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

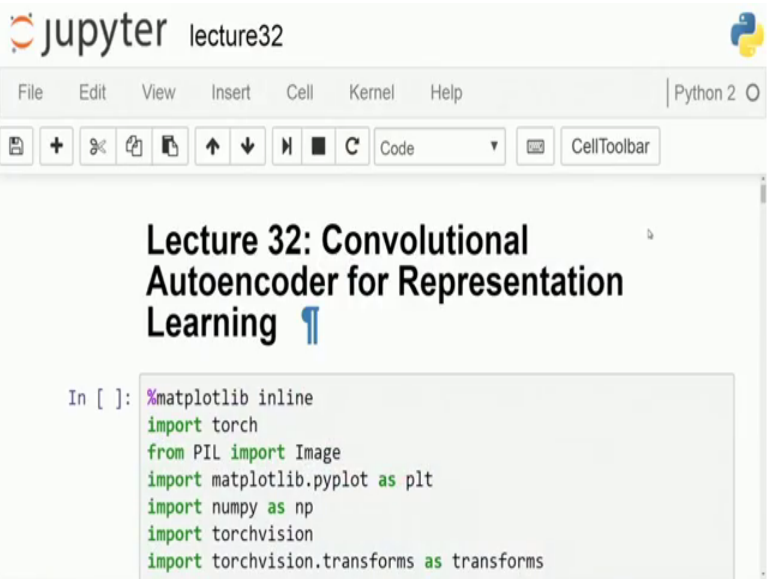
Lecture- 32
Convolutional Autoencoder for Representation Learning

Welcome. So, today we are going to get you introduced to work Convolutional Autoencoder and, as you have done in the early lectures on just connecting now fully connected auto encoder where, you had a bunch of neuron on which, you had your input image as in a batch or may be the whole image which goes through it..

And then it connect down via all of this hidden layers and, the end terminal over there was going to reconstruct your original image and your cost functions were defined in a way, which is trying to minimize the mean square error between whatever is reconstructed and what was focus given down on the input side of it. And the whole purpose of this auto encoder was to lean representations, which can encode them image in it is own, (Refer Time: 00:53) best possible way..

And that would have resulted in nature of features and a set of features which are hierarchically connected across the depth of the network which can be used for classification purposes and this definitely give you edge over all the other methods in which you can use an unsupervised training mechanism, in order to actually learn down features and, then all of the features which are learn down in an unsupervised manner can be used for initializing a fully connected network, in order to do your image classification as an end to end pipeline..

(Refer Slide Time: 01:30)



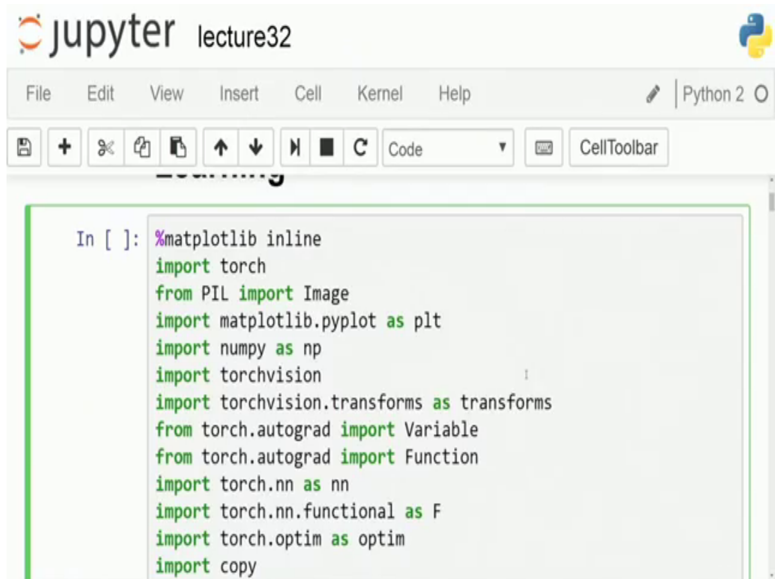
Now while we had done all of these ones in the first month and, then subsequently we went over to understanding convolutional networks. So, in convolutional networks one thing which we were doing is you needed a large corpus of a data, which is already annotated such that over here, while you are back propagating your errors caused by misclassification, your new features are getting leant and this features. Now which are getting leant are in terms of convolution kernel which appear across the hierarchy.

Now, the question which does face to your mind at that point of time, is can we not have a mechanism for actually learning these features in an unsupervised manner as well. And, that would be of a huge application as well as an immense contribution to a field where, you have less amount of annotated data for classification learning while you can have an ample amount of an annotated data. So, which is not labeled, but you have just a bunch of images over there for all the objects which can ever occur. And convolutional auto encoders are basically an answer to this kind of a mechanism..

So, where we go down is what you had seen in the earlier lecture was where we introduced you to a convolutional auto encoder, in the form of how convolutional connections are made down and, then it directly connects down to a fully connected layer in terms of auto encoder and, then you have the rest part of it which is your decoder block and that will be making use of either up sampling operator, or an unpooling operator or something even called as deconvolution operator as we had introduced..

