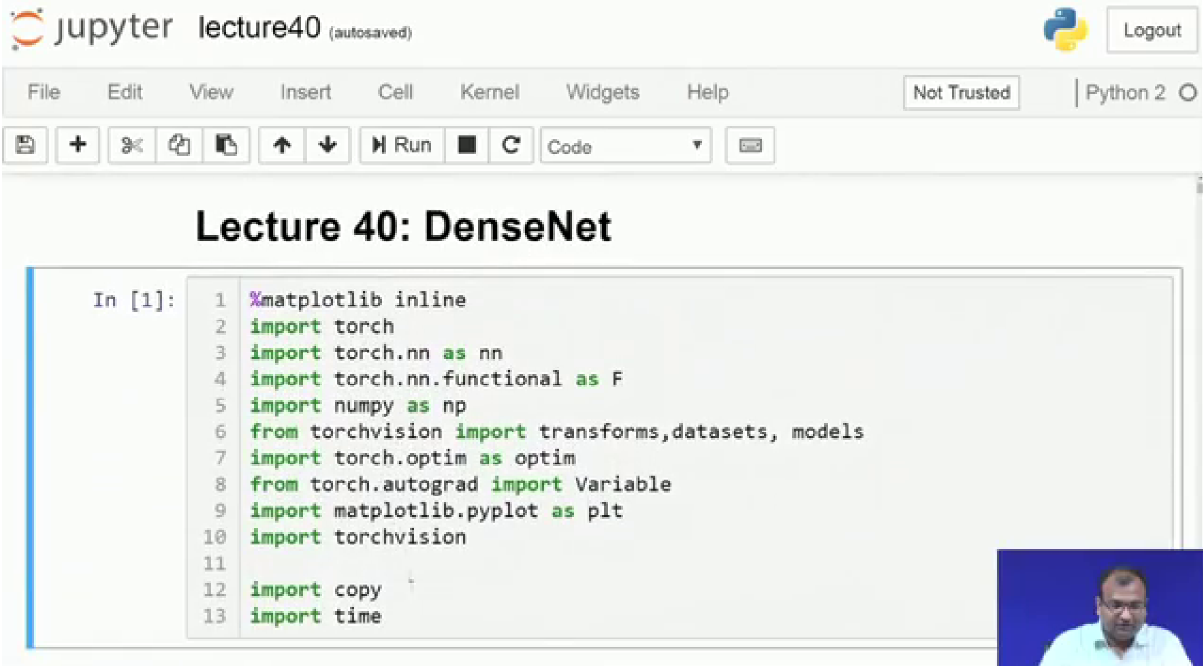
Deep Learning for Visual Computing
Prof. Debdoot Sheet
Department of Electrical Engineering
Indian Institute of Technology, Kharagpur

Lecture – 40
DenseNet

[noise]

So welcome ah today we are going to ah learn about densenet, so while in the last few ones we had gone down very much deeper, so we started with googlenets. And then eventually we went on to ah the newer kind of once where you could have some sort of ah connections from a previous layer to the next layer.

(Refer Slide Time: 00:29)



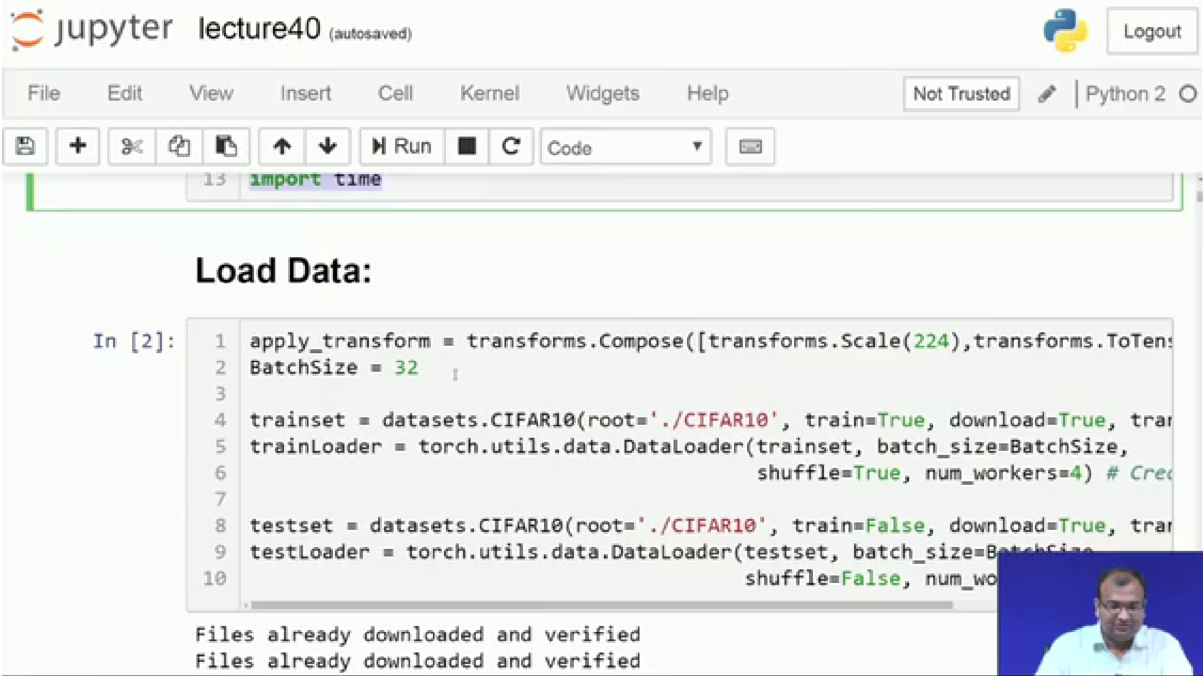
So one of them was called as a residual network which all resnet and thats what we had done in the last one. So the whole idea behind ah resnet or residual network was that you had to take down the part of the input and then combine it with whatever is coming out of the convolution layer. Now as as an essence what the intermediate convolution layers over there are going to learn top is a difference between the expected between whatever goes into the input and what comes off the output and this is what whats also termed as a residual learning problem.

Now this is one part of it where you were just arithmetically adding down your inputs to your outputs. And then creating your resultant output from there. Now on the other side of it is whats called as a densenet residual connection. That they that was the people from c v p r 2017. And what happened out in a dense residual connection was ah more of in a way that [vocalized-noise] you get your inputs then you start ah concatenating your inputs to the subsequent layers and then. So what essentially happens is that you are not going to arithmetically add, but you preserve all the features by adding appending them on to ah the future channels which comes as the output and then subsequently it keeps on passing through the whole ah network over there.

Now ah the ah advantage which you gain in dense residual network over ah coming down from a resnet was that you could reduce down ah significantly the total number of ah weights. So the total number of parameters over there and one of the reasons why you could significantly reduce down the number of parameters was that you could use much lesser number of channels for doing this whole purpose over here. Now being able to reduce down the total number of channels over there is what plays a significant role in terms of bringing down the total model complexity while improving down the total accuracy of the model over there. Now having said that ah while we have learnt down on the theory part of it today let us look into ah dense residual connection network.

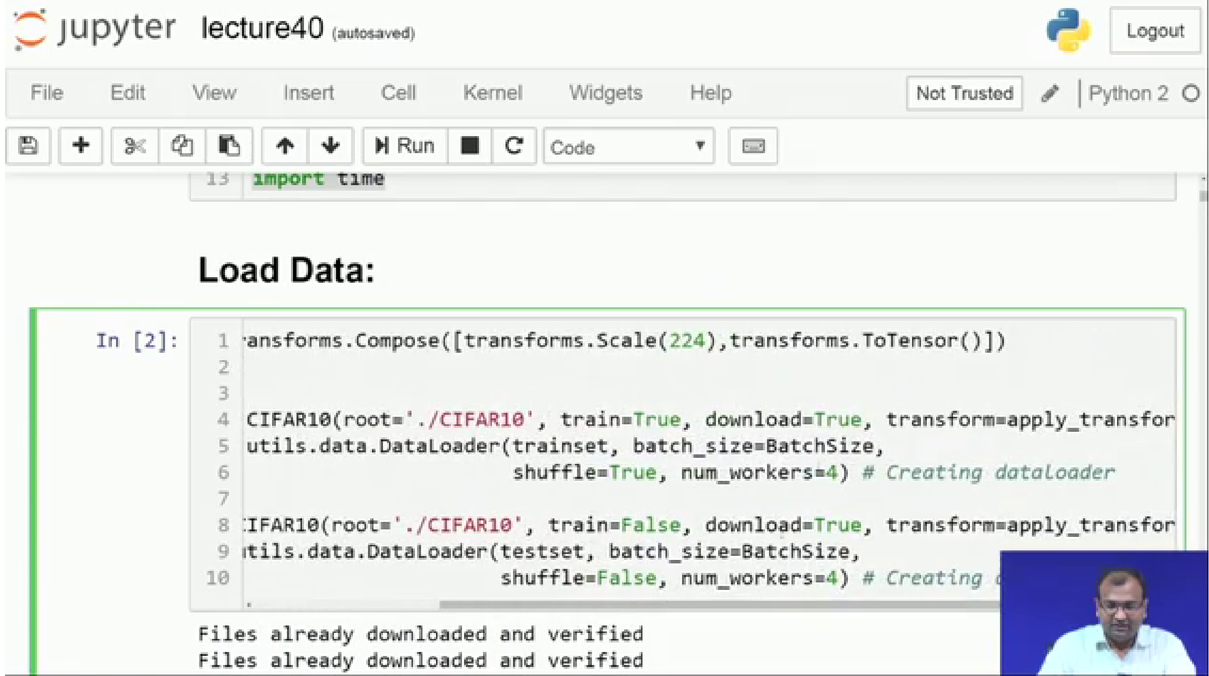
So we are going to do down at with densenet, now as going down with ah any of the conventional ones. Which we have done till, so we have our first part of the header which is quite straightforward. And simple as we keep on taking it up now once this part of the header is completely done.

(Refer Slide Time: 02:33)



The next is that we employ it on a c 5 ah problem.

