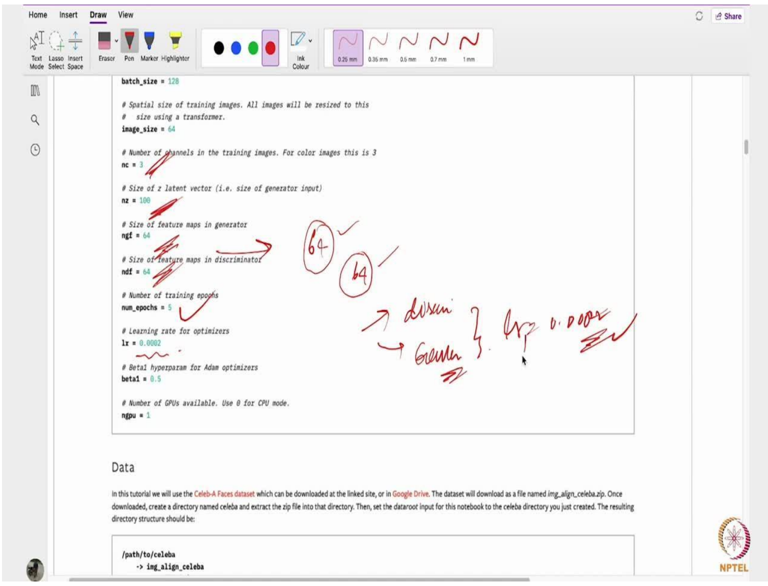


So that you will get some idea let me continue here the random noise input vector is 100 size here if you observe this feature maps values, we can write this one as 64×16. For this one we can write 64×8, this one we can write 64×4 that is right. And again this one can be written as 64×2 that means this value has been fixed.

This is what I am talking about it as ngf value. The same thing will be working out for discriminator session also. Please keep it the point in mind. Maybe this value can be changed according to your according to the problem at your hand, but in this case, we are using this one as 64 only. More or less you can give the same values only.

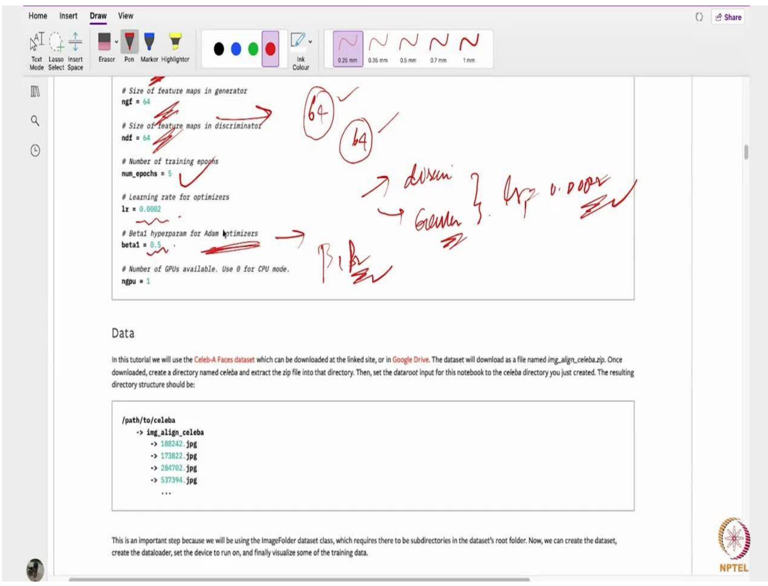
(Refer Slide Time: 35:13)

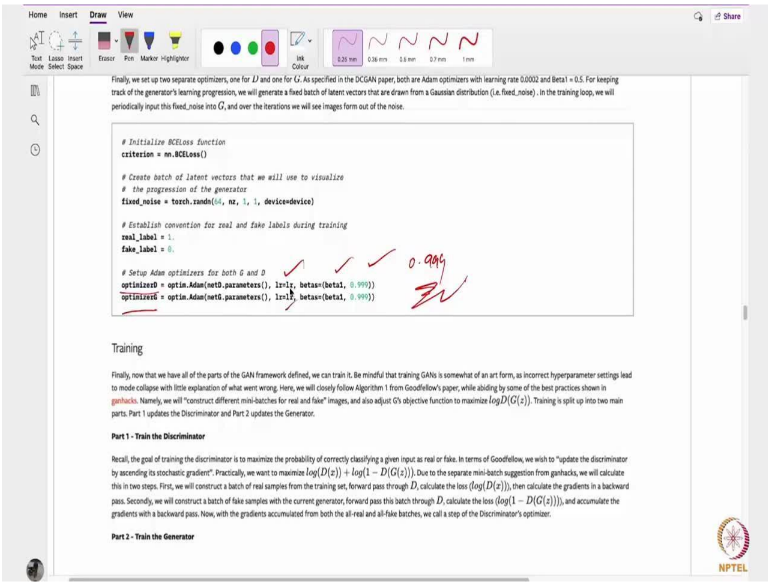




Now coming back to ndf, similarly number of epochs it depends on how many number of epochs you want to go through the entire training process. Similarly learning rate if you observe here number of epochs they have taken as 5 value learning rate has been fixed for both different discriminator we can have 1 learning rate and for generator we can have another learning it, but here they have kept both the values as lr = 0.002, so remember 02. Later on they you can play around with them, so, that you can bring some modifications, modifications in terms of improvement.

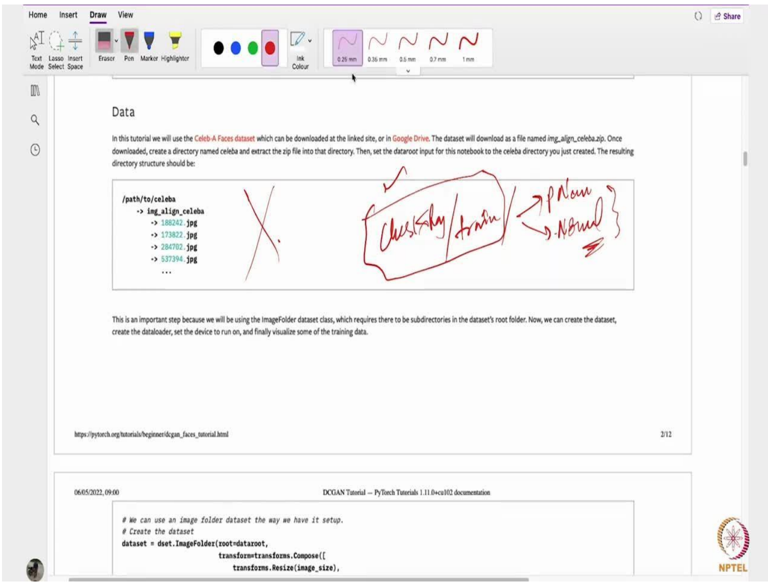
(Refer Slide Time: 35:51)





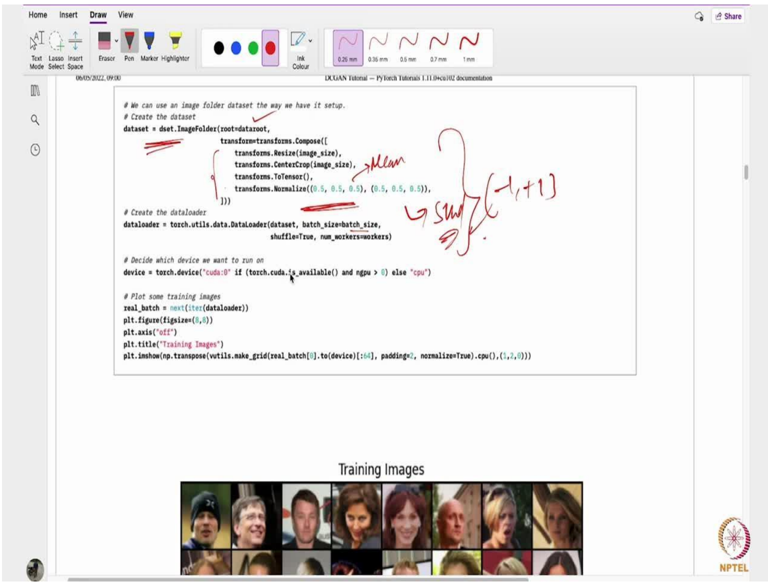
At the end another thing is they have used the optimizer as rm optimizer, optimizer has two parameters β_1 and β_2, β_1 has been fixed to 0.5 and β_2 let us see where and see here the optimizer bm optimizer generator for optimizer generator optimizer and discriminator optimizer they have usually the parameters same learning rate here β_1 is 0.5 and the other β_2 is 0.999 these values are fixed you can change a lr value here or later on when you play around with the hyper parameters you can bring a lot of changes here. These are the discussion of hyper parameters.

(Refer Slide Time: 36:40)



Here they have some other data set in the tutorial session but in our case we are going to use chest X ray data set. In chest X ray data set we have train folder inside the train folder we have 2 sub directories so, our root directory will be this one, here number of classes, here pneumonia and another thing is normal, nothing else. Now coming back to, once you keep these hyper parameters and data here data set organization.

(Refer Slide Time: 37:20)

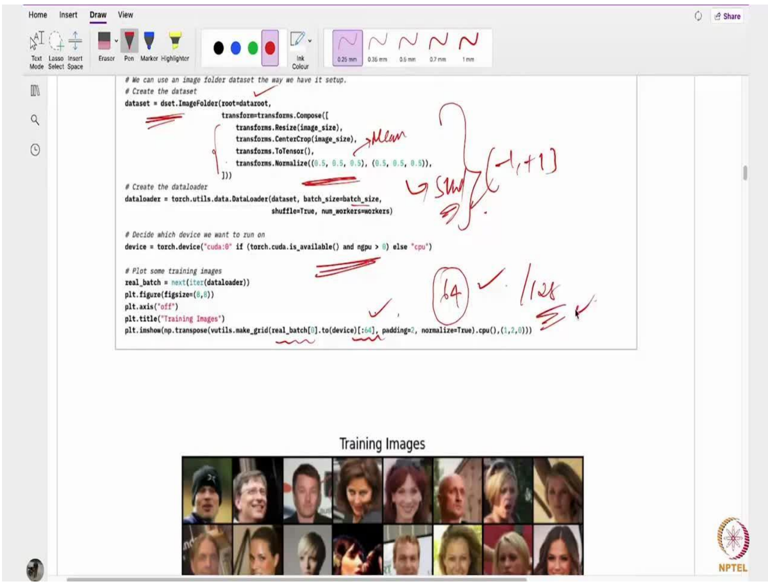


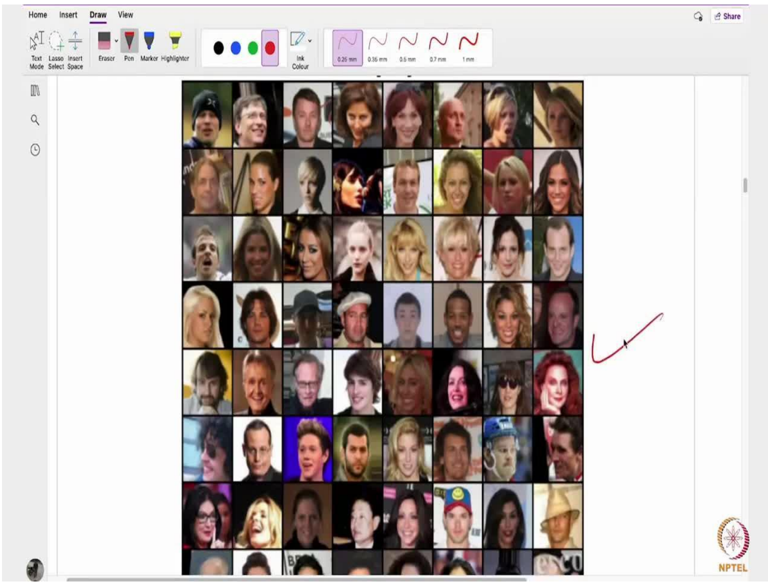
Now coming back to data set and data loader these are the very important concepts here you can see the data root directory will be given these are like transforms I mean you can resize the image you can center crop the image you may have to convert to the usually the entire deep learning will be worked on tensors not a Numpy arrays.

Next thing is we have to normalize the values to minus 1 comma plus 1 these things will first one indicates mean values for each and every channel similarly this indicates standard deviation of each and every channel. Please keep those words in your mind this is more or less like bringing augmentation inside this one.

After creating this data set, we have to go for data loader where we have, we will provide a batch size the entire data set has been given here. So that 128 images, 128 images will be fixed out and will be use it for training. Now please keep this suppose in case you have a local GPU in your at your home or you have local GPU so that you can use this one is command otherwise even in Google Colab also you can use that one.