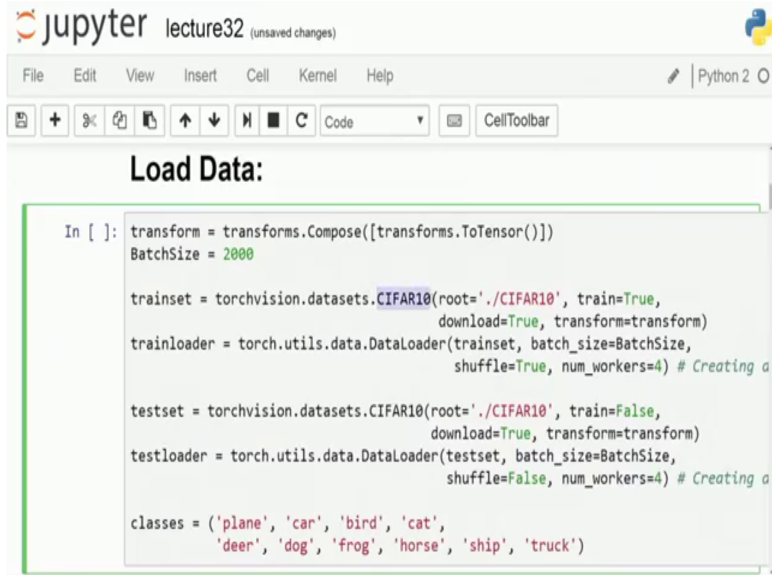
So, today I am going to write down one very basic sample code in order to do all of those parts, we will have a convolutional part of the encoder, we will have a fully connected part of the encoder, then similarly goes down as a fully connected part of the decoder and convolutional part of the decoder. So, without much wait let us get into what this is doing.

(Refer Slide Time: 03:19)



So, as in with all of our earlier parts we just end up loading down necessary libraries which come down over here. So, let us do a simple run over this libraries; so, once your libraries are loaded, then you have your data which is coming down..

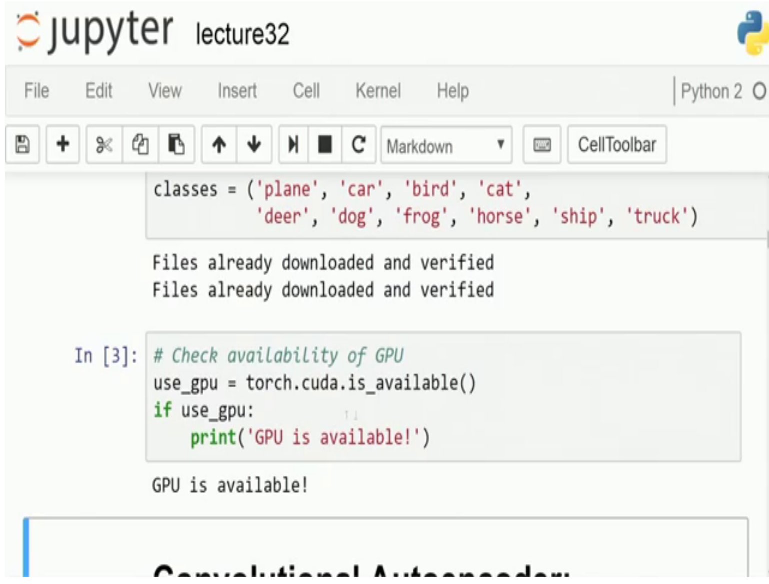
(Refer Slide Time: 03:32)



Now, for the data which we are using I chose to use down just CIFAR over here the 10 class classification problem data which we have used in the earlier lecturer while trying to do it with LeNet as well. So, there we had modified LeNet in order to take in color images and all the color images which were there were from CIFAR.

And so, quite contrary here what we are going to use is although all of this data does have its training labor in terms of classification associated with it, but we are not going to make use of that, we are just going to make use of the image data itself. So, it the labels over here as such are of no use ; it is just the image which is going to be used over here. Now, for batch sizes we use a batch size of larger size over here, it is 2000 images which goes into one single batch. .

(Refer Slide Time: 04:22)



So, let us just get this part running and once that does we check in as usual for our GPU availability. So, my GPU is available and that is great.

(Refer Slide Time: 04:30)

jupyter lecture32 (autosaved)

File Edit View Insert Cell Kernel Help | Python 2

Code | CellToolbar

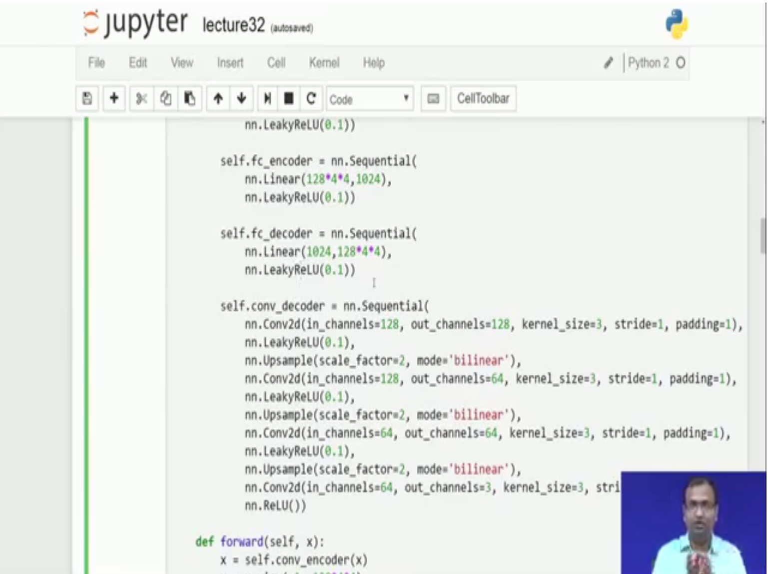
Convolutional Autoencoder:

In [ ]:
```
class autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()
        self.conv_encoder = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, stride=2, padding=1),
            nn.LeakyReLU(0.1),
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=2, padding=1),
            nn.LeakyReLU(0.1),
            nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, stride=2, padding=1),
            nn.LeakyReLU(0.1))

        self.fc_encoder = nn.Sequential(
            nn.Linear(128*4*4,1024),
            nn.LeakyReLU(0.1))

        self.fc_decoder = nn.Sequential(
            nn.Linear(1024,128*4*4),
            nn.LeakyReLU(0.1))

        self.conv_decoder = nn.Sequential(
            nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, stride=1, padding=1),
            nn.LeakyReLU(0.1),
            nn.Upsample(scale_factor=2, mode='bilinear'),
```

So, now I am I am done with my initial part of it in terms of just taking down my data and getting my stuff initialized. Now, the next part is I will have to define my convolutional autoencoder architecture over here, if you go through this model over here, what it tries to do is I have 4 different blocks, in which the whole thing is divided; one is

called as a convolutional encoder, the next part is a fully connected part of the encoder which succeeds the convolutional operations within an encoding.

(Refer Slide Time: 05:00).



Following that I have a fully connected part of the decoder and, then I have a convolutional part of the decoder as well. So, let us go down how it is define now convolutional part of the encoder, it is basically the first part is 2 d convolution, now what it takes in is 3 channel input because, you have your rgb image which is coming down on the input side over there. And then it maps down to 64 output channels..

So, the total number of conversation which you will be having in this first layer is 64, that is a total number of unique channels which inspired to maps down, to . Each of the kernels which we used as a 3 cross 3 kernel not 5 cross 5 any of them, we do a convolutional which is a stride convolutional so; that means, that necessarily the size of the result over here is going to be smaller than the size of what you had, if you are just doing with a normal side of 1 ok. So, this is something similar to as whatever size you would have achieved, if you had a stride of 1 convolutional succeeded by another max pooling with 2 cross 2 and stride of 2. So, we just merge all of them together though functionally, it is different as we had learnt in the earlier lecture as well. So, functionally that are going to be certain differences coming down, but this is just a faster way of computing it out and reducing the total number of compute layers over there. .

And finally, there is a padding available. So, let us just and the padding over here is just the padding of 1 which is present. So, that is like the 0 padding and the periphery over there in order to match it down. Now, all of this is done from the perspective of your calculation which says that your output with is equal to input, with input with minus the size of the kernel plus twice the padding which were giving divided by the stride which comes down plus 1 ok. .

So, if you are input over here is 32 cross 32, then what comes down is that your 32 cross 32 minus my kernels size which is 3, plus my pending which is a plus twice of my padding which makes it 2. So, it is becomes 30 32 minus 3 plus 2 over there. So, it becomes 32 minus so, so it becomes 32 minus 3 which makes it 29 and, then I do a plus 1 over there which makes it 28 and that whole thing divided down by my stride of 2..

So, that brings it down into 14 cross 14 sized over there. Now, following that I have a leaky ReLu non-linear transfer function given down over there. So, I could have used on a standard ReLu, but; however, what you wanted to do was on the negative side of it if you are values are coming down negative, then we still try to preserve some sort of a gradient and not just make down the gradient and repose everything over there 0.

So, that would technically for any error which is negative fits it is just not going back propagate any part of that error over there. So, leaky ReLu helps me to do that with the leak factor of 0.1 maps down. Now, these are personal choices and nothing beyond that the more experience you get, you chose it out, but as of now it just (Refer Time: 08:05) chosen out value. The next layer over there is a gain of convolution layer which maps down from 64 channels on 228 channels. So, there are 128 unit kernels and then you have your kernel size of 3, 3 cross 3 the same way and then a padding or stride of 2 and padding of 1 and it keeps on going down and the final part is where from 128 channels, you again connect down to 128 channels.

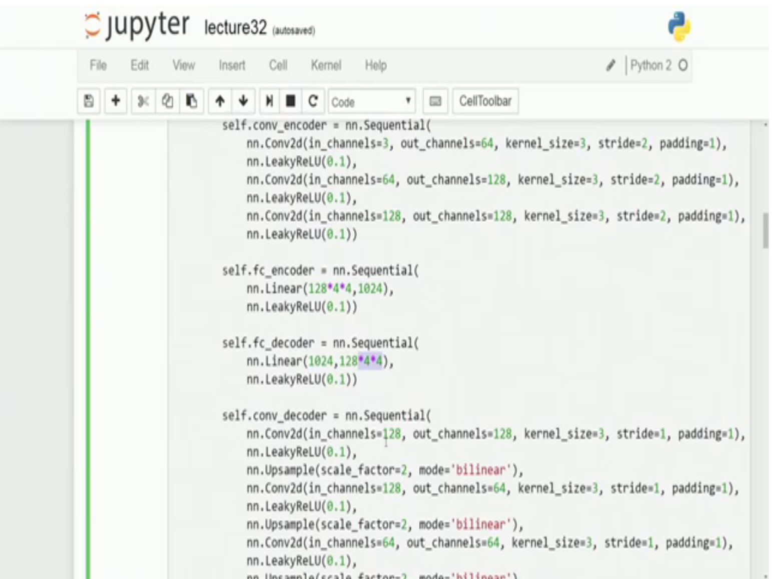
Now, over here by the time you actually least term this linear part over there, what you have is 128 channels are what corresponds over there, this through this whole transformation over there, you end up getting a smaller matrix of size of 4 cross 4. So, the total number of neurons which are present within the full volume over there is basically 128 into 4 into 4. And that is the total number of neurons which will get