(Refer Slide Time: 02:37)

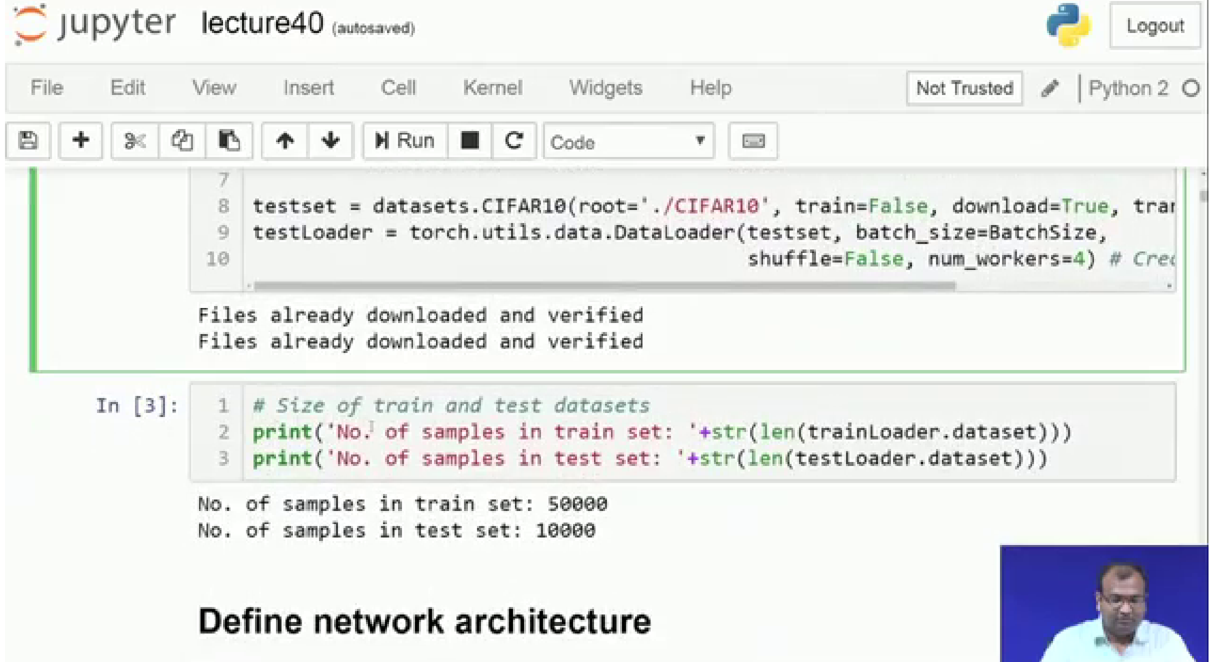


Now over here the data which we are going to take down is of the size of 2 to 4 plus 2 to 4 pixels and 3 channels over that. Now ah we are able to handle down 32 batch sizes. Now this is the really large batch sizes if you look down as comparison to a V G G network. Your batch size was much smaller now one of the reasons that you are able to handle down a larger batch size over. Here is because your total number of parameters actually a much lesser and in essence that. When you are reducing the total number of parameters over there the intermediate memory consumed internally over. There is also reduced down in some sense. Now [vocalized-noise] ah what we are doing is instead of ah looking down into image net kind of a problem which was a 1000 class classification problem, we are just making use of 10 class classifications over here.

So we have the C I FAR 10 which is a 10 class classification problem. We take the train and the test over here. And then ah we are just rescaling out of all the images over there

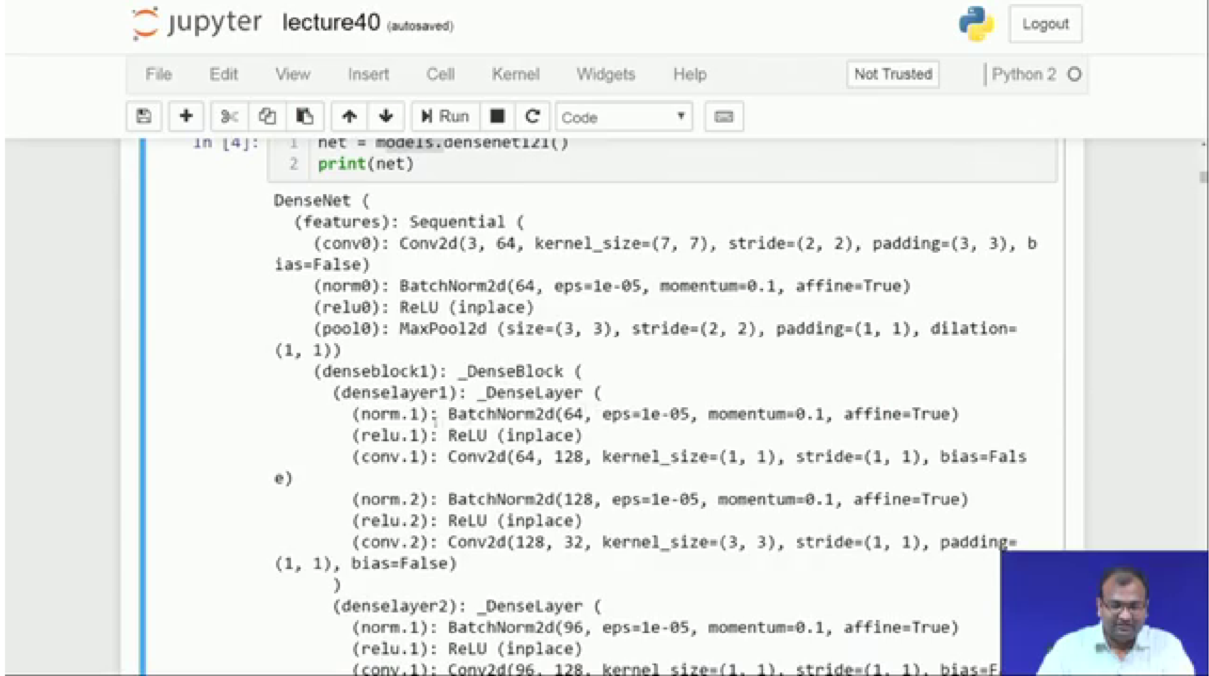
minus 2 2 4 plus 2 2 4. And then eventually we keep on doing it that for now ah having said that.

(Refer Slide Time: 03:40)



The next part is quite simple which is just to look down and on to the size of the images and we just have 50 1000 images for training and 10 1000 images for testing and thats how it simply goes on okay.

(Refer Slide Time: 03:51)



Now for ah the architecture part of it we use the same old concept from our earlier concept of using ah [vocalized-noise] data loader. Sorry using from our torchvision library our models. So these were all pre trained models or predefined models which were present over there and today. We are going to make use of densenet 100 and twenty one in order to solve it out. So I just do a pointer call from my models library over there and this is what is used for doing a pointer call on to and densenet 100 and 21 okay. Now let us look into the network over there this was a plain simple way of printing down the network as we had done in the earlier cases as well.

Now if you look down so you have your first part which is a two d convolution of 3 cross 3 kernels sizes and you have 64 kernels coming of it you. Sorry ah you have a 3 cross 3 kernel size over there. There are 3 channels 7 cross 7 kernels over here for convolution you have 3 channels, as your input channel 64 channels, as your output channelshich comes out and. Now what we also do is we try to do a stride and the stride over here is

the start of two okay. Now ah we do pad down with 3 0s on either sides over there and one of the major reasons of padding it down with 3 0s was such that your size of the image which goes your spatial span of the image which goes as the input and which comes out as an output is quite on the similar size and since it is a stride of 2.

So it is everything is divided by 2. So your input is a 2 2 4 cross 2 2 4 whereas, your output over there is 1 1 2 cross 1 1 2 which comes out okay. Now following that ah we have a batch normalization and then you have your relu and then you have a max pool.

Now this is what constitutes the first layer of convolution. Now in the subsequent one is when your dense layers actually start coming into play then your first dense layer over there.

(Refer Slide Time: 05:39)

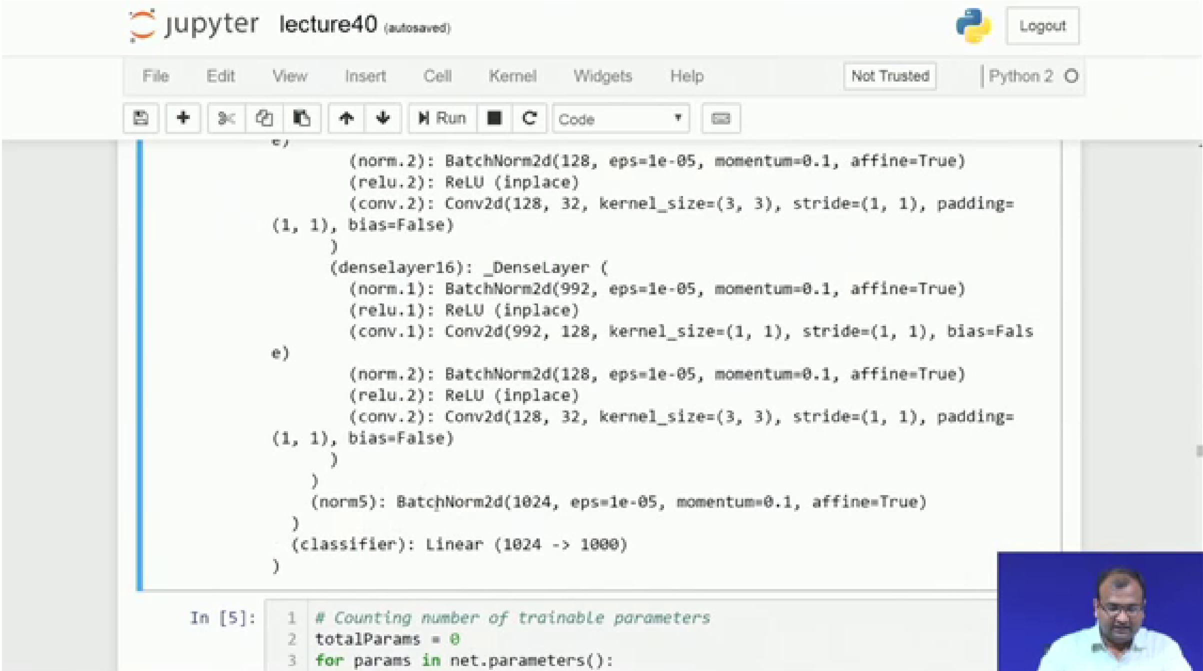
```
jupyter lecture40 (autosaved)
File Edit View Insert Cell Kernel Widgets Help    Not Trusted  Python 2

    (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
    (relu0): ReLU (inplace)
    (pool0): MaxPool2d (size=(3, 3), stride=(2, 2), padding=(1, 1), dilation=
(1, 1))
    (denseblock1): _DenseBlock (
      (denselayer1): _DenseLayer (
        (norm.1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
        (relu.1): ReLU (inplace)
        (conv.1): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
        (norm.2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
        (relu.2): ReLU (inplace)
        (conv.2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1), bias=False)
      )
      (denselayer2): _DenseLayer (
        (norm.1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True)
        (relu.1): ReLU (inplace)
        (conv.1): Conv2d(96, 128, kernel_size=(1, 1), stride=(1, 1), bias=Fals
e)
        (norm.2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
        (relu.2): ReLU (inplace)
        (conv.2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), paddin
(1, 1), bias=False)
```

Ah you have a 2 d convolution which has a size of 1 cross 1 and it takes in 64 channels. Which were the output from the 64 channels on the earlier layer. And then ah it generates an output of 100 and 28 channels over there ok. Now this does it with the stride of 1 comma 1. Which necessarily mandates at your size of the image does not reduce. So your input was of a special size of 1 1 2 cross 1 1 2. Your output is also 1 1 2 cross 1 1 2 ok. Now from there you get into this ah confa the next convolution layer which is the third over there. Now you put down 100 and 28 layers over here and you generate 32