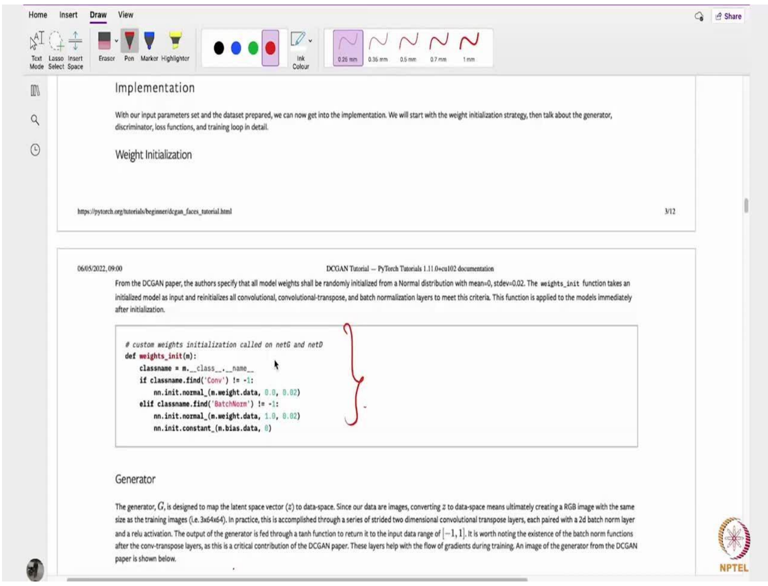
(Refer Slide Time: 38:36)





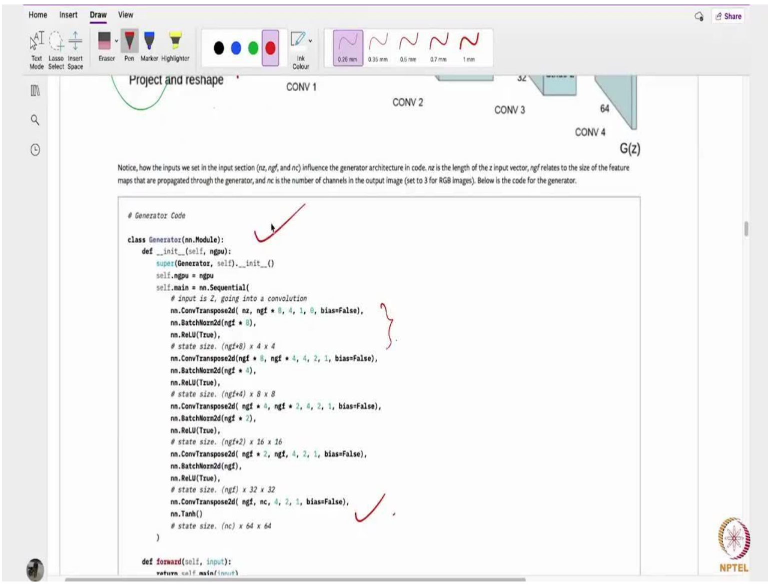
Here they have out of say 128 images as a batch. Let us see 1 2 3 4 5 6 7 8, 1 2 3 4 5 6 7 8, 8 ×8 64. Here they are bringing out some total number of a single here see real batch of 0 dot to the 64. They are attracting 64 images out of 128 from a particular batch and they have shown here using matplotlib.

(Refer Slide Time: 39:10)



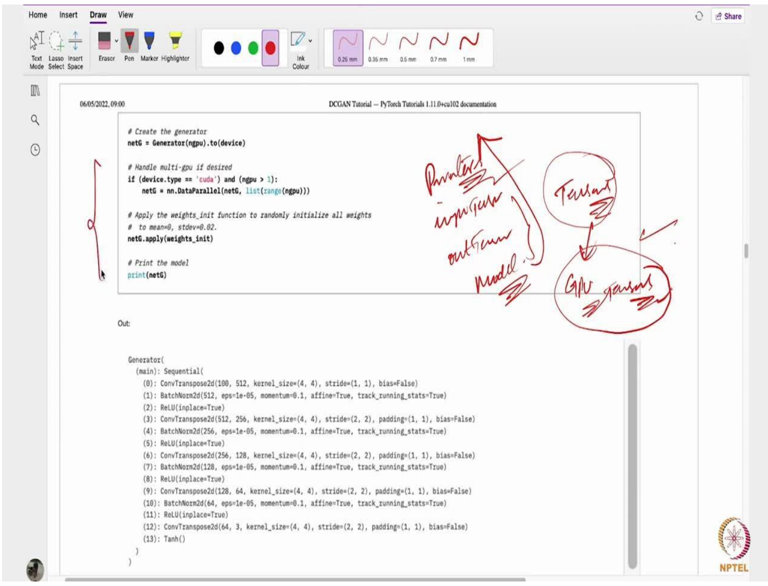
Next thing is weight initialization during the earlier one I have told you we have to initialize the weights.

(Refer Slide Time: 39:19)



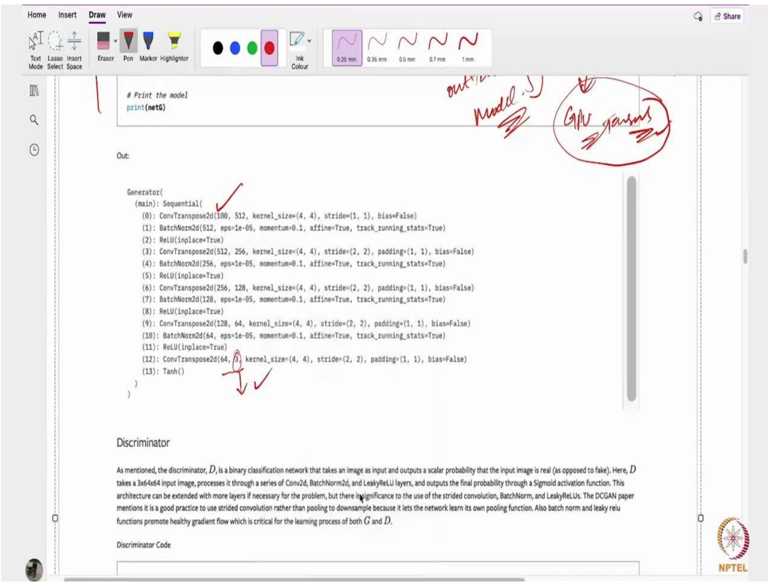
Next thing is generator once we go through here in generator section they use only transpose convolutions and now batch normalization and ReLU activation function at the end only they are used only tan(h) function I have already discussed in the theoretical part. Same thing has been implemented here. Batch normalization is present here everywhere this will follow the same diagram what they have shown here. You will get to know it is not very difficult.

(Refer Slide Time: 39:49)



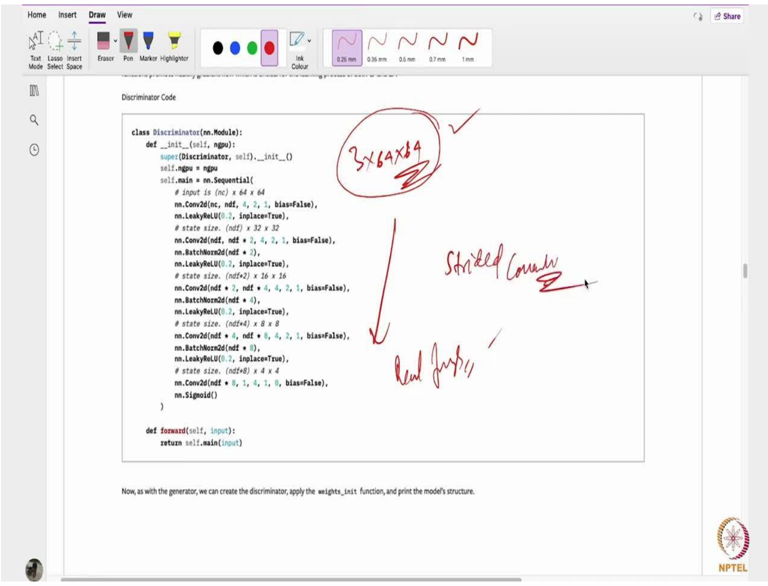
Next thing is implementation of generator block. Here in deep learning all the parameters all the input tensors or the output tensors or even the model means these parameters will come into play. Everything should be in the format of tensors only. Suppose if you are using GPU all these tensor has to be load to GPU tensors This is a fundamental point whatsoever the data you have every all those input images or whatsoever the data irrespective of the kind of data you attached images, images any other everything should be converted into tensors. Suppose if you are using GPU those tensors, tensors should be converted with GPU tensors, this is what they are doing here.

(Refer Slide Time: 40:41)



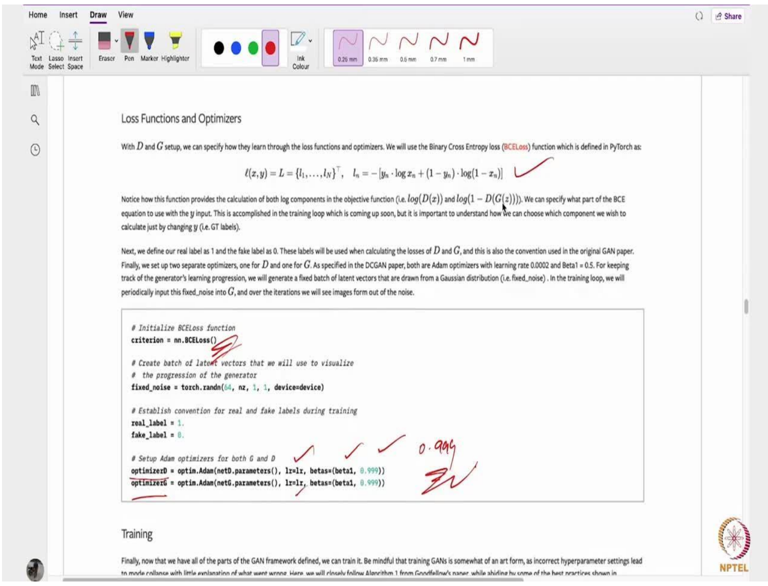
Now, you can see 100 dimensional random noise vector has been kept as an input 512 feature maps outset (512,256) , (256,128) , (128,64) at the end, you will have only 3 feature maps containing output, that is not required here.

(Refer Slide Time: 40:57)



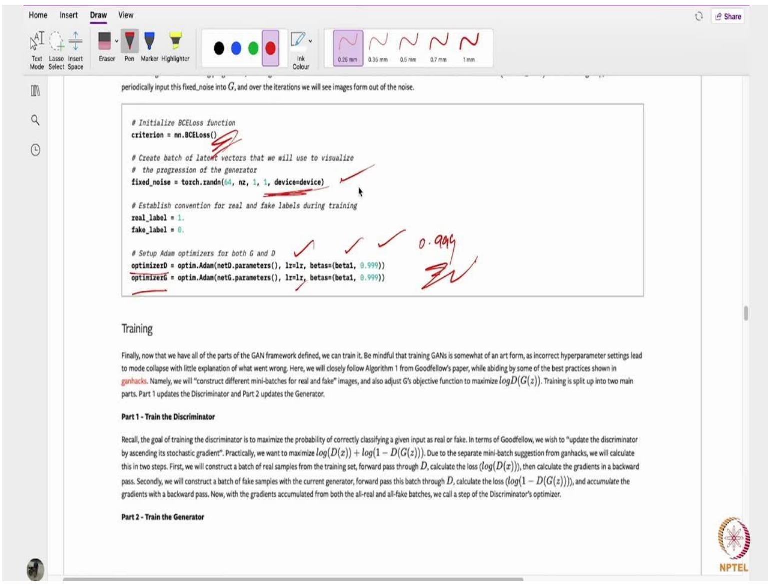
Next thing is discriminator same thing here you will feed the input images 3 64×64. And the output will be probability whether this particular image belongs to that what probability score is particular image belongs to real image that is output of this discriminator same thing here as we have followed leaky ReLU activation functions, batch normalization and strided convolution, strided convolution. Now, this is a discriminator block earlier we have discussed about generator block, before that we have just to prepare the data set and convert it into data loader in terms of batches.

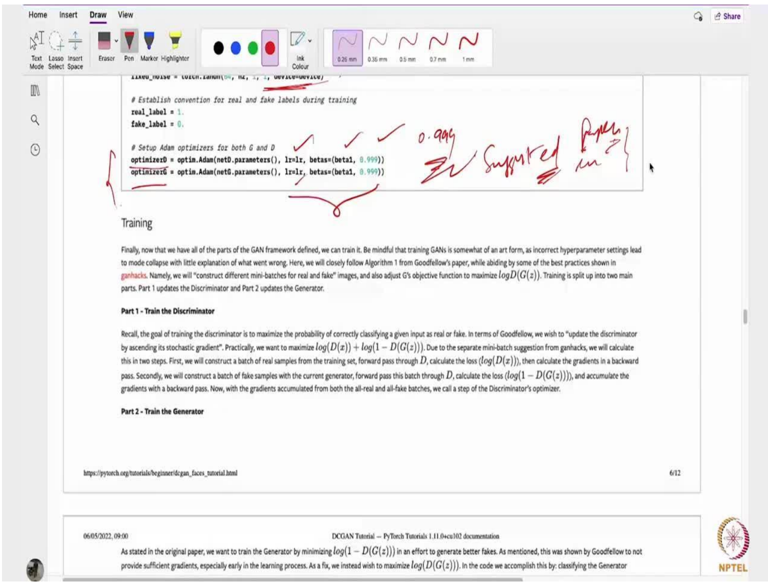
(Refer Slide Time: 41:40)



Now, coming back to loss functions as I have already mentioned, we are going to use binary cross entropy loss only this loss function.