connected and the linear fashion fully connected fashion 200 to 1024 neurons and following that you again have a transfer function which is a leaky ReLu.

Now, this complete your part of the auto encoders. So, the like hidden layer which is shared between the encoder and decoder units over there, has just 120 has 1024 neurons present over there, now these 1024 neurons and now been passed on to my decoder unit. So, what I do on the first part is 1024 neurons they will just blow up to the total number of neutrons which is equal 128 into 4 into 4 which is the same as over here. So, this kind of an encoder decoder is symmetrical as it appears over there. Now, following down my transfer function over there, then I start converting all of this onto my conversational block.

Now, since my view or linearization and repacking is not part of my architecture, but they just operations which I do on the data so, that is something which will be defined in my forward functional. Now, let us look into the decoder part of it..

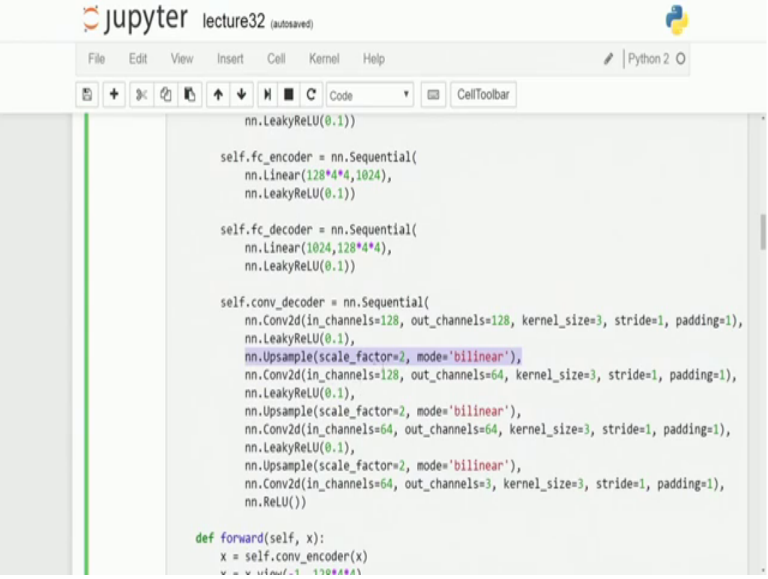
(Refer Slide Time: 10:00)



Now in the decoder it is quite symmetric to how it was going down in the encoder part as well. So, in a encoder you had 128 channel which connect down 128 channels. So, it is a same thing which comes on except for this output the number of channels is what will correspond to the input number of channels and, this input number of channels over here, on the encoder is what will correspond to the output number of channels in the decoder.

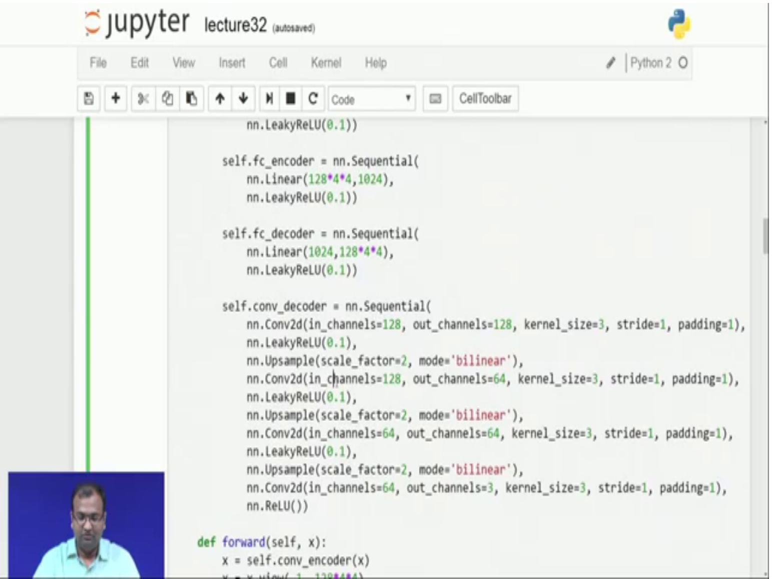
Your kernal sizes are still 3 cross 3 and your stride which you are using is of 1 and a padding of 1; the whole reason being that over here, when I do a stride of one and a padding of 1 I am going to retain object of the same size ok. Now, whatever was the size coming out of here so, here basically it was a 4 cross 4 and that is what comes from over here as a result as well. Now, that 4 cross 4 size 1 which has 128 channels is what is get (Refer Time: 10:47) out over there, then you do a leaky ReLu and then you do up sample. Now, this up sampling over here is just interpellation up sampling which is taking place.