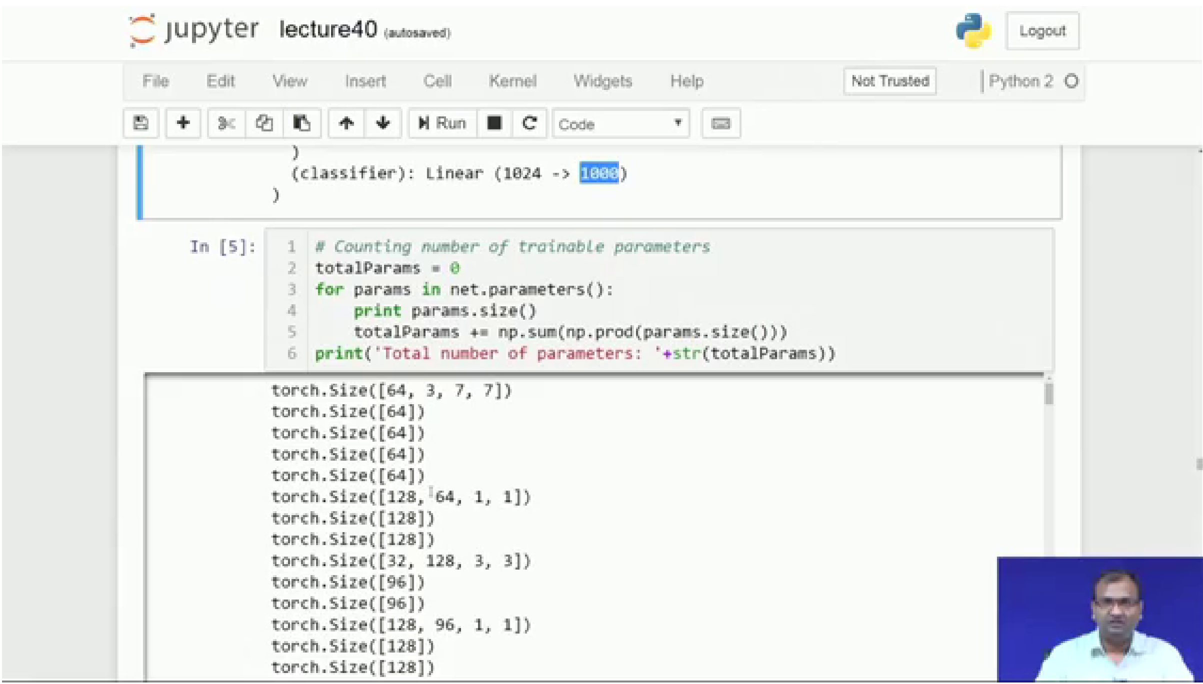
layers over there. Now the kernel size is over here are very different these are now 3 cross 3 kernels with the stride of one comma one, so that means that your total spatial span has in no way decrease the spatial span is still going to remain the same. Now this is what happens down with [vocalized-noise] one test block over here and then you enter the name next dense block and. So on and so forth. You keep on going down now within each ah dense block as it keeps on going that you keep on appending your layers subsequently from the previous layers to the next one ok. Now ah with that being done down or what you come down at the end is [noise] straight for yeah.

(Refer Slide Time: 06:58)



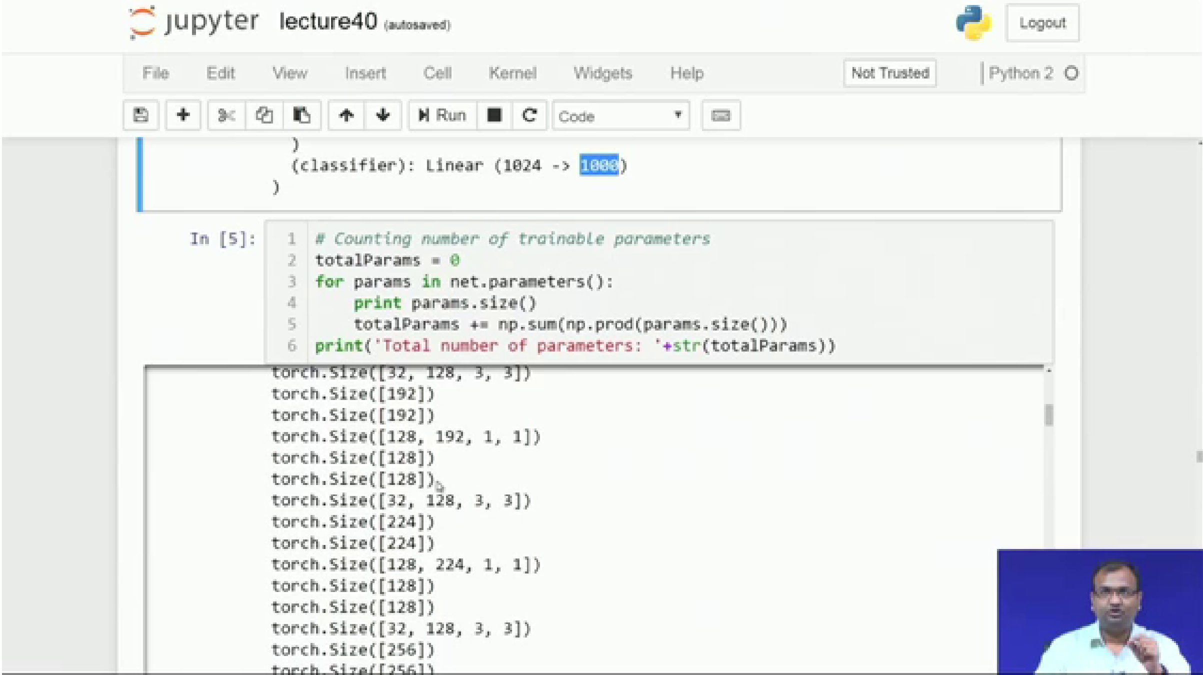
So at the end over there is finally when you get down 1024 ah neurons at that the final linearized output, and then as we have been doing with the earlier cases was connect down these 1024 1 2 1 1 1000 neurons for your classification notes over there and then this is your classification which comes out okay.

(Refer Slide Time: 07:17).



Now that goes out good now over here we just do a calculation on the total number of parameters. So this was again in the same way as we had repeated with earlier examples of sending it down.

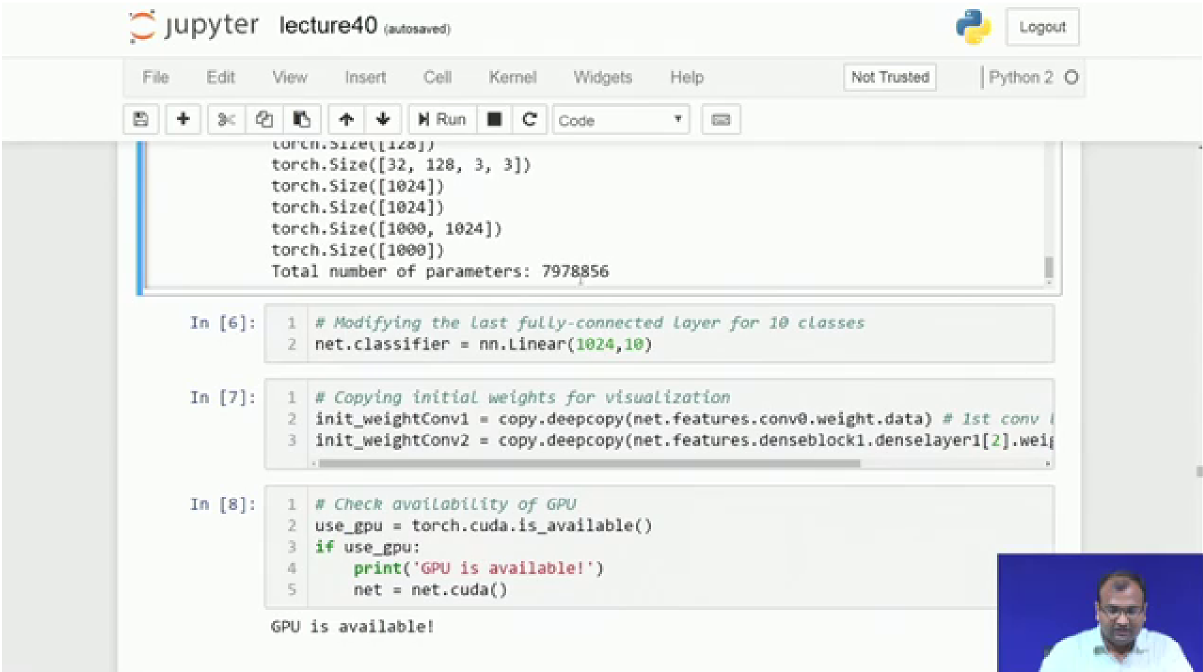
(Refer Slide Time: 07:29)



So we just looked down and the number of tuneable parameters or the kernel size. (Refer Time: 07:32) So where ever ah you have ah 2 d kernels or. So so wherever you have

convolution kernels present over that this is what what comes down and then these ah bits over here are something which is associated with the biases as well as ah with the ah extra additions which keep on go. So whether you are going to replicate all the models over there on the dense ah convolution part over there. So thats what keeps on going and if you take a sum total over all of these then you would be finding out that.

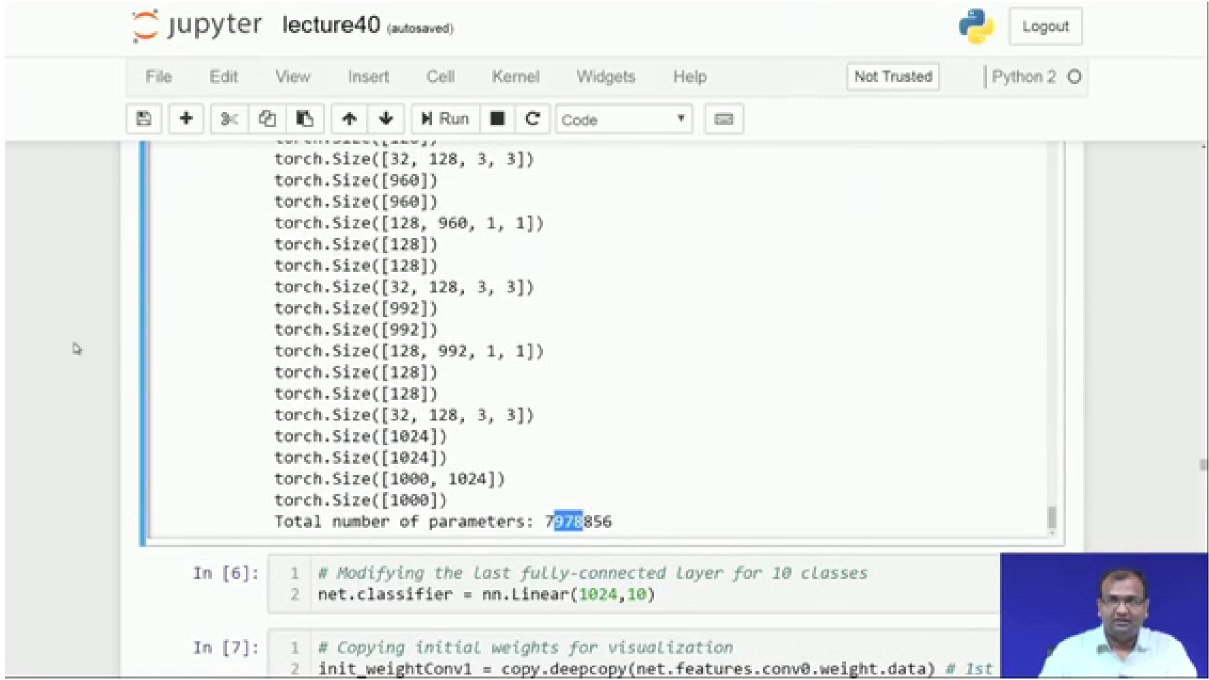
(Refer Slide Time: 08:03)



The total number of parameters over there is about ah 7.9 million. Or close to 8 million. So thats 7 ah so this is 7 million 978 1856 parameters in total which you have over here okay.