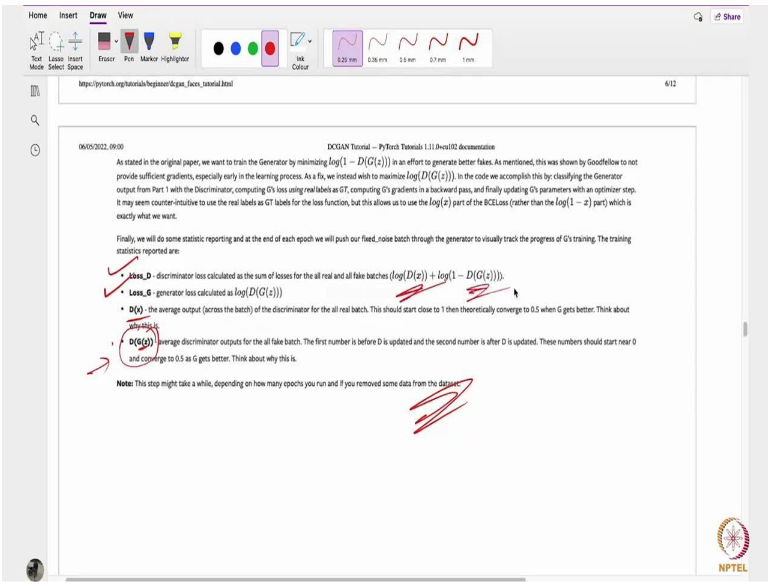
(Refer Slide Time: 41:47)





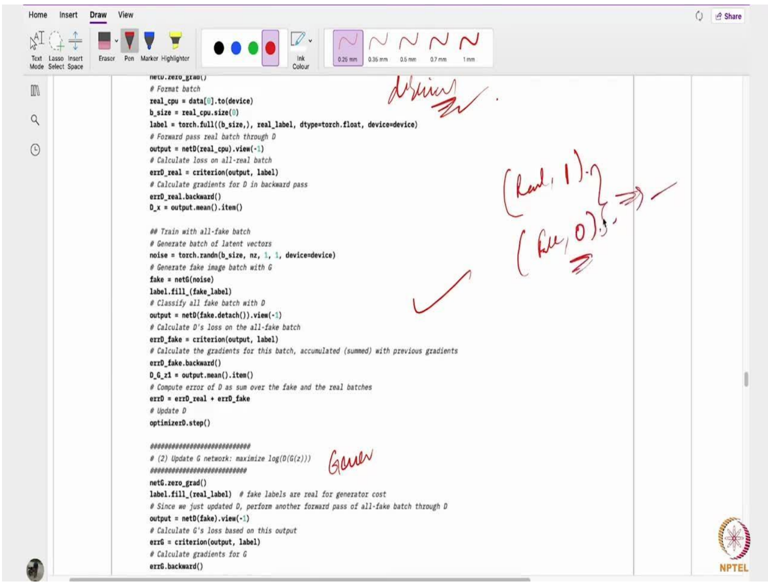
Here they have used the fixed noise this is for the sake of seeing how generator is generating the images real label has given as 1 and fake label has given as 0. Here you can see 2 optimizers will be given individually for generator and discriminator, you can bring a lot of changes here as you can play around with them hyper parameters, this β_1 value and a β_2 values are suggested in the paper actually we have taken the same values suggested in paper they use the same values.

(Refer Slide Time: 42:21)



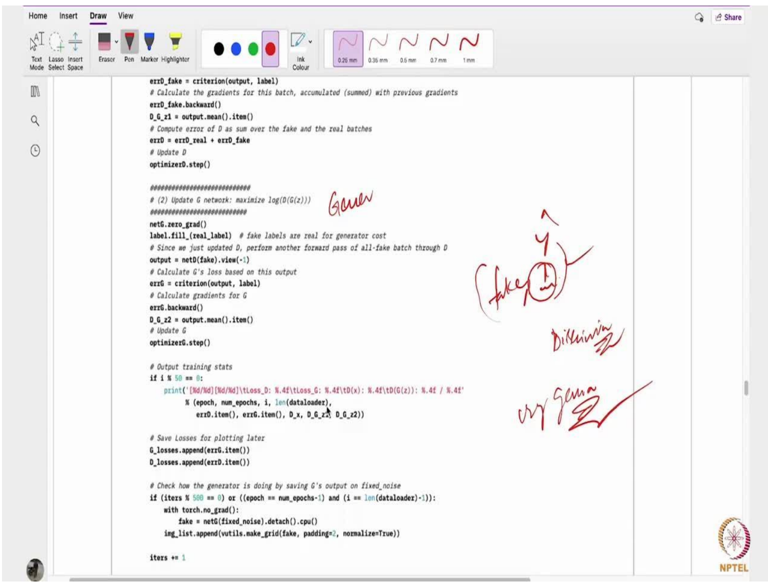
Now, coming back to training in the training section you can see train the generator, train the discriminator and compute the loss obtained in discriminator loss obtained in generator. D(x) indicates discriminator output then we feed input sorry real image from the data set whereas G(z) indicates output if generated by the generator and it has been fed into discriminator. So, D(G(z)) indicates the probability score that is output given by the discriminator when we feed the fake image generated image generated by the generator this is what is exactly these formulas indicate the terminology.

(Refer Slide Time: 43:13)



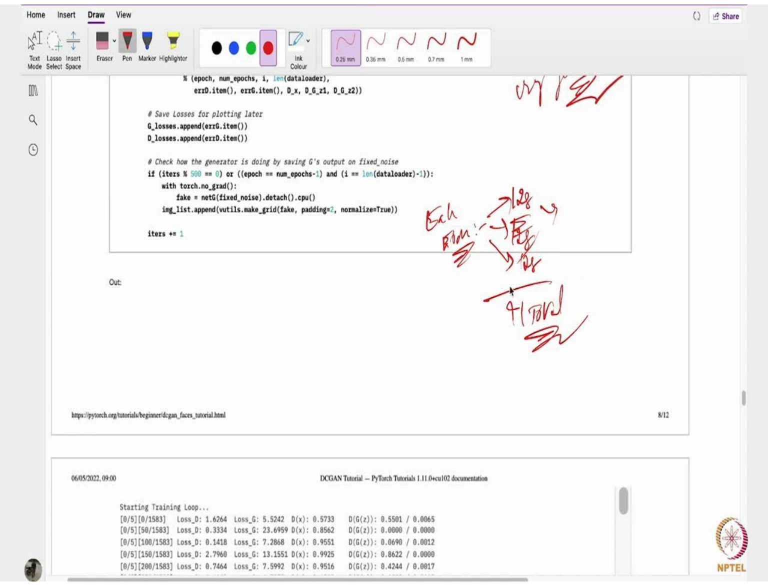
Next thing let us move on this is entire, this one is like training loop this is entire training loop here with respect to generator and this loop has been written with respect to discriminator. I have already told you in discriminator section the real images and the target values will be kept 1 whereas if you give fake images, the output target value will be kept 0 and the loss functions were calculated accordingly.

(Refer Slide Time: 43:38)



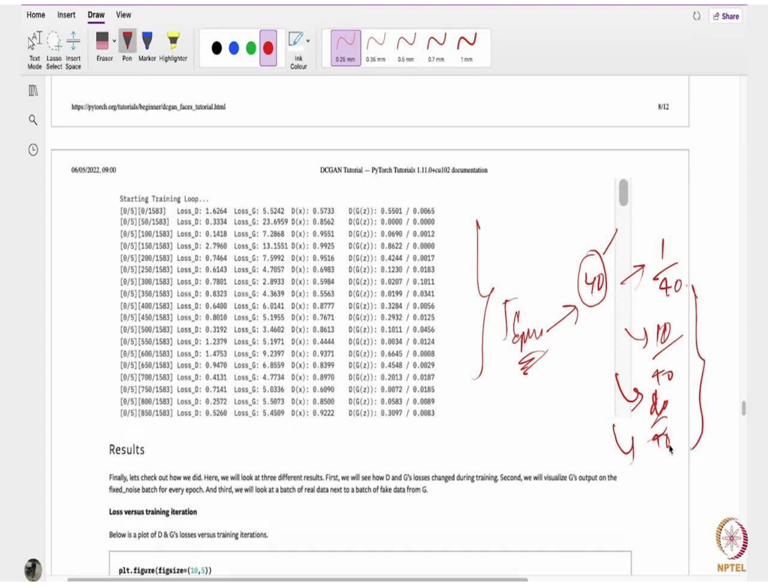
Whereas in terms of this generator what we do for the fake image itself only there they keep the target value as 1. So that we are fooling the discriminator here using generator, using generator this is how it goes and the lost values, error values everything is computed.

(Refer Slide Time: 44:01)



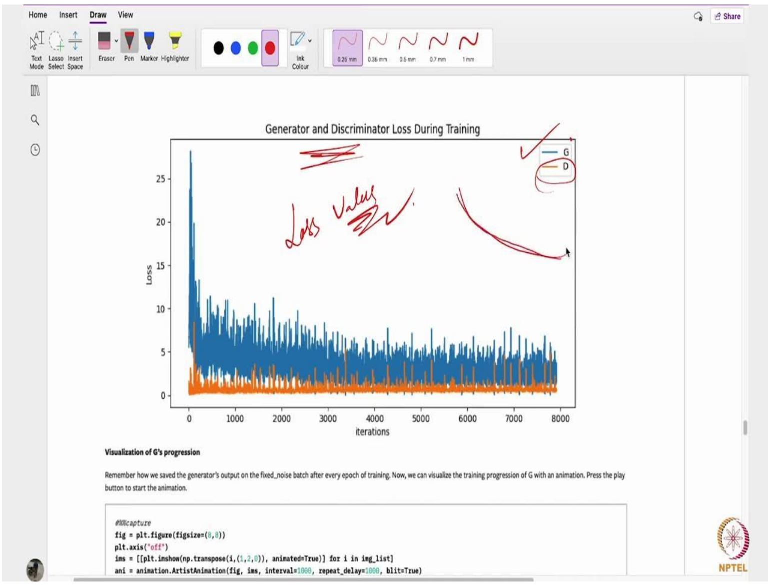
Suppose, I have already mentioned So, in our case our data set we have around some 5 8 something something like 128 batch size we have taken that means in each epoch, each epoch contains a total of 41 total, 41 batches or 41 iterations.

(Refer Slide Time: 44:27)



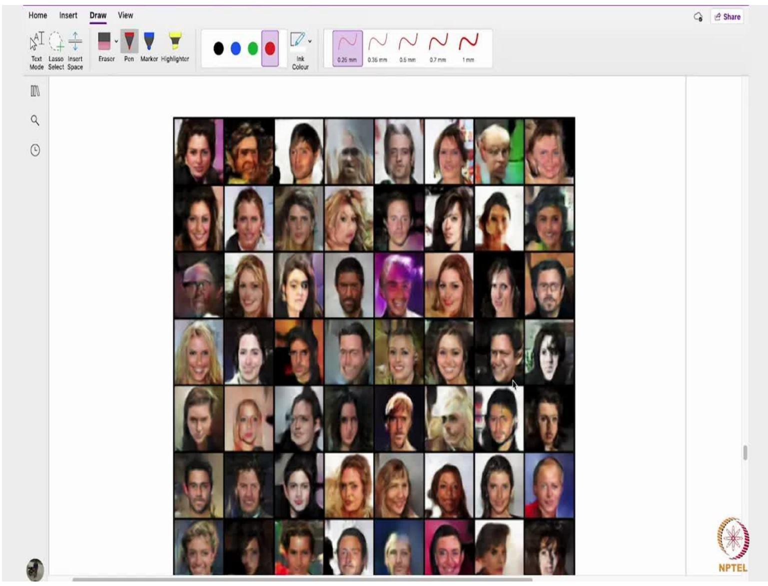
So, if you observe the results here, here they have use 5 epoch for each and every epoch, we will have 40 iterations out of 40 iterations we are going to see only for first one and 11th one or 10th or 20th iteration likewise we will see the output once I will show you how to run the code also. But as an explanation sake I am saying.