(Refer Slide Time: 10:56)



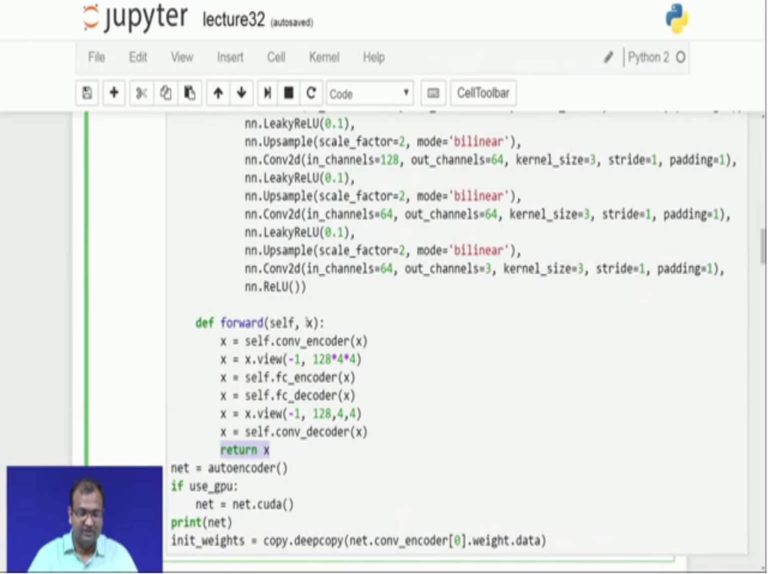
Now, this interpellation will convert this 128 into 4 into 4 size thing on to 128 into 8 into 8 sized ok. And that is something which matches the output, which can be fed down to this particular which is sort of symmetrically fed down to this particular earlier layer..

(Refer Slide Time: 11:17)



And that is what I am going down. So, were here I am going to convert 128 channels on to 64 channels that matches on my encoder sized the symmetrical value. So, you need to keep in mind that whatever is the kind of number of channels and the size on the encoder side is it is just going to revert it in the same way in the decoder side as well. So, finally, when I get down from my last layer over there, so in the last layer I just have a value which is equal to 3 channels and this output over here is going to be 32 cross 32 ok. Now, this is about my definition of the architecture..

(Refer Slide Time: 11:52)



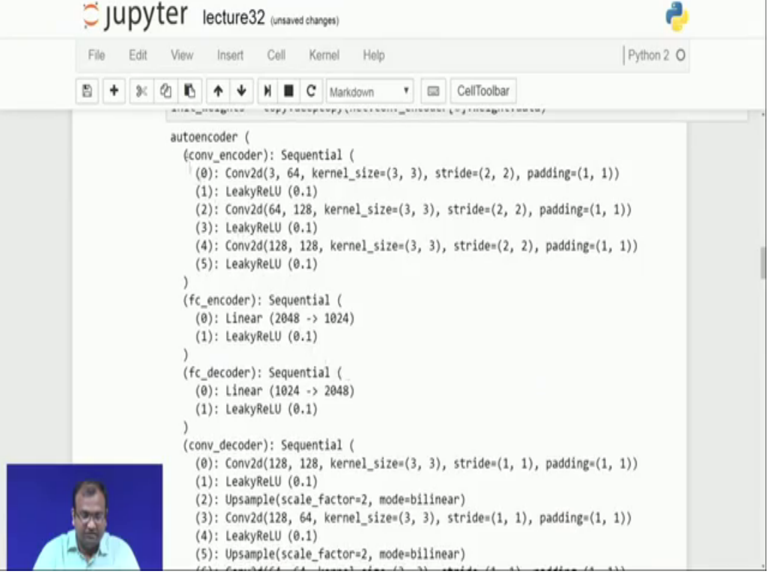
Now when I do a feed forward, but I am going to do is first I am going to just feed through the auto encoder through the encoder block or the convolutional encoder block over here, once it comes down from there I need to linearize each of them and that is by using the view function which is to cast all the neurons into one single linear layer. .

Then from there it feed through the encoder from there, it feeds through the decoder, the fully connected encoder and decoder. Now, once I am over here is just a bunch of linear neuron which have to be undone and again converted into my vector space of 128 cross 4 cross 4 and, that is what I am doing over here..

Now, once that is done you can pass down this 128 cross 4 cross 4 size attention on to your convolutional decoder block over there. And once I come down over here, I see that my output size is the same as that input which I had provided over here. Now, once this is done I can define my network and, then if my GPU is available which for me is there then I can convert it to cuda and (Refer Time: 12:49) and then so, this just for printing the network..