Now ah given the fact that

(Refer Slide Time: 08:22)



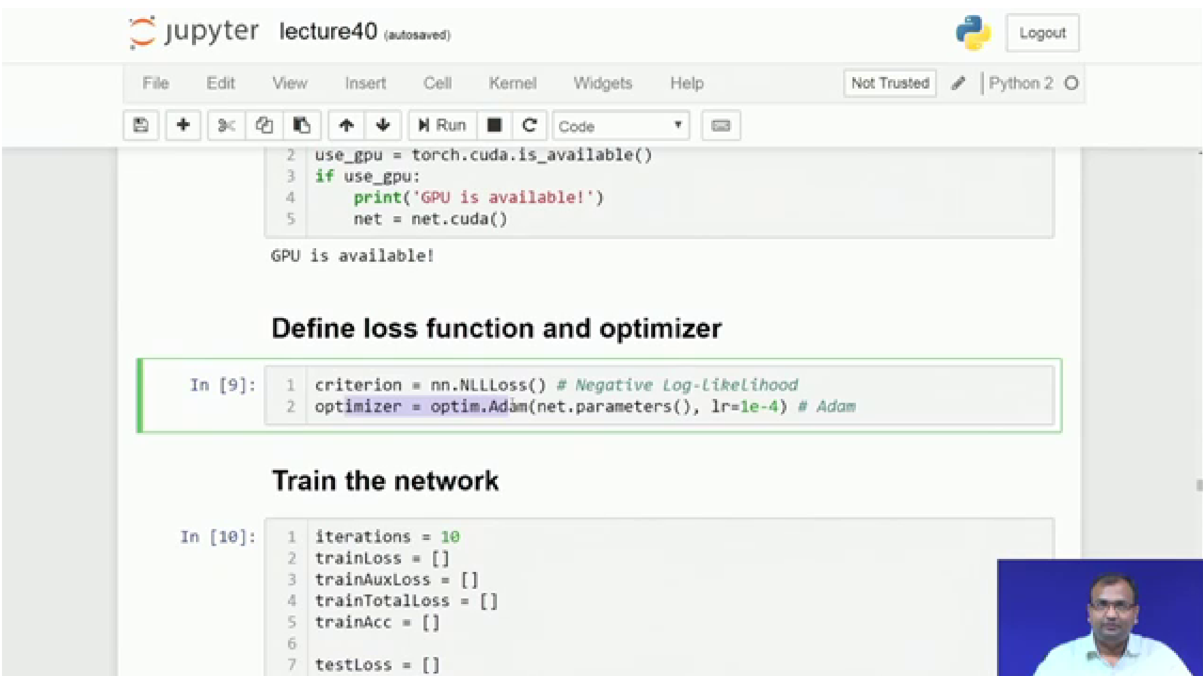
this was just 121 layer. So if you are kinds if you are trying to compare it with one of your residual connections over there. Then the total number of parameters which you will be finding down in your residual connections is something in the same ah is is almost twice as that of with a dense residual connection over here. Now if you just just compare ah full full residual connections with your dense residual connections over. Here then you would find out that you have almost half the number of parameters and thats what we had seen down in the earlier lecture slide as well.

So with just using half the total number of parameters as you would have otherwise had you can actually come down to the same accuracy level over here okay. Now comes down the modification which we need to do. So the whole point was that in the earlier case we were doing it with image net kind of a problem where you had to classify eight to 1000 classes. Now I cannot have a mechanism of classifying it to 1000 classes.

So we will have to bring it down because here it is just 10 classes to which it has to be classified. So let us ah just classify it down to 10 number of classes. And thats the next addition which we do and for that what we do is net dot classifier is the pointer which is present over there. So we just put down this ah rest of the pointer onto this one which connects down 1024 nodes onto just 10 nodes over there. So this dense and then closes our ah total network for the classification for a 10 class problem okay.

Now ah [vocalized-noise] doing that what we start by doing is now that we have our modified new network which and work out actually on c five ten. So we start by copying down the weights. Now once we copy down all the weights we just keep it over there.

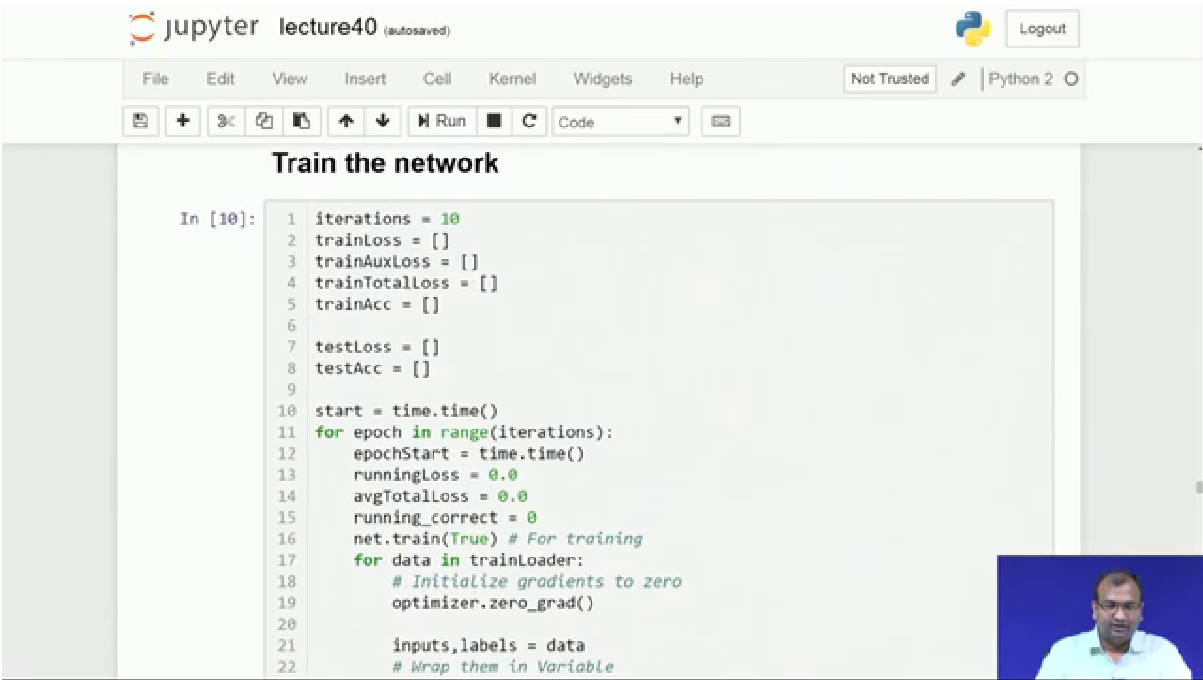
(Refer Slide Time: 09:58)



And then I just check out if G P U is available for my system to work on. Now if G P U is available then thats well and good and we can keep on proceeding over there. Now the next part which comes down is what is the ah actual ah loss function which we are doing and again falling down with what we had done in the earlier cases was if you have a classification problem. We just stick down to using negative log likelihood criteria, so we are still taking down the same loss function over here. So this is n l l ah loss function okay. Now for optimizer based on our previous experiences and what we had done in the earlier class as well, so we are just taking down to Adam which is one of the best optimizes to use over here. So we stick down to Adam and and we are just making use of Adam for our optimization purposes over here.

Now comes the training part of the network okay.

(Refer Slide Time: 10:45)



Now within this training part of the network what I am going to stick down is that I am just going to take take down 10 iterations over there no more not more than 10 iterations. And just run it simple now ah since the total number of parameters over here.

(Refer Slide Time: 10:59)

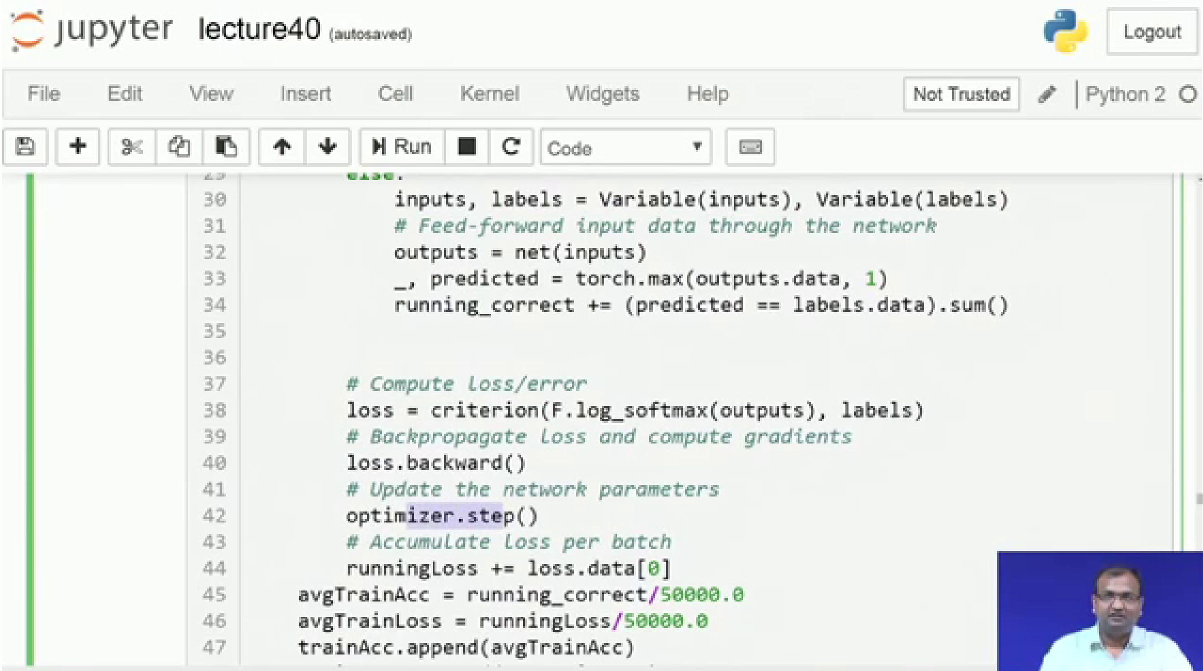
```
jupyter lecture40 (autosaved)
File Edit View Insert Cell Kernel Widgets Help        Not Trusted   Python 2

 7  testLoss = []
 8  testAcc = []
 9
10  start = time.time()
11  for epoch in range(iterations):
12      epochStart = time.time()
13      runningLoss = 0.0
14      avgTotalLoss = 0.0
15      running_correct = 0
16      net.train(True) # For training
17      for data in trainLoader:
18          # Initialize gradients to zero
19          optimizer.zero_grad()
20
21          inputs,labels = data
22          # Wrap them in Variable
23          if use_gpu:
24              inputs, labels = Variable(inputs.cuda()), Variable(labels.cuda()
25              # Feed-forward input data through the network
26              outputs = net(inputs)
27              _, predicted = torch.max(outputs.data, 1)
28              running_correct += (predicted.cpu() == labels.data.cpu()).s
29          else:
30              inputs, labels = Variable(inputs), Variable(labels)
31              # Feed-forward input data through the network
```

Is much lesser as well so that would assist me in a big way and because of lesser number of parameters my total epochs will be taking lesser amount of time well that. Now what

we need to get down into it now the first part of it is which looks into poor epoch rebased losses is is quite the same the next is to get down into our patches. And thats where we make use of the train loader over here ah to do it. Now what I do is actually to find out if G P U is available then I just convert down my available inputs and my network onto a G P U if G P U is not available then ah I am just not going to convert that over there. Now once that part is done.