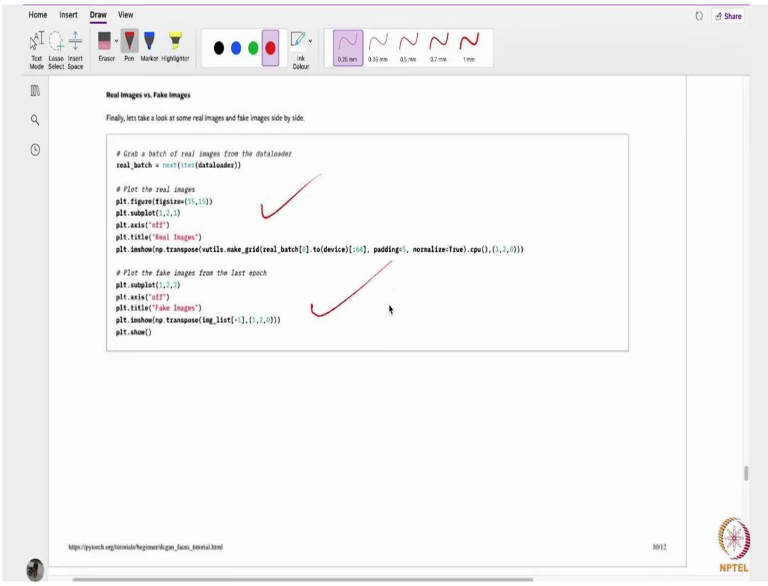
(Refer Slide Time: 44:56)

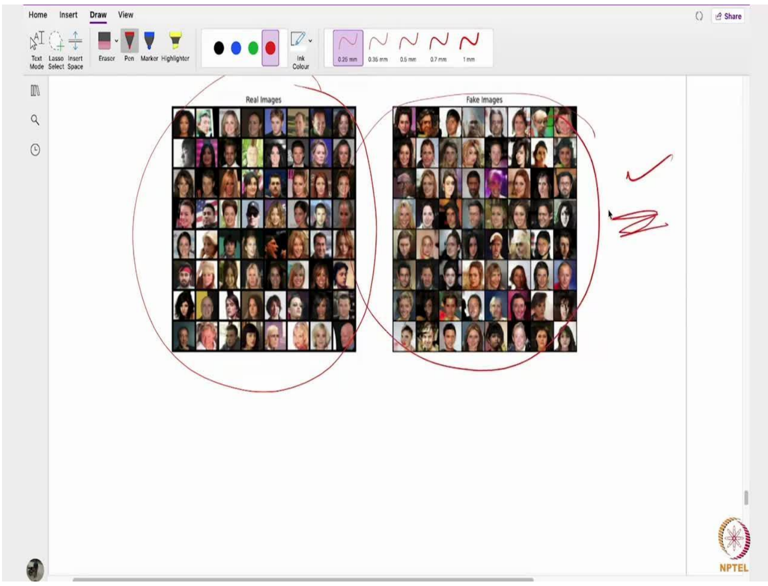


Now coming back to the results. Here you can see the loss functions loss value, discriminator loss and generator loss both are computed here, this color indicates discriminator and a blue color indicates generator loss. End of the day both the loss should decrease.

(Refer Slide Time: 45:24)



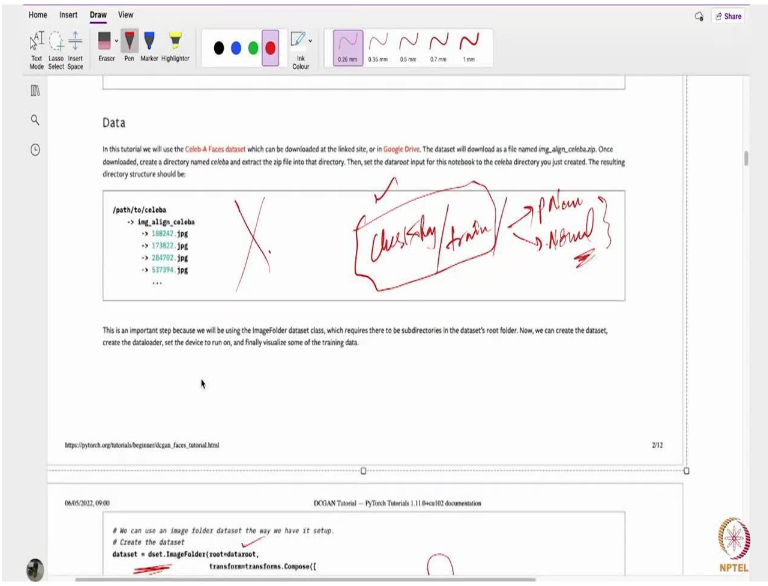


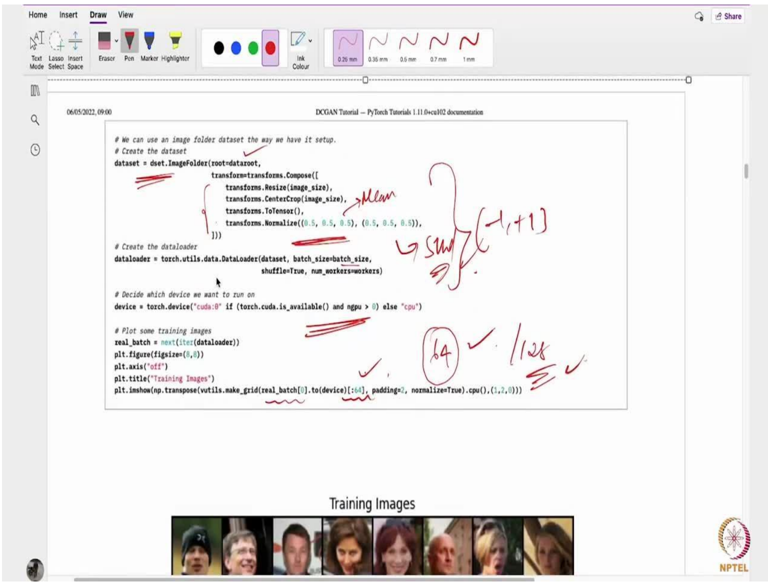


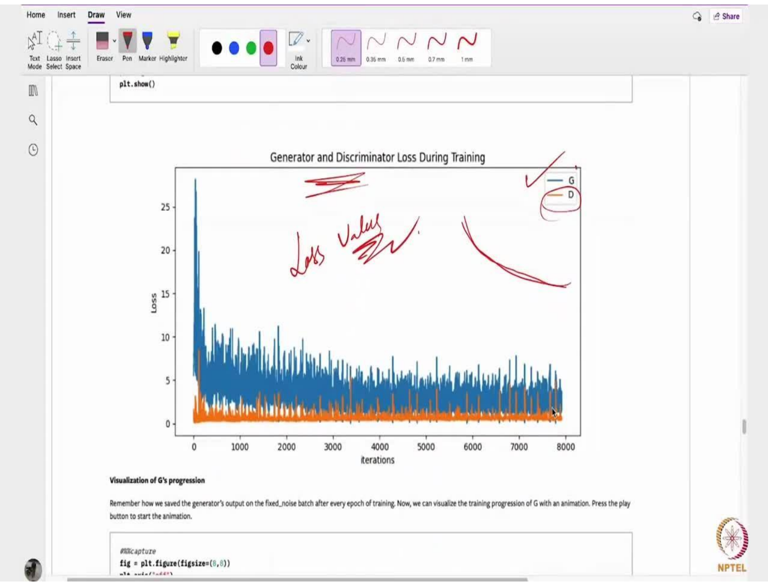
Now, this is just for the sake of how generated is progressing by taking the input random noise vectors and outputting see here, all these images are not completely, not completely drawn, you can see slight modifications are there, that means they are at the initial stage of trainings, but as you pass by if you do very good training, all of them will come in to a very good pictures later on.

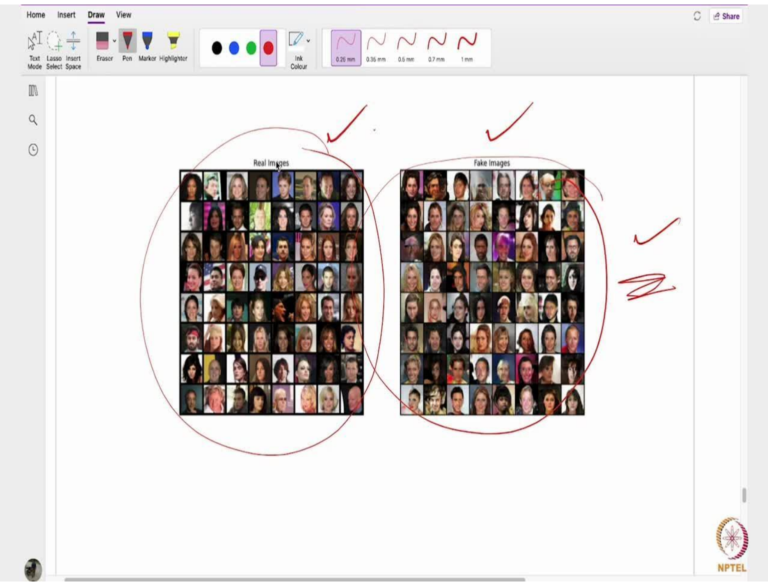
At the end of this training, after playing so, many ways after playing around with the hyper parameters during the training, you just have to verify the results here, here they have drawn the real images and fake images as an output. This is what shown here, this is the final thing, this is the complete explanation sake.

(Refer Slide Time: 46:15)





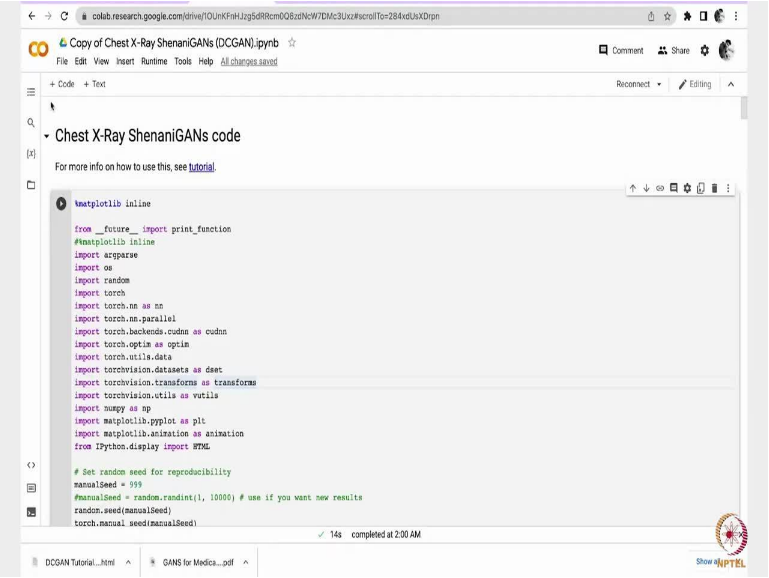


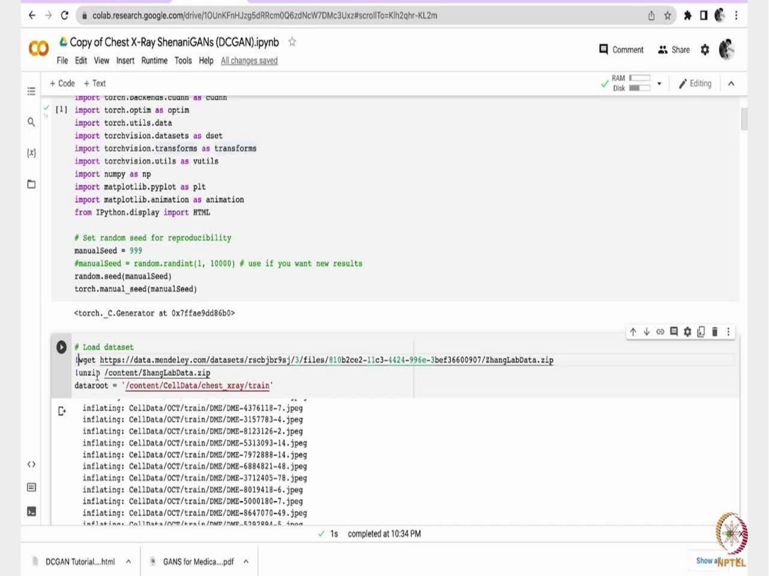


As a summary note, let me explain again, in summary, we have to report all the modules, here we have to set the hyper parameters, we have to set the hyper parameters, here the data set, sub directory has to be kept in a certain format using this data set and data loader we are going to convert the entire image data set into batches.

Later on the training loop will come, generator, discriminator, loss functions, training loop and we have drawn the graph of the generator and discriminator loss during training and at the end we have drawn the images generated by generator and real images that are present in the training data that is it, nothing else.