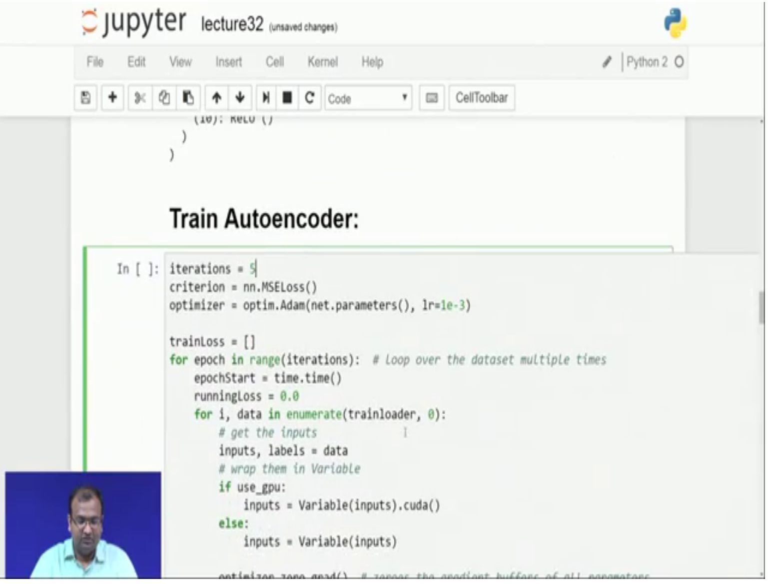
And since I want to look into what is the kind of weight change which has happened over here. So, I would just be copied on my weights and keeping it down. So, let us execute this part and we will see what comes out. So, you have your total auto encoder which is defined like this.

(Refer Slide Time: 13:08)



So, the first part is your convolutional encoder, which has all convolution connections and your transfers in term of leaky ReLu, then you have your fully connected part over there in terms of a sequential part of the encode, then you have your fully connected part of the decoder and subsequent to that you have your convolutional part of the decoder present over here and this maps down pretty much symmetrically one to the other one.

(Refer Slide Time: 13:36)



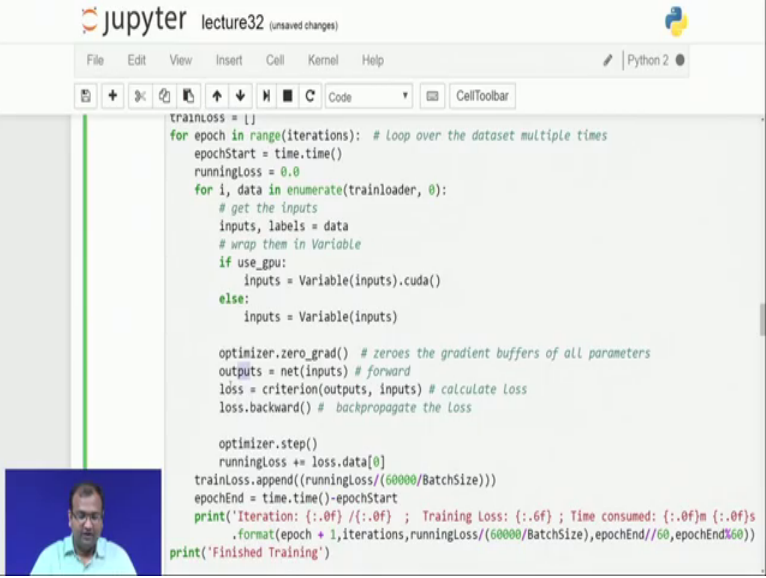
So, once this network is defined for me. So, let us get onto training down this auto encoder. So, what I choose over here is to train it over 5 epoch I have my loss function which is MSCLoss function because this is not a classification problem I am solving in any way, I am just trying to solve down regression kind of a problem it is just trying to encode the image in itself. So, you just use an MSCLoss over there and, I can go in on the best by the best of the optimizations we have at the current stage that is on Adam.

So, let us keep this one running because, it might some time to run it while I go through the codes. Now, over here what I do inside is I set my epoch counter running over here, which runs over 5 iteration or 5 epochs and within each epoch it is going to load down my batch of the data, which is 2000 samples as I had set in the earlier case and, from there what I do is basically look if there is GPU available and since my model is converted on GPU..

So, I need to actually get my data also converted on to my cuda. So, this is where my inputs get converted over there, now following that what I do is I set my gradients in the

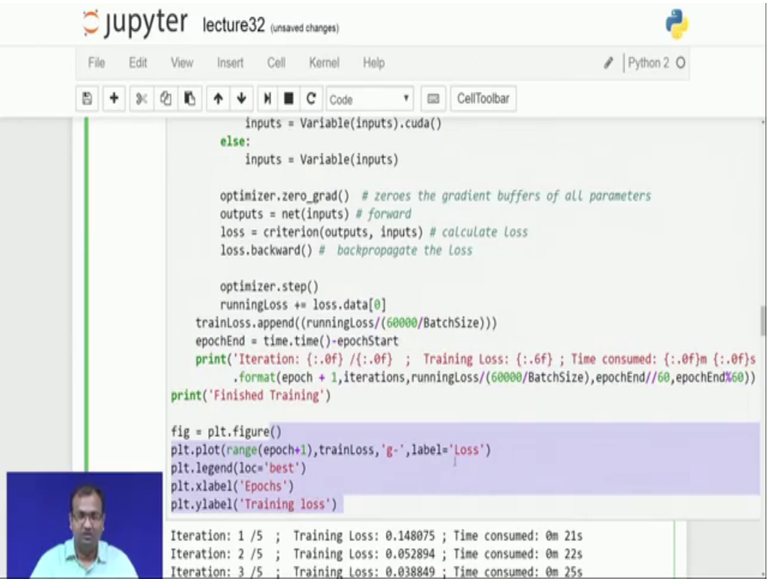
optimizer as 0 and, then I just feed my inputs to the networks such that I get my outputs ok.