(Refer Slide Time: 11:38)



The next is to find out at my loss. So this loss is what is computed down using my criterion function for batch.

Now once that is computed and I have my loss available. So I do a back propagation and the gradient computation is what is done with lost or backward over there. And then I just ran down the optimizer step over there .Now the ah point of running down this step is just to run down the optimizer and within each patch we are going to do. Now what you need to keep in mind is that this network is quite unlike googlenet where the issue was to get down auxilary losses husband, but here since ah we do not have any of these auxiliary classification arms coming down. So we do not have an issue of auxiliary losses computed. So it is its just the loss which is computed at the end of it now the other fact which was like if there is a loss which is computed at the terminal part of it then

does that loss somewhat propagate back and get down to the earlier part of it yes thats also pretty much what it does because you have your dense residual connection.

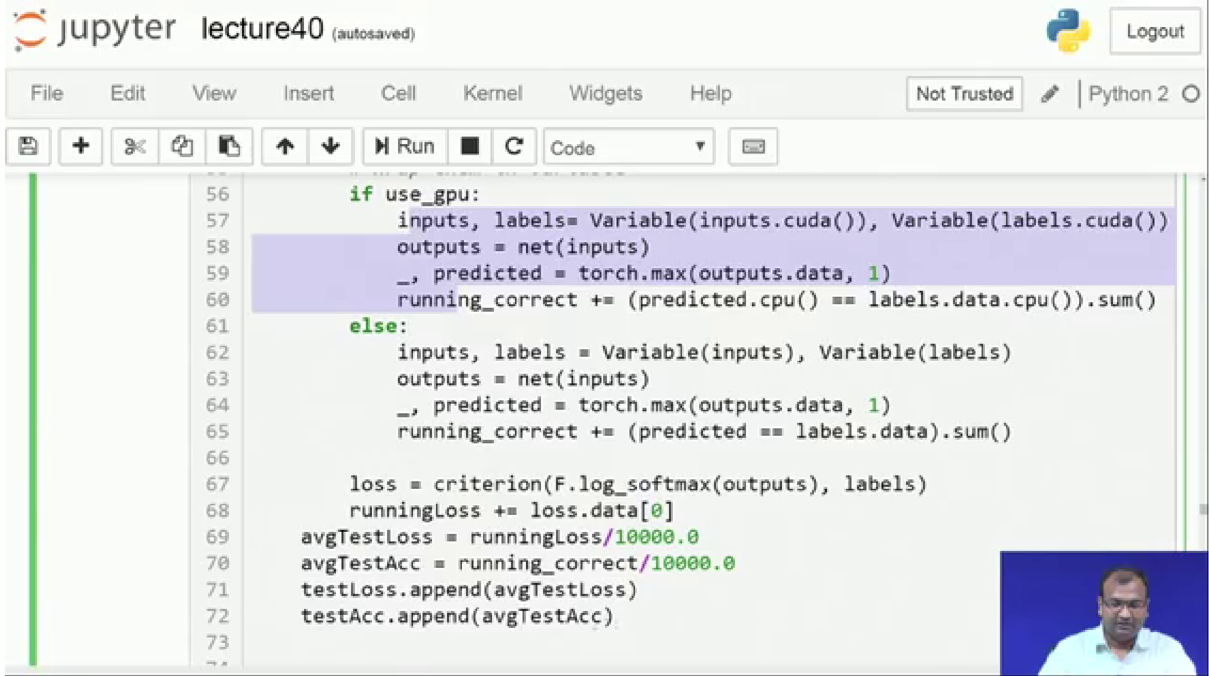
So there is always one part of the tamp which comes down from the first part of the network. And goes to the complete end and as well as taps into all of these middle. So what happens during a back propagation is that each of these parts themself back propagate all the gradients back to the start of the network. And thats how the whole weights and everything are updated over there okay.

(Refer Slide Time: 12:59)

```
44            runningLoss += loss.data[0]
45        avgTrainAcc = running_correct/50000.0
46        avgTrainLoss = runningLoss/50000.0
47        trainAcc.append(avgTrainAcc)
48        trainLoss.append(avgTrainLoss)
49
50        # Evaluating performance on test set for each epoch
51        net.train(False) # For testing [Affects batch-norm and dropout layers (i
52        running_correct = 0
53        for data in testLoader:
54            inputs,labels = data
55            # Wrap them in Variable
56            if use_gpu:
57                inputs, labels= Variable(inputs.cuda()), Variable(labels.cuda())
58                outputs = net(inputs)
59                _, predicted = torch.max(outputs.data, 1)
60                running_correct += (predicted.cpu() == labels.data
61            else:
```

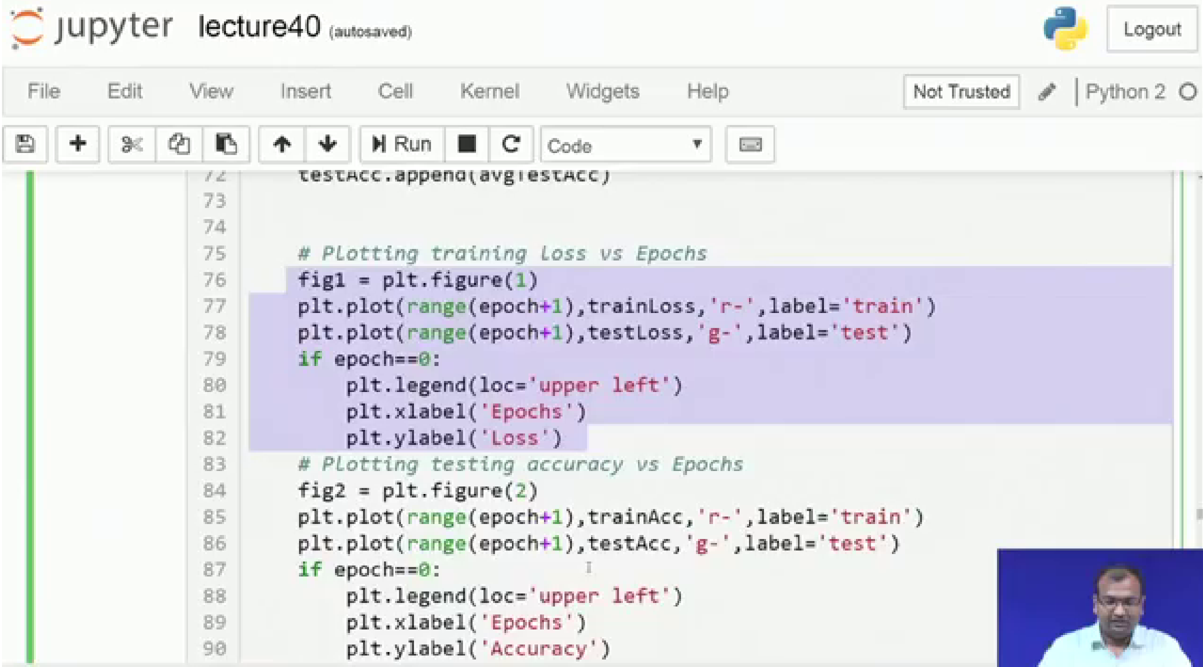
Now from there fee come down to our running loss per epoch and then you just sum it up and get down an average value of it. So this was plain straight as we had done in the earlier examples as well and there was not anything new to do over here. Now the next part is to look into our testing data. Now in the test data also I just check out if G P U is app level then I just work it out on the G P U over there otherwise I just leave it out on the c p u.

(Refer Slide Time: 13:24)



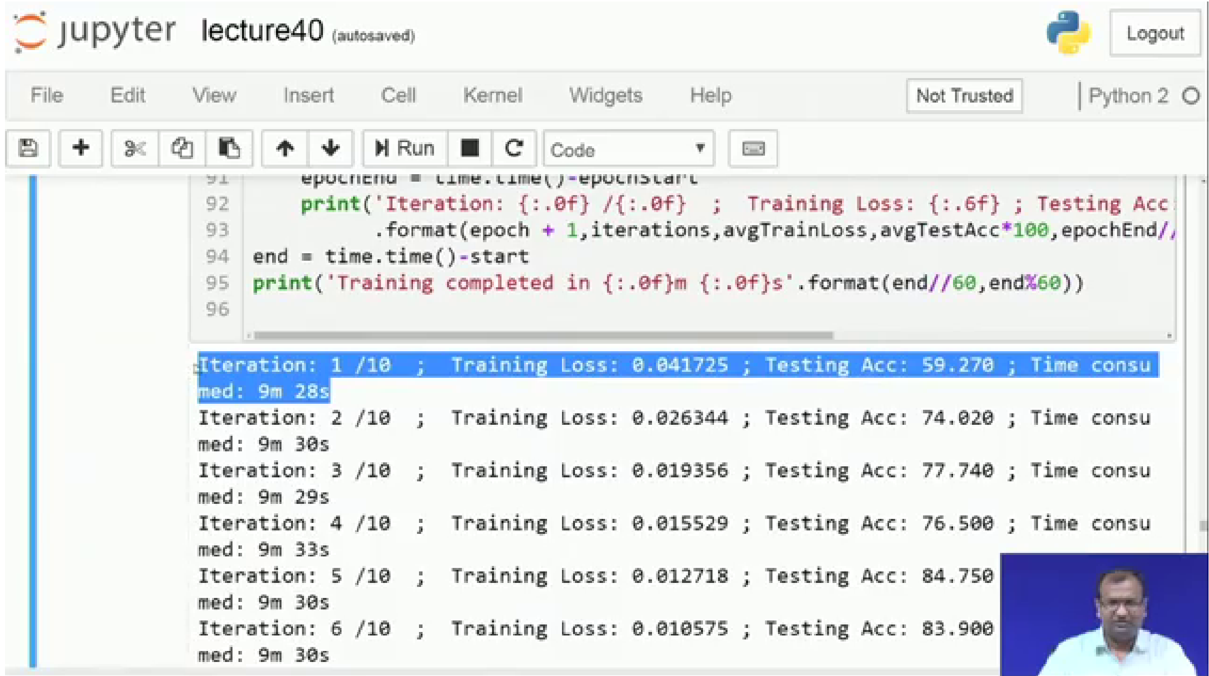
And then find out my loss and then I get done whatever my loss comes down okay.