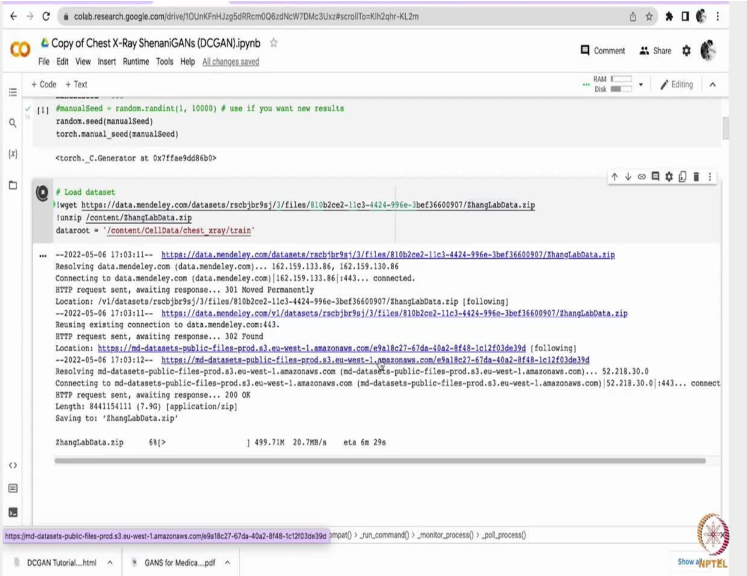
(Refer Slide Time: 47:13)

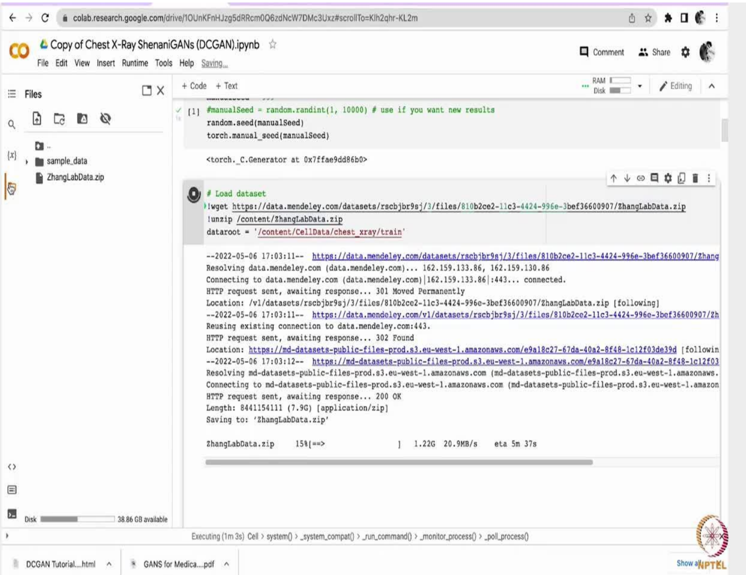


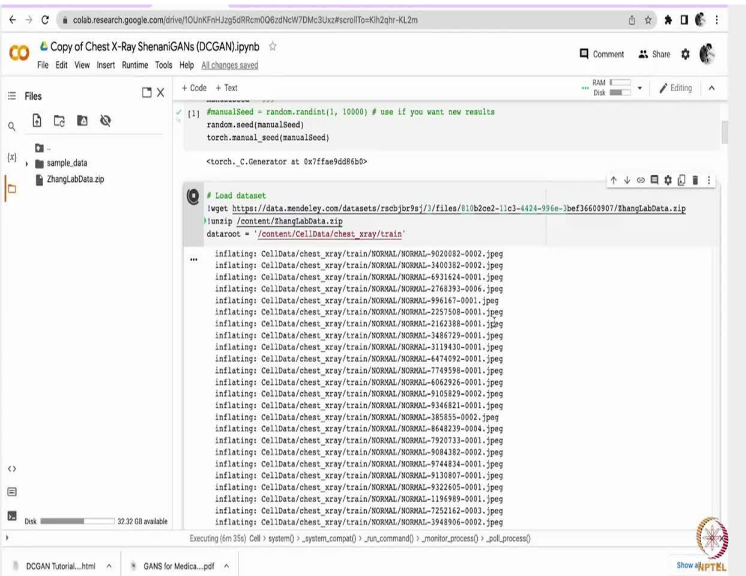


Now, let us get back to code how exactly you can use this notebook this is the code actually, see here I will run each and everything just focus. Once you open the Google colab, you can see this one, it takes some time you need some patients. First part is done, here we are going to download the data set. If you observe clearly from this link, from this link, we are downloading the data that zip data has been unzipped here. See here, this file name containing data set has been downloaded zip file that zip file has been extracted.

(Refer Slide Time: 48:31)

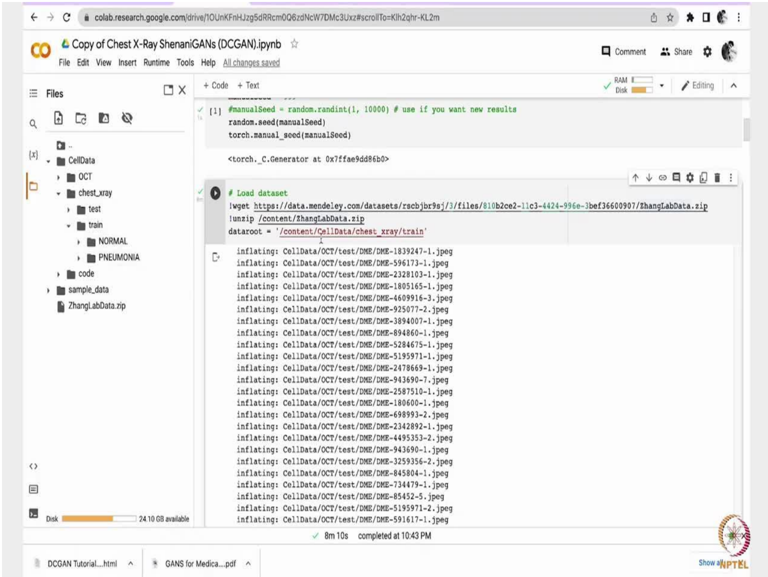

Copy of Chest X-Ray ShenaniGANs (DCGAN).ipynb
File Edit View Insert Runtime Tools Help All changes saved
#manualSeed = random.randint(1, 10000) # use if you want new results
random.seed(manualSeed)
torch.manual_seed(manualSeed)
<torch._C.Generator at 0x7ffae9dd86b0>
Load dataset
!wget https://data.mendeley.com/datasets/rscbjbr9sj/3/files/810b2ce2-11c3-4424-996e-3bef36600907/ZhangLabData.zip
!unzip /content/ZhangLabData.zip
dataroot = '/content/CellData/chest_xray/train'
HTTP request sent, awaiting response... 301 Moved Permanently
HTTP request sent, awaiting response... 302 Found
HTTP request sent, awaiting response... 200 OK
Length: 8441154111 (7.9G) [application/zip]
Saving to: 'ZhangLabData.zip'
ZhangLabData.zip 6%[>] 499.71M 20.7MB/s eta 6m 29s


Copy of Chest X-Ray ShenaniGANs (DCGAN).ipynb
Files
sample_data
ZhangLabData.zip
Load dataset
!wget https://data.mendeley.com/datasets/rscbjbr9sj/3/files/810b2ce2-11c3-4424-996e-3bef36600907/ZhangLabData.zip
!unzip /content/ZhangLabData.zip
dataroot = '/content/CellData/chest_xray/train'
Length: 8441154111 (7.9G) [application/zip]
Saving to: 'ZhangLabData.zip'
ZhangLabData.zip 15%[==>] 1.22G 20.9MB/s eta 5m 37s
Executing (1m 3s) Cell > system() > _system_compat() > _run_command() > _monitor_process() > _poll_process()


Copy of Chest X-Ray ShenaniGANs (DCGAN).ipynb
Files
sample_data
ZhangLabData.zip
Load dataset
!wget https://data.mendeley.com/datasets/rscbjbr9sj/3/files/810b2ce2-11c3-4424-996e-3bef36600907/ZhangLabData.zip
!unzip /content/ZhangLabData.zip
dataroot = '/content/CellData/chest_xray/train'
inflating: CellData/chest_xray/train/NORMAL/NORMAL-9020082-0002.jpeg
inflating: CellData/chest_xray/train/NORMAL/NORMAL-3400382-0002.jpeg
inflating: CellData/chest_xray/train/NORMAL/NORMAL-6931624-0001.jpeg
Executing (6m 35s) Cell > system() > _system_compat() > _run_command() > _monitor_process() > _poll_process()

Later on we are going to keep the data root value. See how it goes. Let us download the data set. It takes some time.

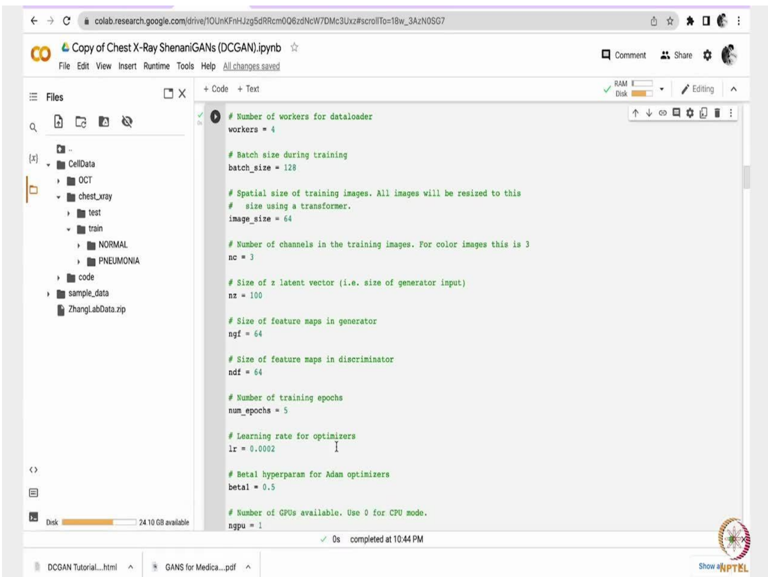
(Refer Slide Time: 56:45)

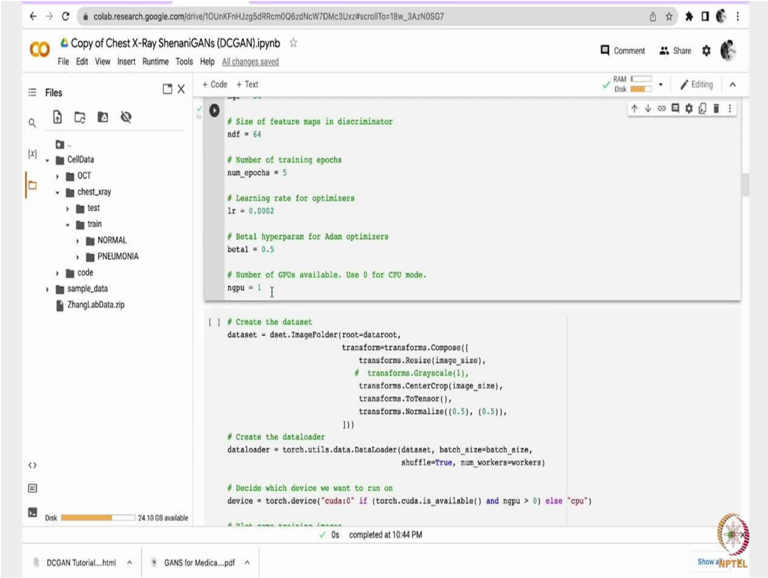


Now, you can see that after downloading the data, we have personal details downloaded that is failing sorry. The thing is cell data has been extracted from this particular file name, if we observe in the cell data we have was OCT, chest X ray and code. But as of now, we will use chest X ray dataset.

And in chest X ray data set my aim is just to focus on training data, train folder only and inside see here, this is the content to extract the subdirectory of the train, just have to click here and copy path and that path will be this one. So, do this one, one should download the data and everything has been done.