(Refer Slide Time: 14:43)



Now, on my criterion function over here since my loss all or the cost function, which I was going to use is an MSCLoss function which basically tries to look in to minimizing the error between whatever is the input and what comes from output side of it. So, for that reason the argument which goes into my criterion over here, is there is an output which come over here and the reference in the earlier case when you are doing classification this used to be labels, but for me these are my input values over there..

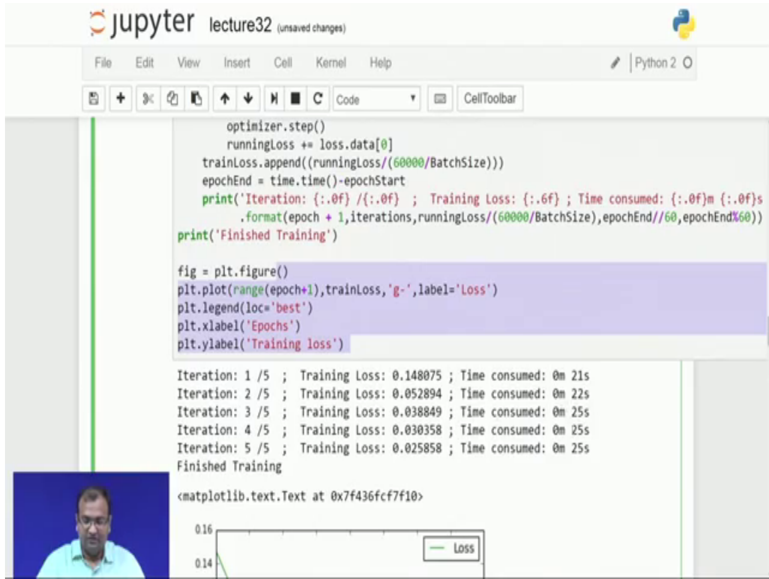
(Refer Slide Time: 15:15)



Then once my loss is calculated I need to find out nabla of loss, or the gradient of the loss. So, once that is done then I do my optimizer dots step which is to iterate back propagate the whole radiant through the network, in order to an error back propagation and, then I just do a accumulation of the loss as it keeps on going over though number of batches which goes down. Now, with this what I do is technically take down an average loss per epoch and, so my back size over there was 2000..

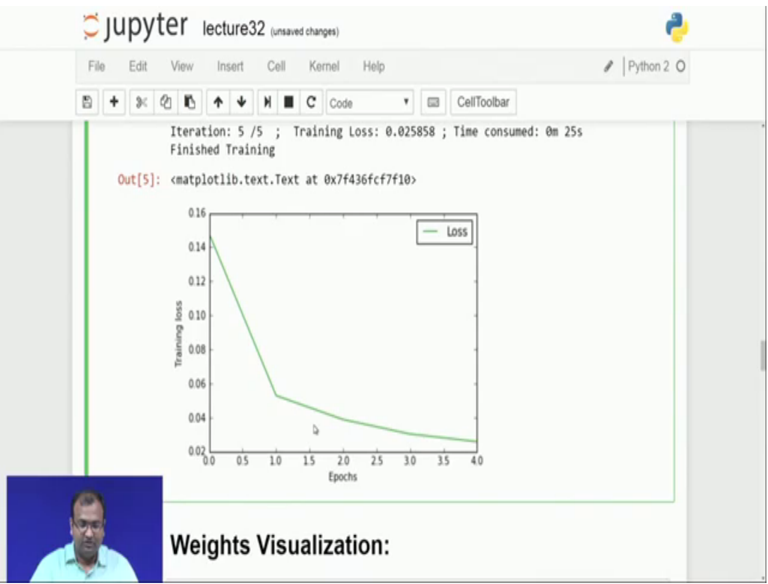
So, what I am doing going to do as basically 60000 are the total number of sample present and each batch taking down 2000 samples over there. So, it ends up being just 30 number of batches which will be present when I am trying to do an update per epoch. So, just take an average over that and, that is that is the average loss per epoch and, then I just keep on saving it out do a timer and then finally, it is to plot down what the loss looks like. .

(Refer Slide Time: 16:12)



So, if you look over here it takes down about roughly 25 seconds to run it down we just run it 2 iterations over there..

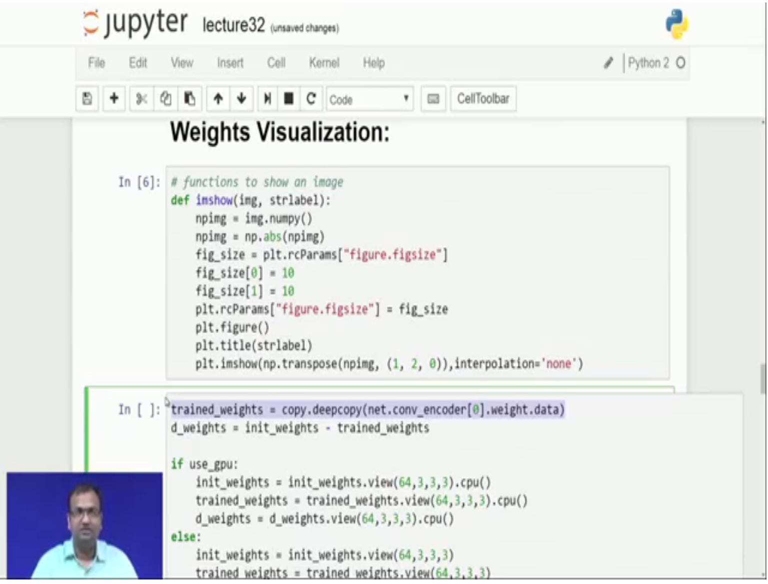
(Refer Slide Time: 16:19)