(Refer Slide Time: 13:31)



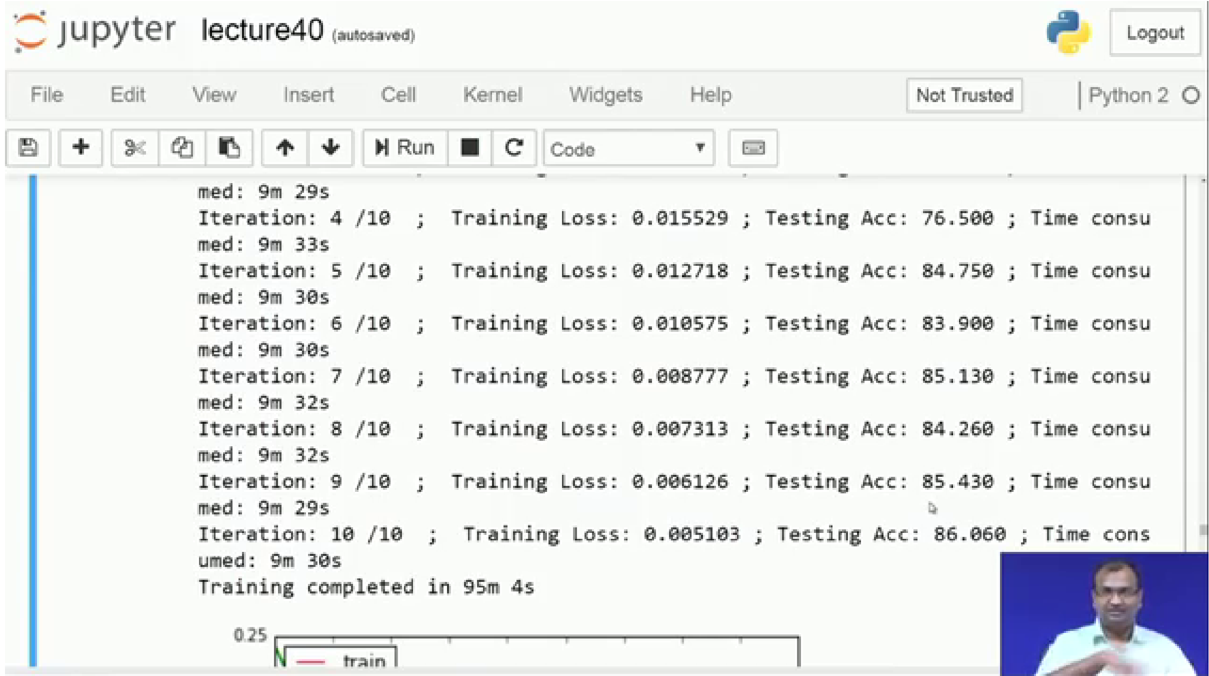
Now with that final ah we come to a plot over here and this part of the code is again principal as we had done in the earlier cases as well.

(Refer Slide Time: 13:39)



So let us look into what it takes down now we have just shifted over to a different machine and a different G P U which is a bit slower. So it takes slightly bit otherwise 9 minutes is over quoted. If we were doing it down on the earlier one then it would not have taken. So much of time

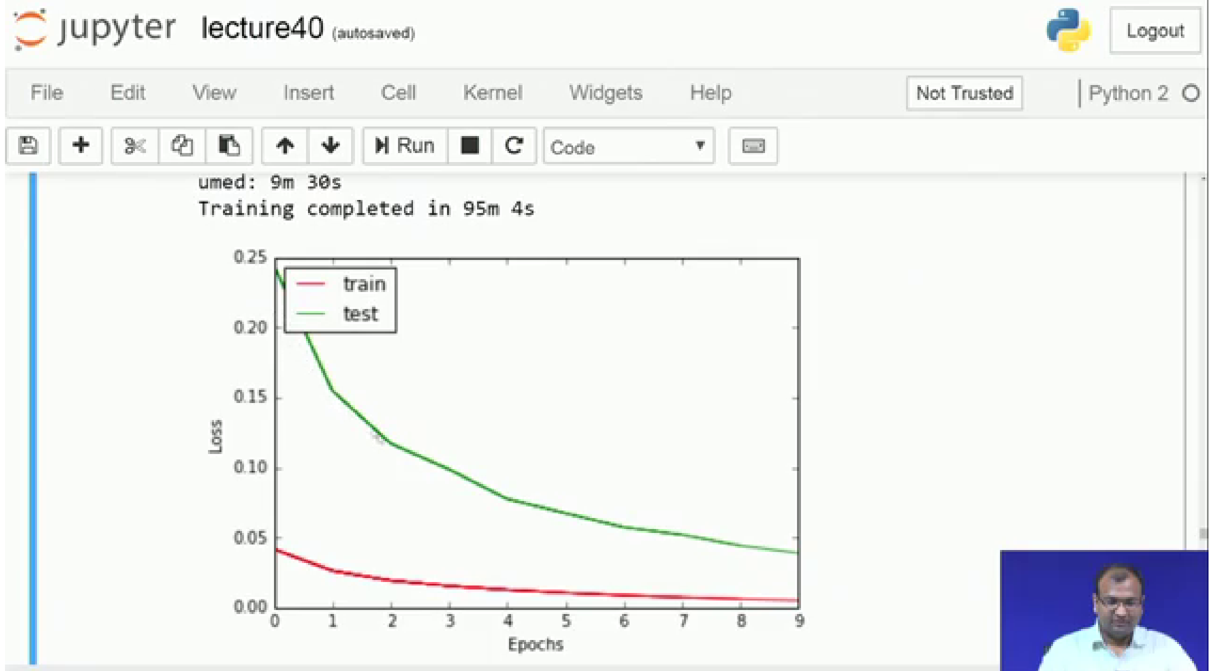
(Refer Slide Time: 13:54)



Now so thats thats something which is very much hardware dependent as I had said in the earlier classes as well. Now if you look down into your accuracy so you are ah starting down with an accuracy or something like 59 percent which is quite simple like. And most of these cases it was randomly initialized and your accuracy is is technically around 50 percent is what we have seen with any of these very deep networks over there and what happens is that it it closely escalates.

So if you look down so within the first epoch itself it goes out from 59 percent almost 60 percent to a 74 percent a 14 percent jump. Then it does a 3 percent jump then there is a minus one percent change. So this is where it starts to stagger down a bit and then does it. So one of the major reasons why this happens is that it starts it is local oscillations over there. And then since it is not able to get down the exact minimum position. So it is just oscillating around and takes a mole bit of my time to come down to a convergence.

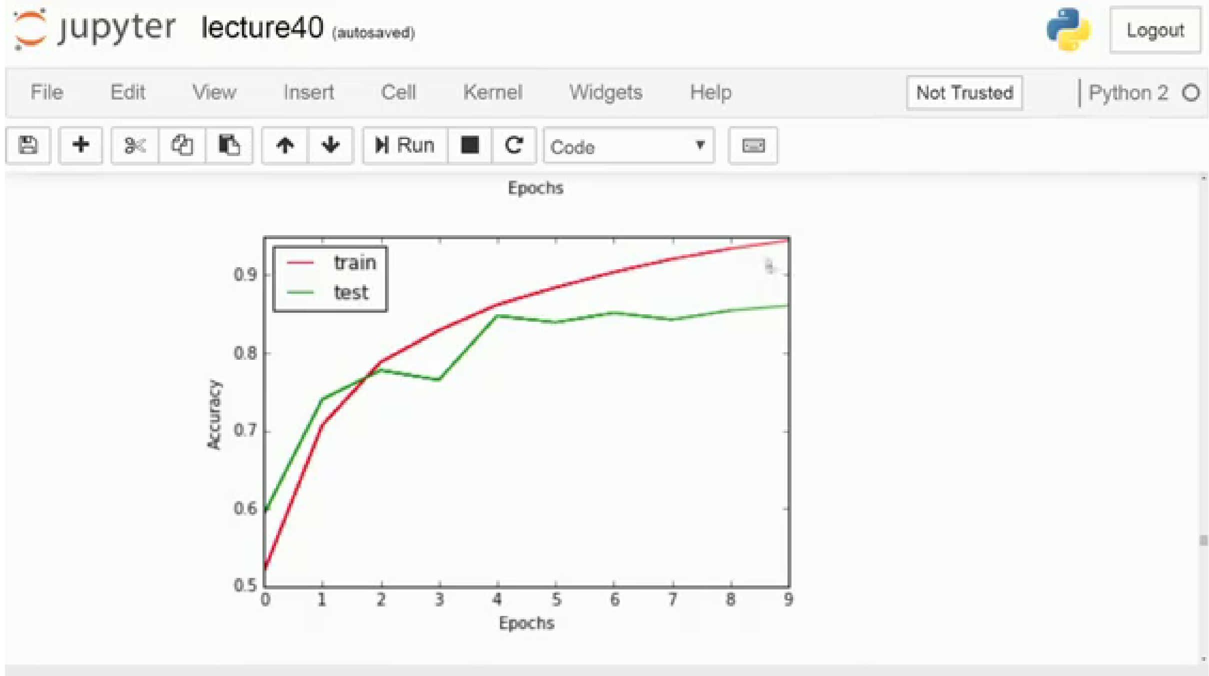
(Refer Slide Time: 14:53)



So this is what comes down with my so my red part is with my training losses over there my green is with my testing. Now there is something ah interesting which which happens over here as well now while my training losses are much lower my testing loss is; obviously, higher and this is pretty much known, but the good thing is that this test curve over there is something which is coming down much faster at a rate.

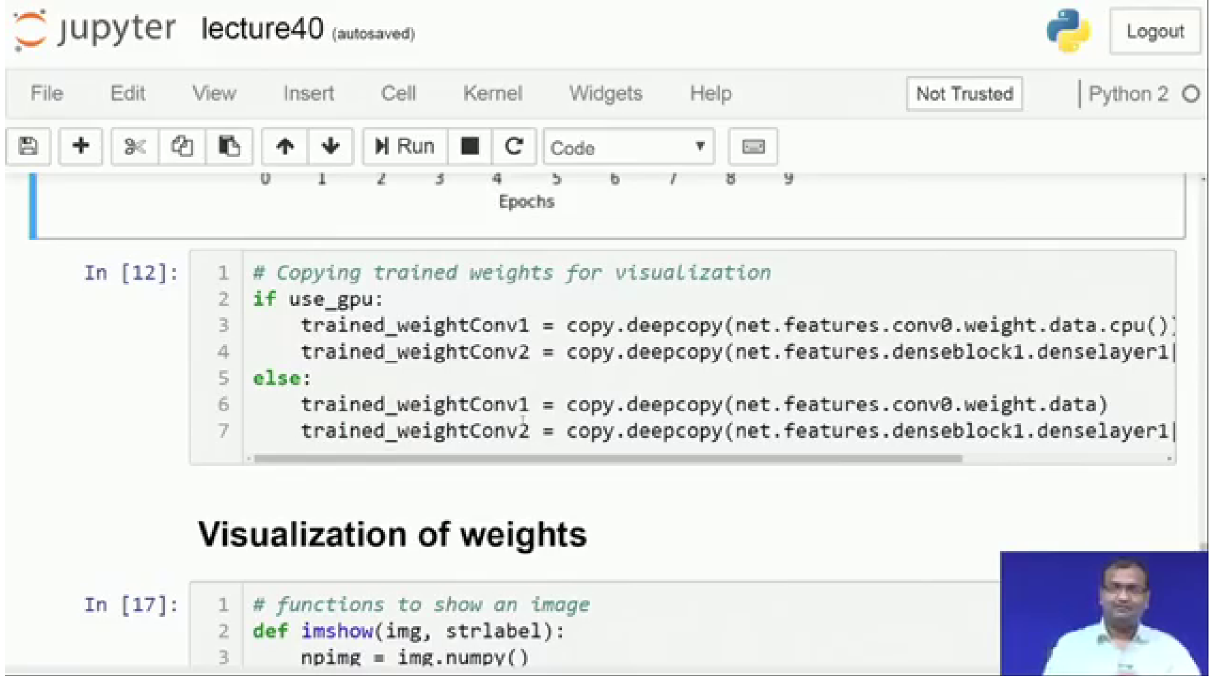
So it means that the generalizability of the model is going up in a better way