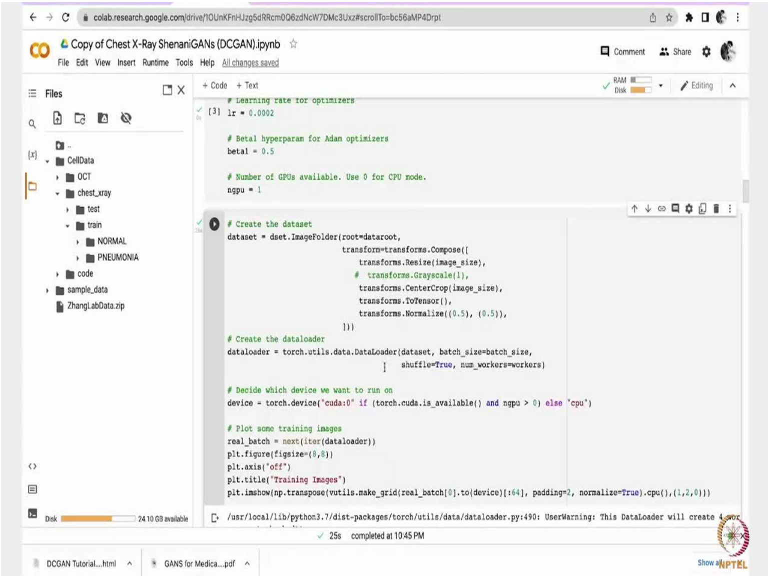
(Refer Slide Time: 57:34)

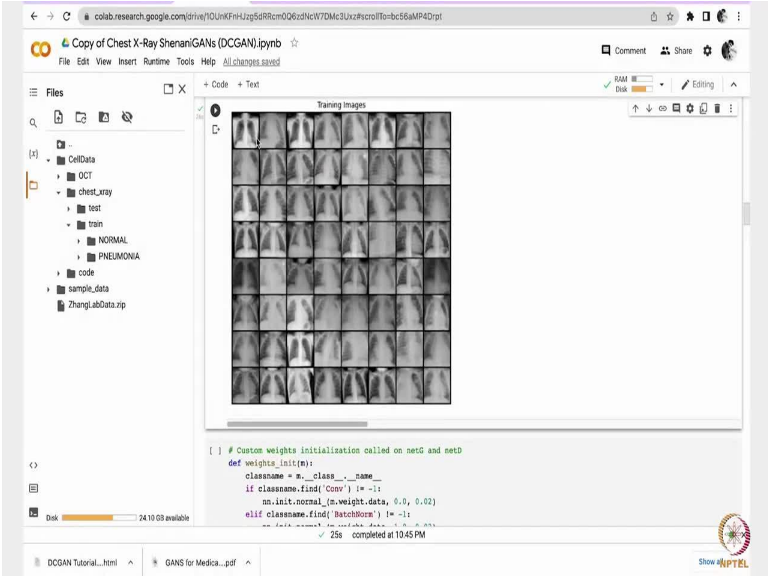


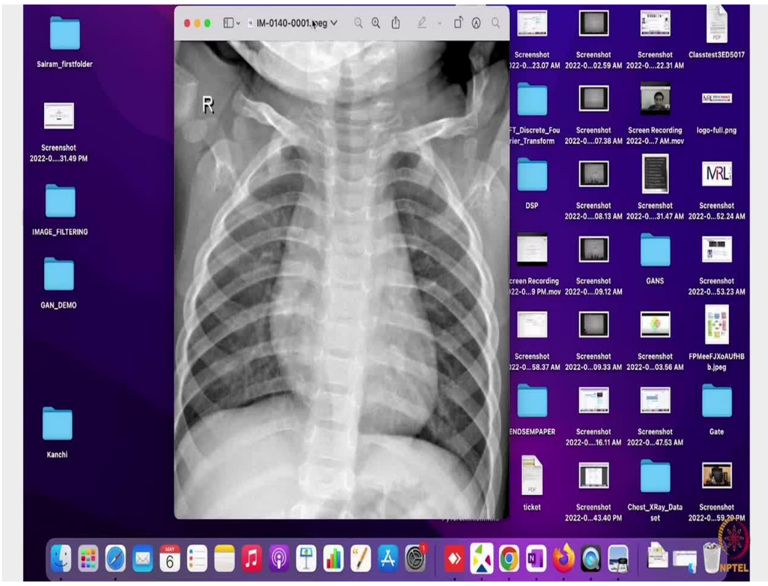


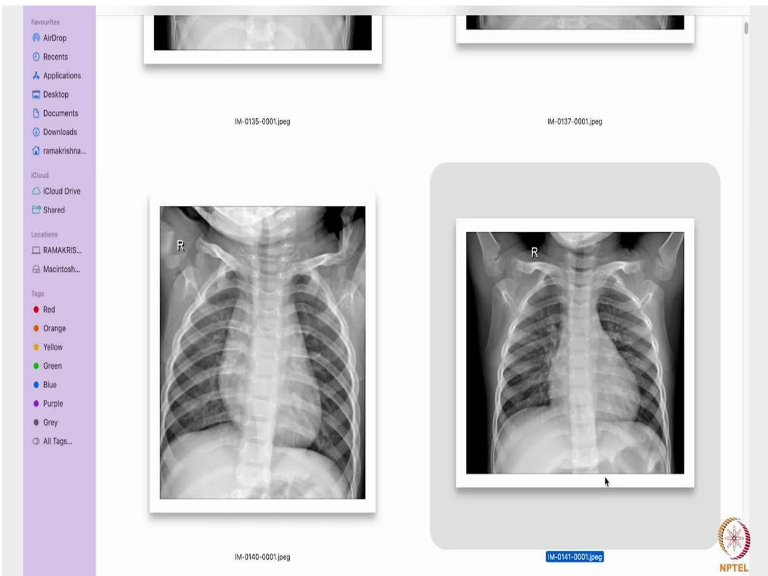
Now, let us get back to setting the hyper parameters everything let us run this one, number of channels are 3 GB images next ngf ngf view, next thing is learning rate for both optimizers has been kept as 0.0002 later on when we, we can play around that is not an issue β_1 has been kept as 0.5 for batch optimizers, GPU is number of GPUs are available suppose if you have 4 or 5 GPUs also you can do for parallel processing also. Suppose the data set you have is very large and you have the available computational facility of having 4 or 5 GPUs you can go for parallel processing that has also been accepted.

(Refer Slide Time: 58:12)



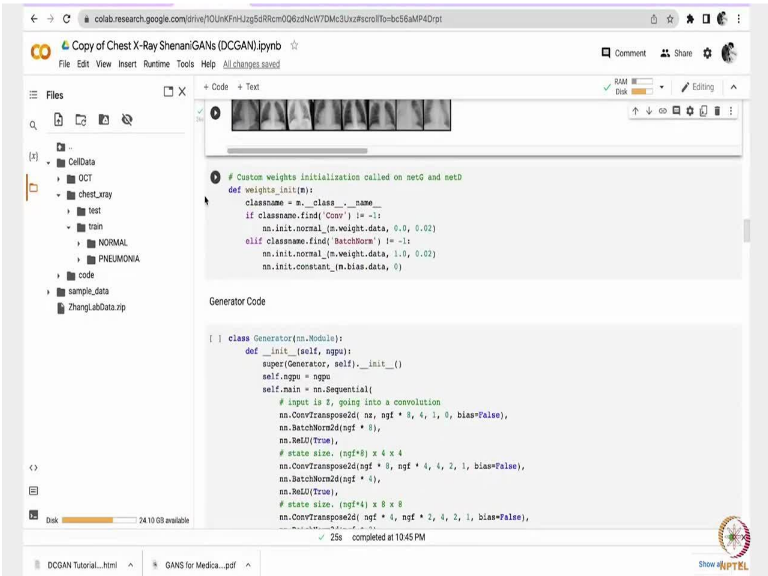

Copy of Chest X-Ray ShenaniGANs (DCGAN).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Files
CellData
OCT
chest_xray
test
train
NORMAL
PNEUMONIA
code
sample_data
ZhangLabData.zip
Training Images
# Custom weights initialization called on netG and netD
def weights_init(m):
classname = m.__class__.__name__
if classname.find('Conv') != -1:
nn.init.normal_(m.weight.data, 0.0, 0.02)
elif classname.find('BatchNorm') != -1:
completed at 10:45 PM
DCGAN Tutorial....html
GANS for Medica....pdf

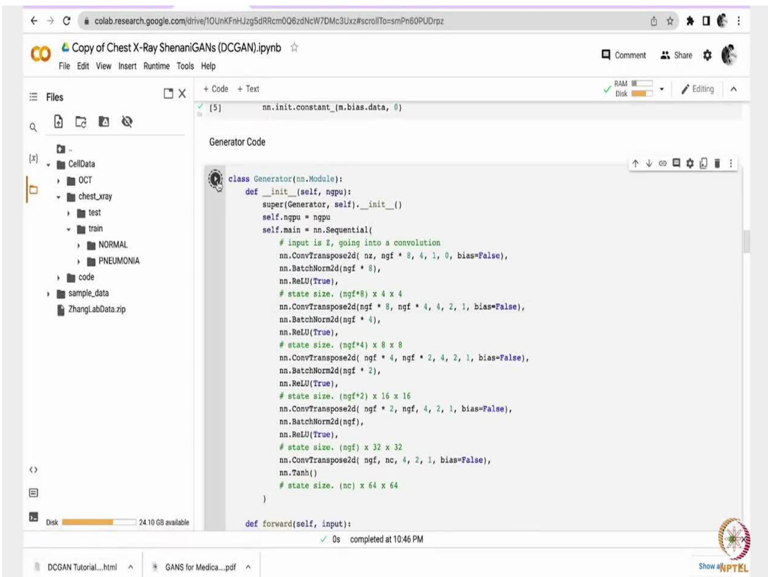

IM-0140-0001.jpeg
R
Sairam_firstfolder
IMAGE_FILTERING
GAN_DEMO
Kanchi
GANS
Chest_XRay_Data set
Gate


AirDrop
Recents
Applications
Desktop
Documents
Downloads
iCloud Drive
Shared
Red
Orange
Yellow
Green
Blue
Purple
Grey
All Tags...
IM-0135-0001.jpeg
IM-0137-0001.jpeg
R
R
IM-0140-0001.jpeg
IM-0141-0001.jpeg

Now go for dataset creation, go for dataset creation, it takes some time because in batches there are a lot of difference in batches. Now, the data loader has been generated at the same time we are extracting some 64 images and they are all will be shown here. See here other chest X ray images, there all digital images I will show you see here inside the training data. Normal means see here this is very huge in space they are above 1000 also I mean size of the image, know the pixel. You have seen this one every data contains 1000 by 129, 2930 like 4 digit numbers on the rows and both columns also.

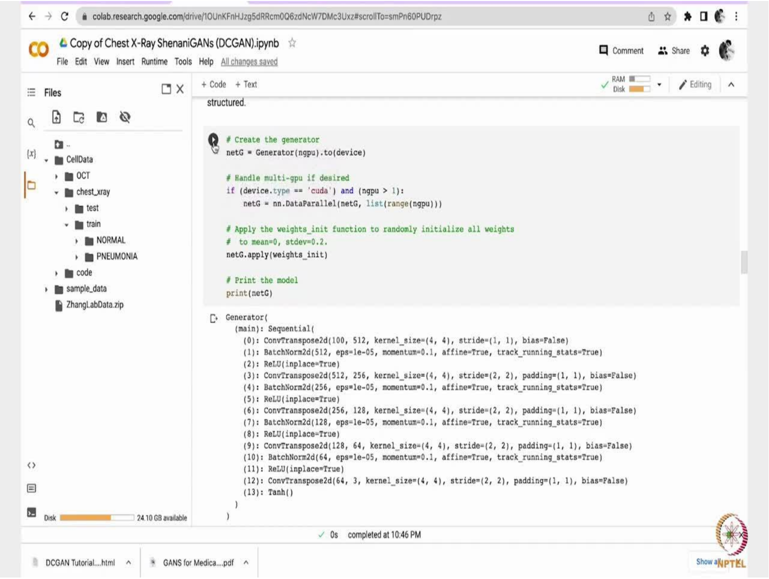
(Refer Slide Time: 59:36)





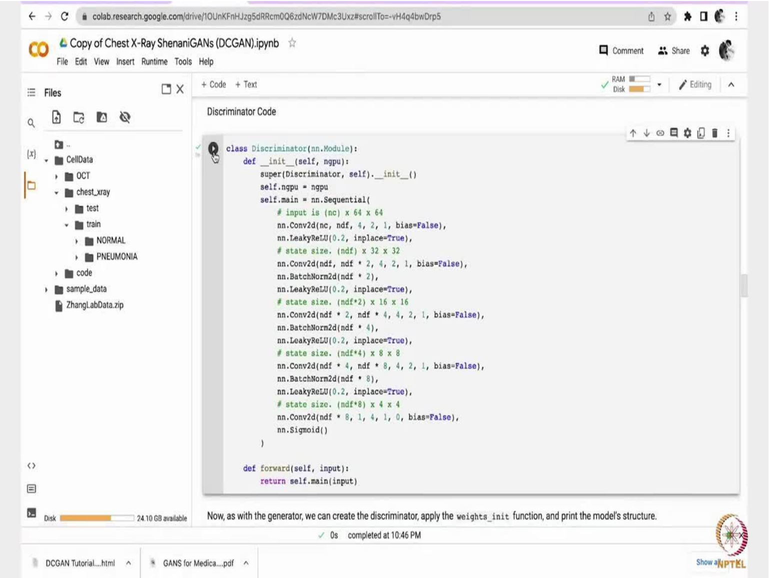
Now coming back to custom weight initialization on generator and discriminator. This is generator code.

(Refer Slide Time: 59:50)



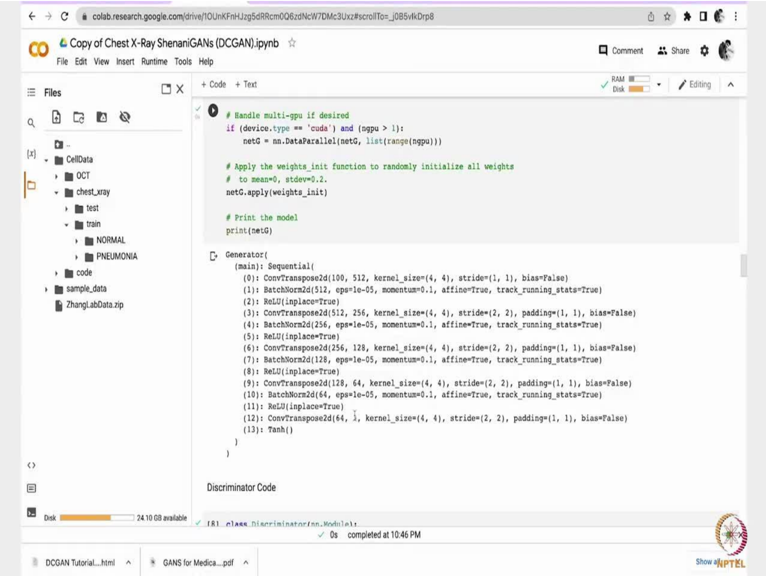
Next thing is we will convert the model of generator into GPU model. The initialization weights are also done here.