(Refer Slide Time: 15:23)



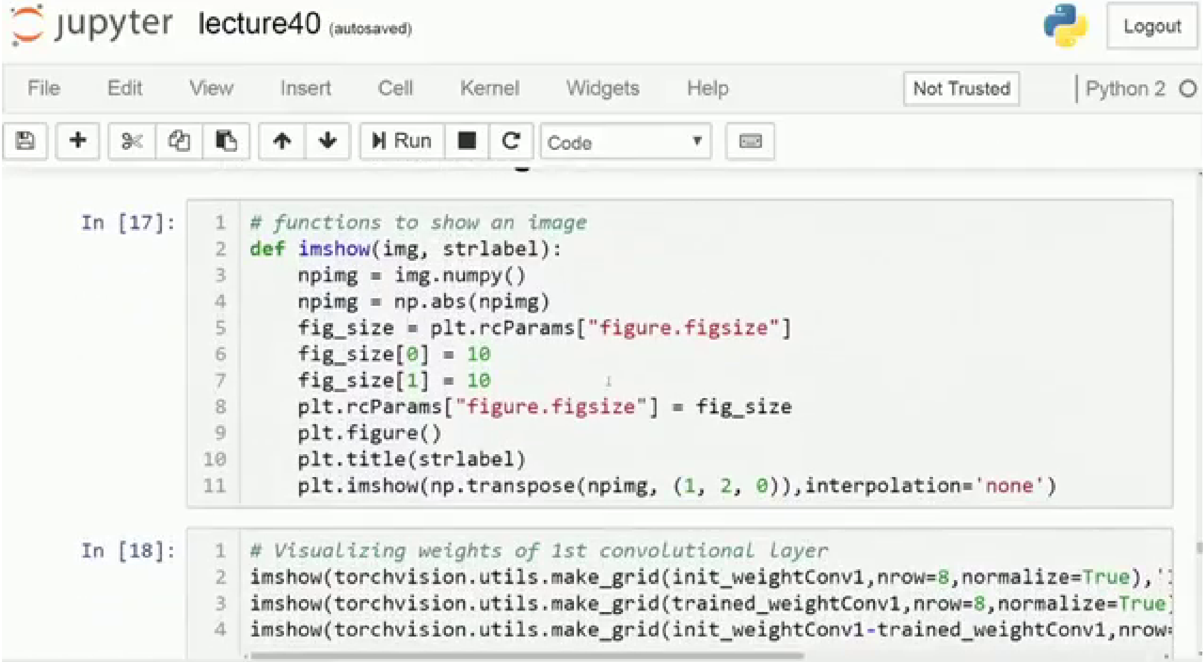
now having said that we try to look into our accuracy curves over here. Now while the training accuracy is really going high and a lot of you might tend to think that it is its now going close to over fitting, but then it the testing accuracy is not something which is going up to that ah level. So now you need to keep on running it beyond 10 epochs because to any proxy is just a very small amount of time in order to on continuing this one.

(Refer Slide Time: 15:47)



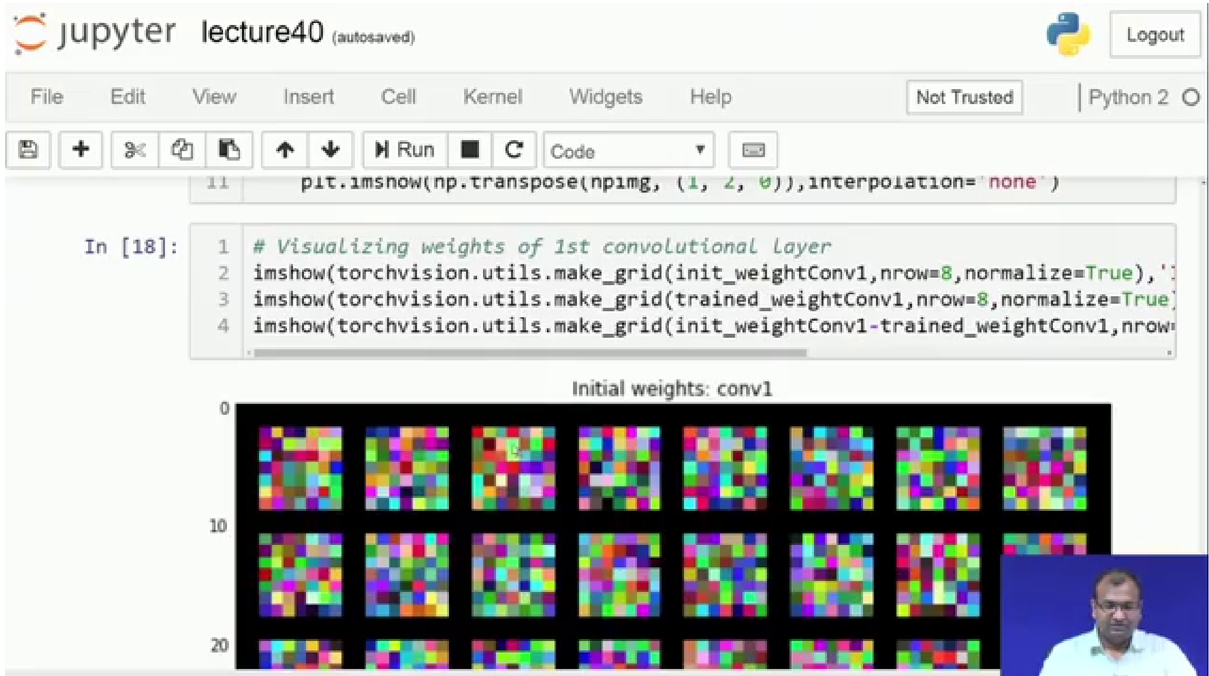
Now now that my training is completely over the next part which we need to do is that we need to find out what has been the nature of the weights and whether there has been a change of the weights anything no whatever we had done random initialization versus what comes out at the end of the training process over here.

(Refer Slide Time: 16:03)



So that we go down ah

(Refer Slide Time: 16:06)

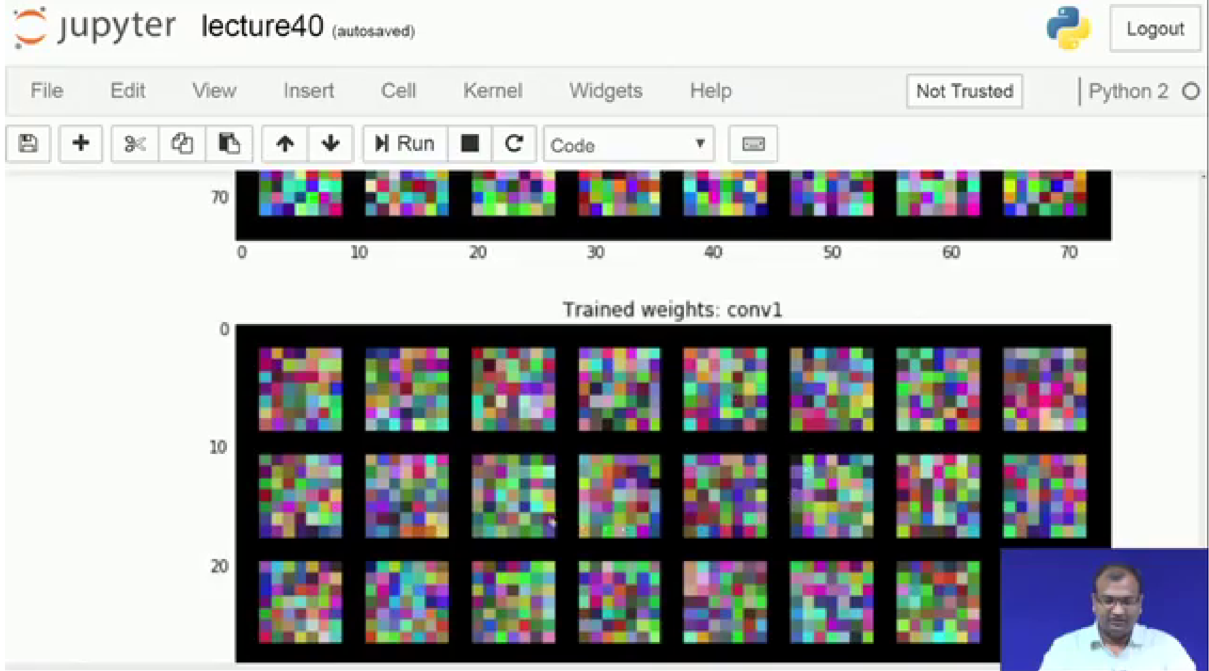


And get into my weight initialization and weight visualization practice. So these were the weights over here now if you remember the first. So let us get back onto the model over here once more.

So our first layer for the module was something which had corner sizes of 7 cross 7 and there were 64 of them. And these were all taking in so this is my first convolution layer which took down 3 channel inputs over there 3 channels, which would correspond it to an r g b and it produced out 64 channels. So there were 64 unique corners present over there and each of the kernel was of a size 7 cross 7 okay now what came out of from here is that we have eight cross eight matrix of 7 cross 7 corners each over there and they are all r g b in nature okay. So let us get down to visualizing these corners over here. So these are the nature of these corners which get to learn down over there.