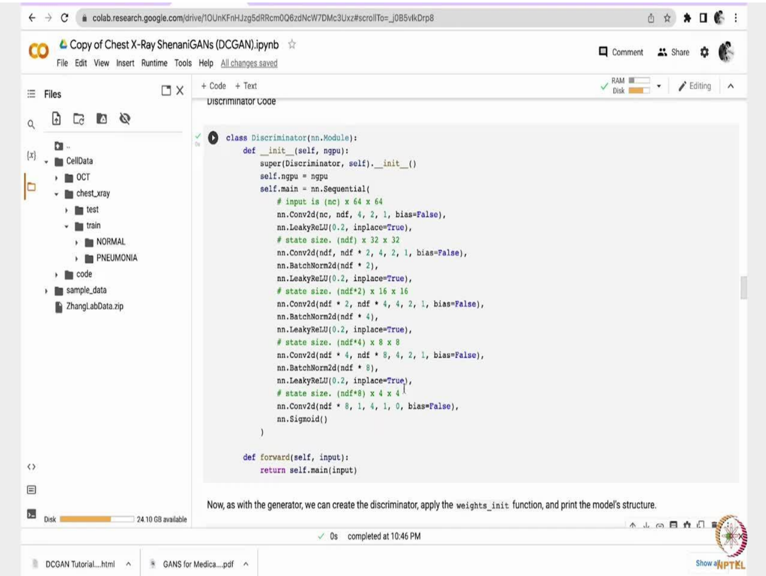
(Refer Slide Time: 59:56)



Now coming back to discriminator block. It has been executed similarly discriminator.

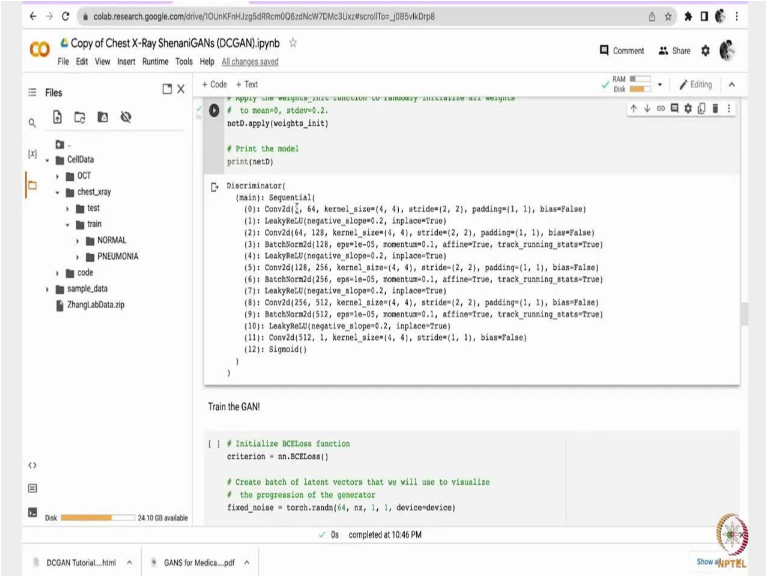
(Refer Slide Time: 60:11)





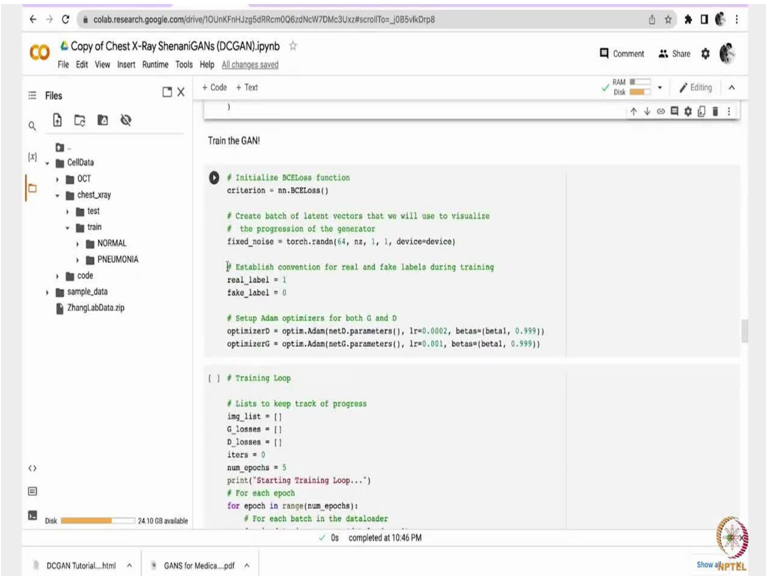
Observe here generator this is 100 dimensional input vector and outputting 3 channel image whereas the discriminator it is taking the input as a number of channels cross 64×64 here we have increase around only 64× 64 and is giving an output as probability score see here.

(Refer Slide Time: 60:33)



Input as 3 channel image and outputting sigmoid sigmoid output is nothing but 0 to 1 rounded of 2. So, we are considering them as probability scores. Now, discriminator block is done, generator block is done and these models have been sorry both networks have been moved to device GPU devices. Next thing model converting model into GPU device in the sense all the parameters that have been converted into GPU tensors.

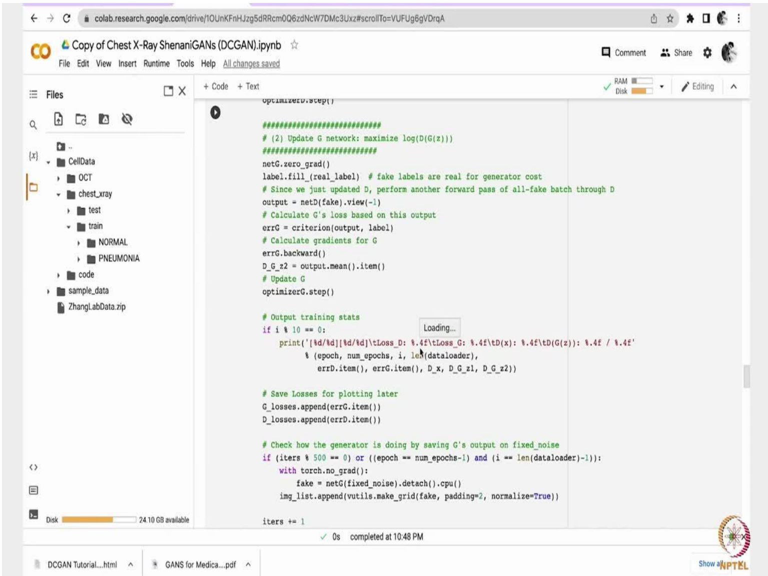
(Refer Slide Time: 61:04)

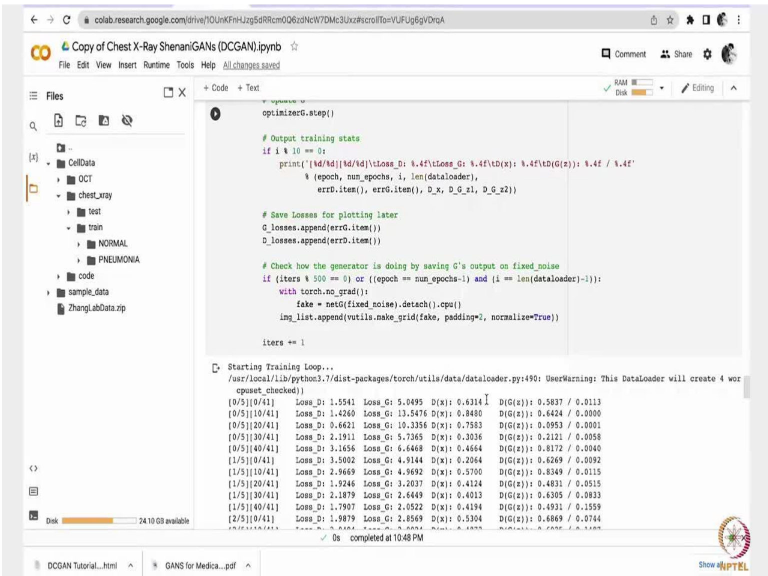


Now, coming back to training the GAN, here we are used the lost binary cross entropy loss only it is a noise this is for visualization purposes focusing on generator later on you will understand. The real label is 1 optimizer and discriminator see here, how changes the value to earlier in the case of discussion I have told both the learning rates have been kept to 0.0002 and 0.0002.

But here, after doing some modifications or playing around with the values here, I came to know that maybe this value is giving somewhat good understanding of hyper parameter. So I kept this one but you can play around with first, you can start with both 0.0002, 0.0002 and later on you can do some changes to learning rate.

(Refer Slide Time: 61:50)

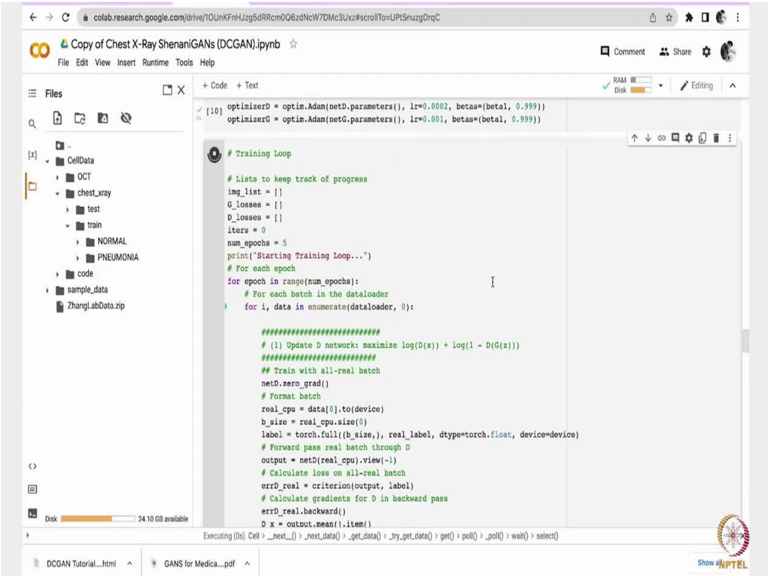


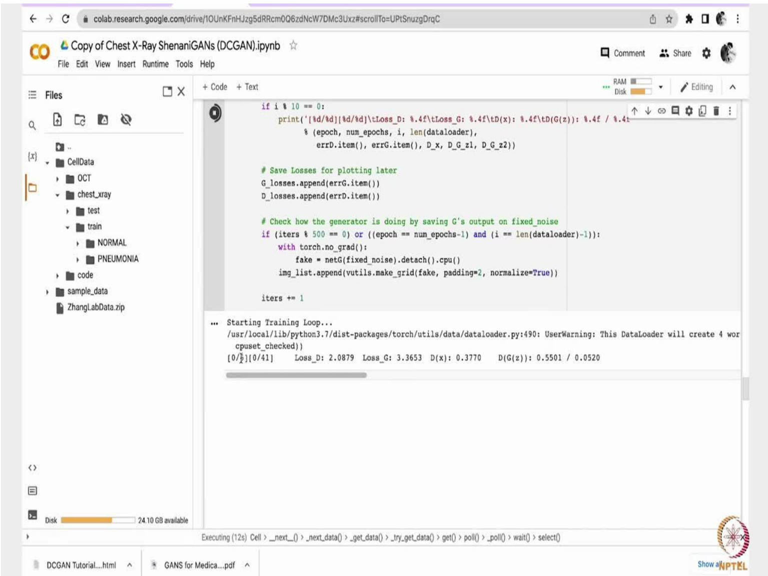


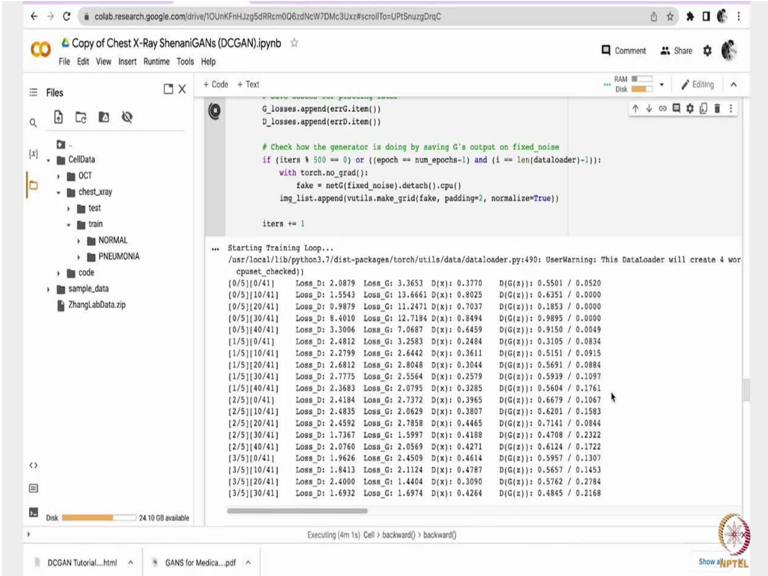
Let me run this one. Now, the final thing training loop, generator updation and discriminator updation, you can see here, as I told you earlier, we have only 41 batches, each batch contains of some 128 images out of this 41 also and in each and every epoch I have 41 batches, I just want to show the, I want to see the loss value at after every 10 iterations that is why I have kept an i /10 = 0.