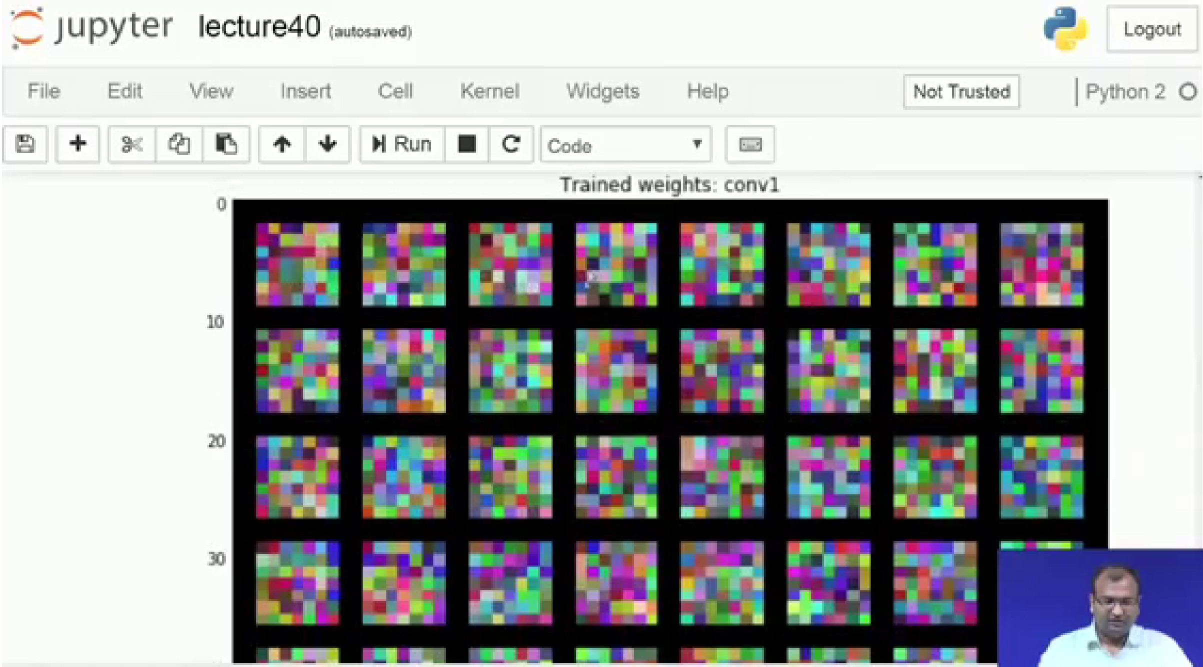
Now initially ah this is the one which is initially before training. So they are all random numbers which are present over there.

(Refer Slide Time: 17:07)



This is what happens after training.

(Refer Slide Time: 17:10)



And you would see that the intensity of a lot of these colours have changed whereas, the shade of the colour does not change. So that practically means that the difference between the r g b over there is not. So the proportionate ratio for r g and b channels are not changing much, but the intensity or or the nature of the weights are changing.

(Refer Slide Time: 17:27)

So let us get into the weight difference over here you see a lot of these random patches ah coming up over here. So there has been significant amount of change around that, but then the proportion in which.

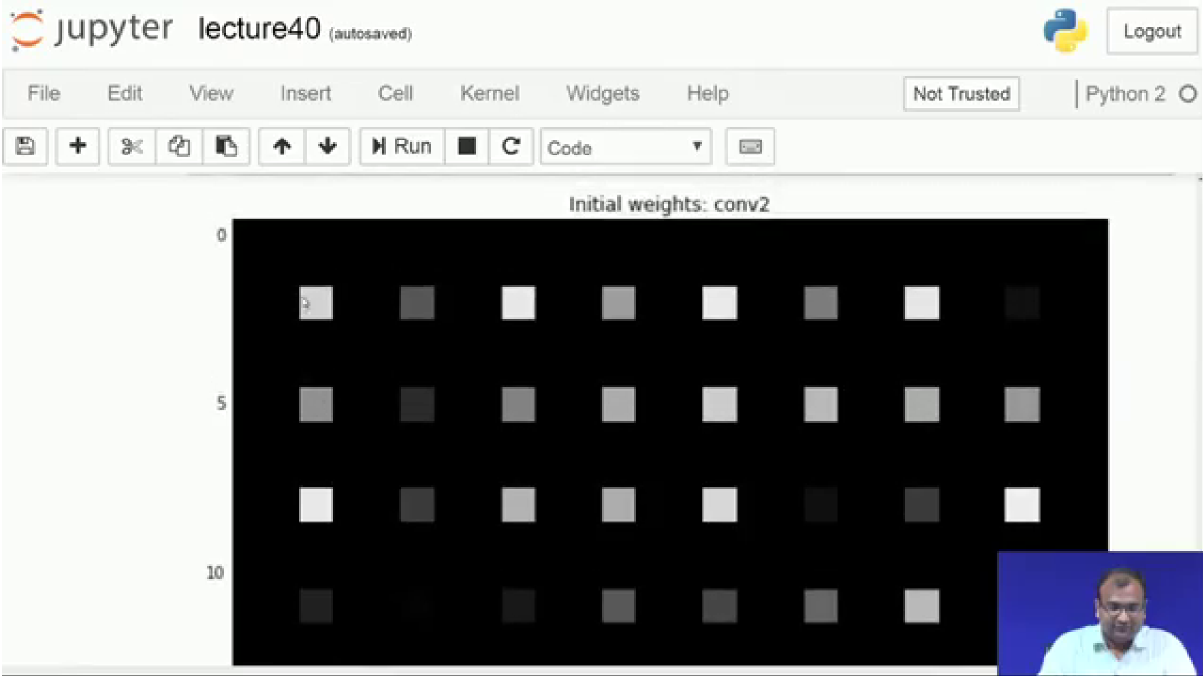
So that thats the reason why the shades are not changing and whats really interesting is that you do not fail do you you do not see actually any distinct shape of objects or distinct corners coming up. Now these are very complex corners and and they do not have an immediate visual representation. If you are looking down at shallower networks over there with the shallower network it was much easier to learn down kernels. Which are very represent of the image with the more deeper you keep on going you might not always end up getting down contests which are very descriptive of how the image ah properties are and then thats thats thats the nature of these kind of very deep learning systems as they are okay.

(Refer Slide Time: 18:15)

Now the next part over there is if you go back and look into the second convolution layer the second convolution layer is what takes in yeah. So this is my second convolution layer which takes in 64 channel input .Which I have over here and then I produce 100 and 28 channels and the kernels are a size one cross one so.

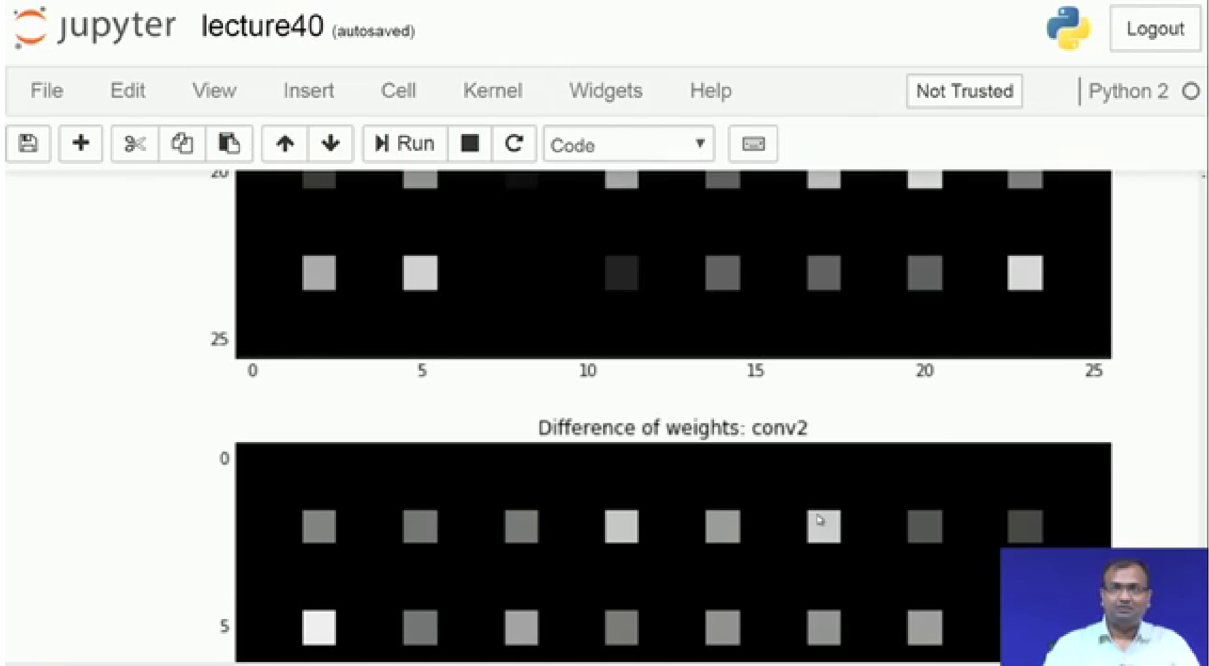
Now what I have over here is just one single matrix value represented. So how we have represented these is that I just take out the first channel out of this 128 channels and then I display it is response over here. So there are ah total 64 number of kernels. So thats what you see.

(Refer Slide Time: 18:58)



So each patch over there is just 1 unit kernel 1 1 1 channel of that 1 cross 1 kernel and you have 64 channels. So there are 64 six patches over here so one two 3 4 5 6 7 8 and you have 8 over here in total. So that makes it complete. So these were the weights which were at the initial part over there and these are the weights which are there after training and then you have the difference of the weights coming up over here. Now ah this this difference of the weights is something not not much to be intrigued about and then they keep on ah changing.

(Refer Slide Time: 19:32)



Now we have not yet come down to a total convergence. so thats thats one of the main reasons why these weights are still changing over then and you do not see a flattened out feature now as you can keep on training them for more number of epochs.

So we are still at the point of just 10 epochs and this has not yet come down to a saturation. So typically at your disposal you can definitely chain it down for longer number of epochs for 100s of epochs at rata stretch and then keep on doing it out. So ah thats something I would leave it down to your side of it now having said that we have discussed all of this, but in the subsequent lecture we are going to focus more on getting the actual compute complexity and and for this one. So what will be the total number of bytes taken down to actually save the model ah when it loads out on the Ram. How much of bytes would it require what will be the total number of floating point computation, which will be required over here as well as and and this floating point computation is something ah which is going to ah help you decide as to given a particular hardware.

Which has a certain amount of floating point computations delivery rate then what will be the total time, you would be taking in order to get this delivered out of the system on top of that. There is also another important point which we will be discussing in the subsequent lecture. And that is one the total operational space complexity. Now for the operation of space complexity ah major factors that ah. If we have larger size kernels and if there is an intermediate intermittent activation response which also keeps on

generating, now if this is large then we will take more amount of space and as a result my batch sizes will have to be reduced.

Now if I do not want to go around with that then I will have to figure a way of solving it out. So this is what we will be discussing in the next class and thats thats where you will get to know that y in google ended. We had smaller batch sizes of 16 whereas over here, we can have larger batch size 32 64 and and. So on and so forth. So there is an integrate dependency, but then it is not just dependent on what is the total number of convolution kernels or the number of channels present over there it is dependent on also dependent on what will be the total number of ah activation maps which are which you are going to produce out. So this is what we keep on tuned for the next lectures. Now this is the point where we are already done and through with our basic understanding of deep network.

So in the subsequent weeks we will be understanding more about how what is a complete. Complexity problem within these deep networks what are my transfer learning issues within these deep networks. And how do we how can we have efficient transfer learning between them. And then subsequently we will get down ah to even interesting problems including can we solve out regression problems or convolutional classification called a semantic segmentation problems ah with with these kind of networks. And these what we have learnt till now are what will be forming down my foundational basics for whatever I am going to do in the subsequent ones. So thats all what I have to say for today ah and then at the end of this week. So just stay tuned for the ah subsequent lectures in the next week

Thanks.