So, if you want to see out of these 41 iterations you may want to see for every 5 iterations just keep a 5 value here. Suppose 20, just keep a 20 value here. So, we are observing the discriminator loss, generator loss, output of the discriminator, at the same time output of the discriminator run fake image is given and at the same time real images is given, this is how the entire thing goes on.

(Refer Slide Time: 62:56)



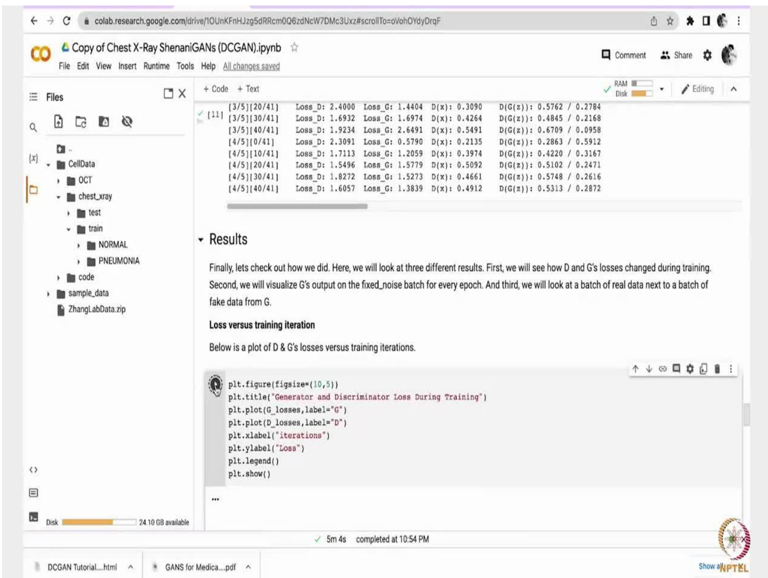


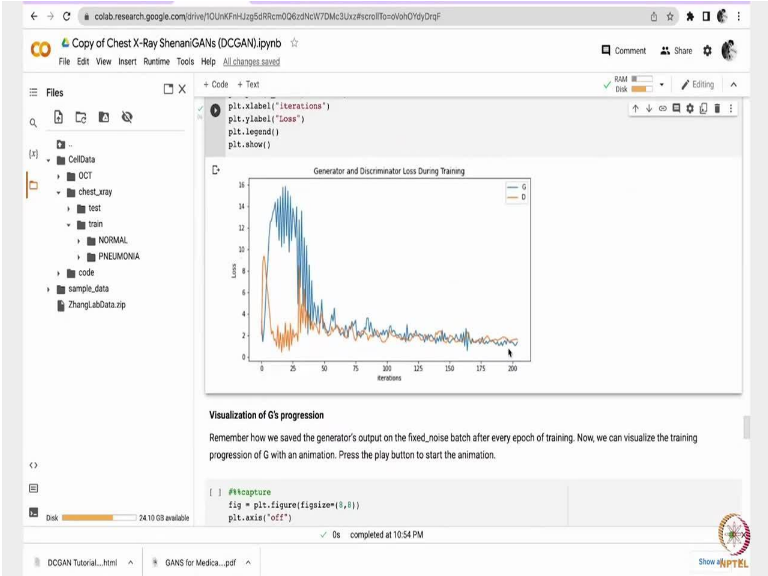


Now, let us run this value. So, let us run this training loop, let us see how the training continues. See here. This is the first step out of 5 epochs and inside that 5 epochs, sorry, inside that first epoch only, we are observing the first iteration out of 41 iteration see here.

This is the 10th iteration. These are the values of loss values, discriminator loss values and the generator. By observing these loss values we just have to observe these values and make a decision, 4th epoch is going now, here given number of epoch is 5 epochs only. Now, the training has been done.

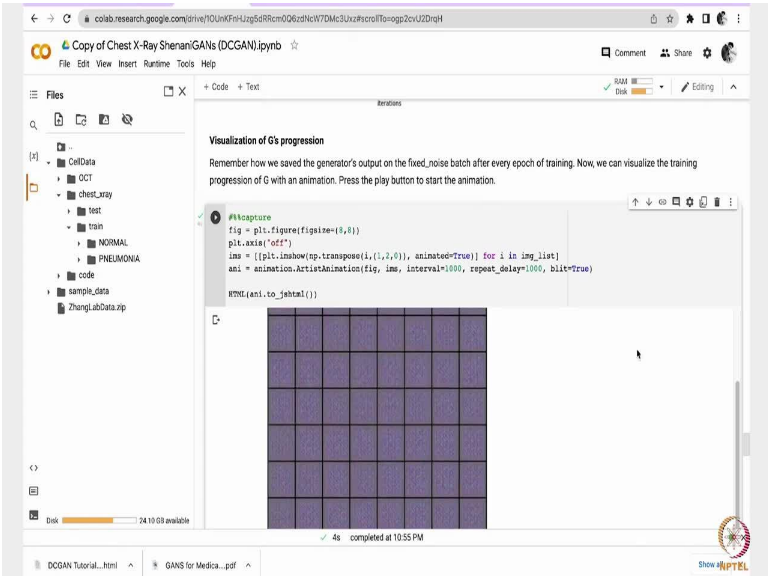
(Refer Slide Time: 68:13)

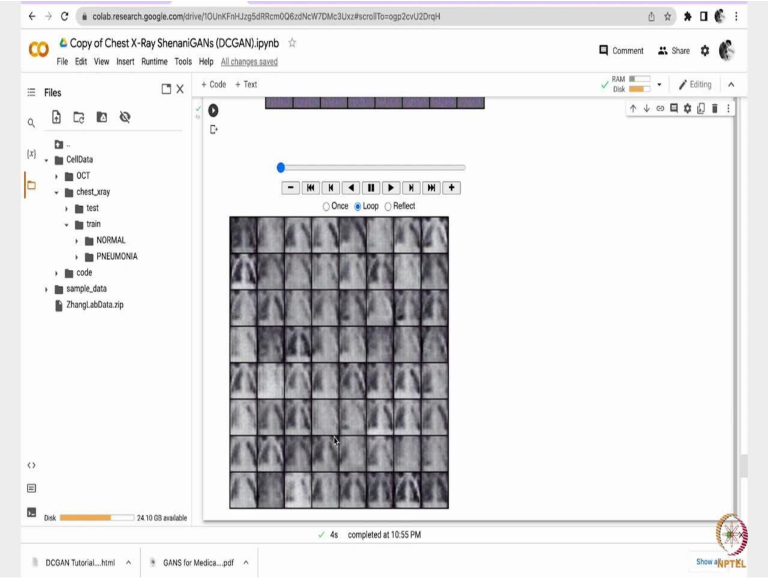




Now, let us see how the loss function, loss cause will be, see here both the losses are going down. But here you can see a sudden jump of generator from the discriminator loss value. But as the days pass by as the number of epochs increasing since I have used it very very less number of epochs you can go and change that to later on you can play around let us see.

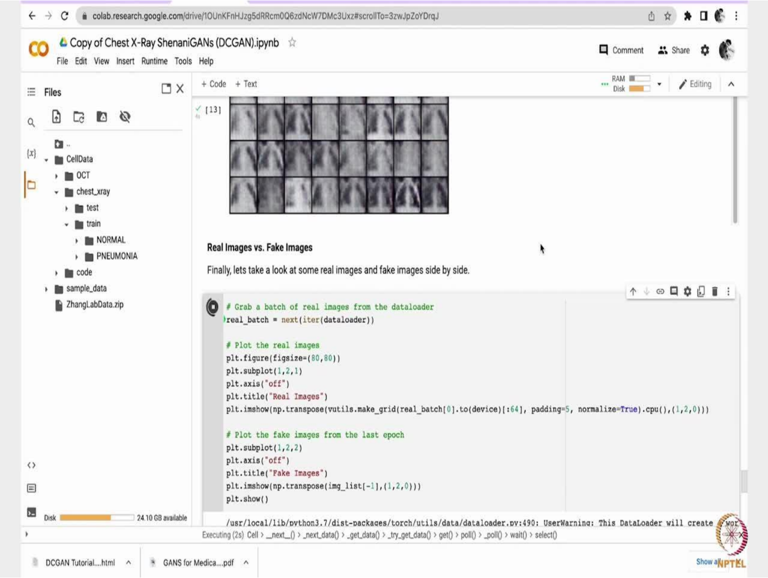
(Refer Slide Time: 68:44)

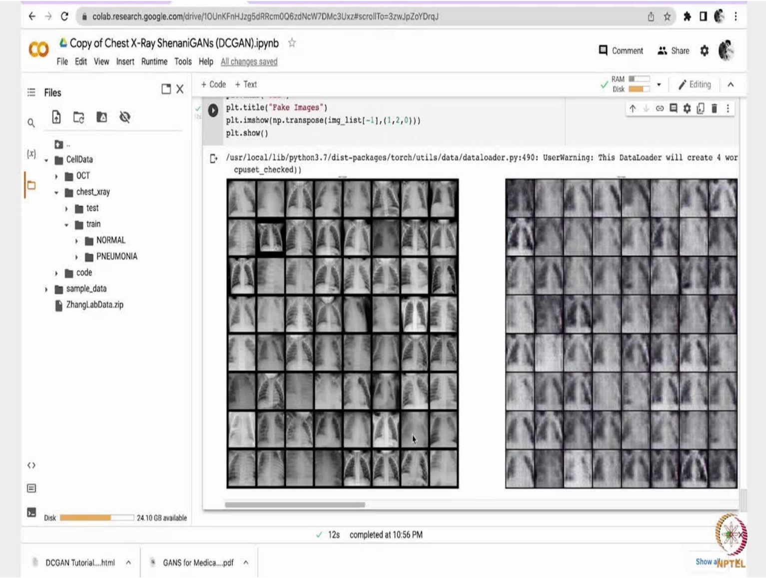




This progression I mean here in each and every loop we have the noise vectors it is corresponding, see here you can see the images are similar to chest X ray images whatever, but not almost close to them, but you can see those replications somehow you can identify the.

(Refer Slide Time: 69:08)

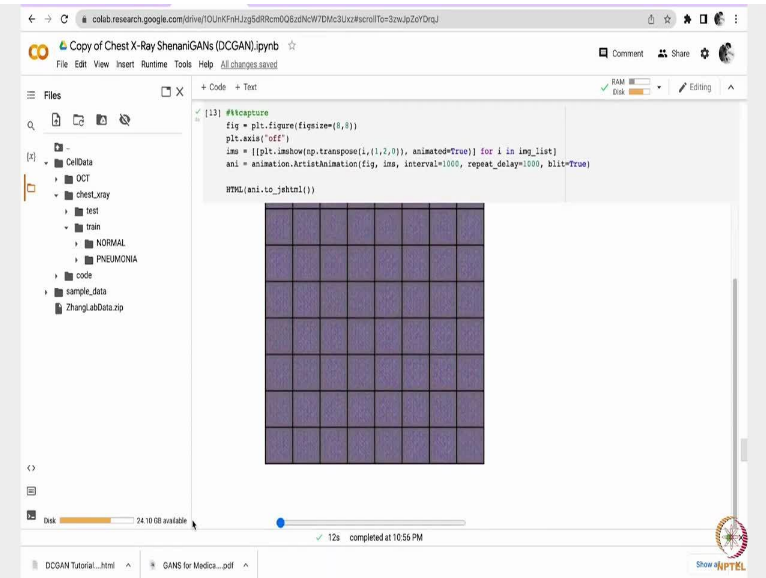




Let us see first 64 images were exactly the output has been taken out, see here these are all left side you can see all the real images whereas the right side you have only fake images. So, now the entire task is done.

Now, our turn is to modify hyper parameters. So, that you should get this distribution or this chest X ray images almost similar to these values. Then we can say that generator has been trained very well and we can save that particular model. You can now randomly give a random noise vector with a particular generator saved model and you will definitely get a chest X ray image out of it. So, the entire problem has been done.

(Refer Slide Time: 70:20)



Now, as a summary note I can say that it is completely for any model hyper parameter tuning is very important that is where your skills matter. Now, coming back to suppose if you are, you should have using chest X ray data, you may have some other data and keep the directories like this inside the chest X ray go to train data inside this suppose you have some N number of classes, keep all the N number of classes here and this is how the directory should be present. So, I can see that the task has been done, thank you.