And this what comes down on the loss side over there. So, it is started with an MSCLoss somewhere around of a 0.14 when down to a loss of 0.024. So, there was good amount of one point of decimal loss coming down; however, it has not yet resaturation. So, please keep on running over a long duration of time, or you can even put down learning rate update rules over there to come down much more conversion; however, the interesting

part is actually to look down into the visualization and this is what we have been doing for the last few lectures. .

(Refer Slide Time: 16:44)



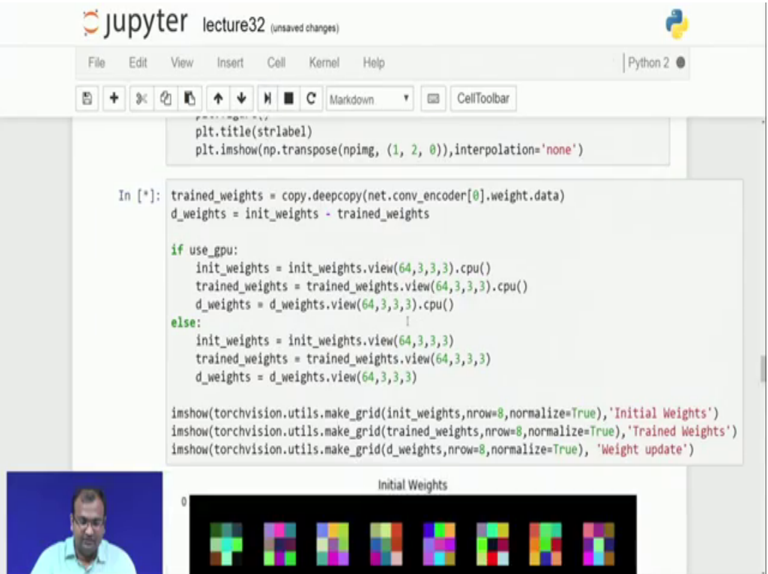
So, let us get down so, what I have is a just a simple definition of the function, to get on my weights and visualize it out. And since initially I had just copied on my weights and kept it. So, I am going to use my initial weights. So, what I do over here is when the network is completely define, then I make one effort to copy down all my initializasion grades, or the initialize weights, which are just random initialized over all the kernels which are present over there. Now, that helps me in a significant way because, now I am quite prepared in order to look down what happens after the whole training process is over.

Now, once a whole training process is over after defining this function what is do is actually copy down all the weights, which are have been trained. And then my next part is to actually look into what is the difference of the weights, which is which gives me an idea of how much each weight over there has been updated within each of the kernels. Now, having done that the next part is that your weights can exist either on the CPU or on the GPU, but based on like that you are executing, if you have a GPU available then it pulls it automatically and goes down on to the GPU in order to do it. .

Now, it would just need a type casting over there because the rest of the functions for displaying out and then showing out the weights, there have what exist on the CPU and

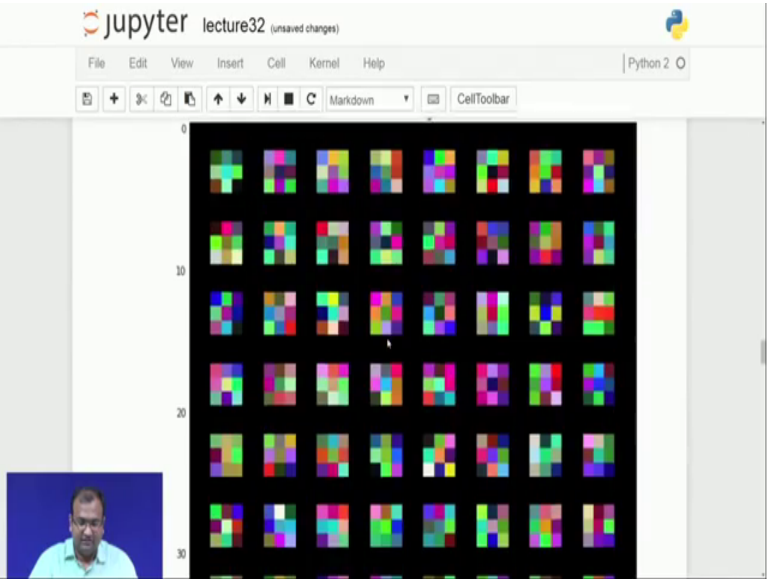
not on the GPU. And that is the whole purpose of doing this type cast over here and, the finally, we just use a visualization to in order to see this out ok.

(Refer Slide Time: 18:11)



So.let us look into what it goes..

(Refer Slide Time: 18:15)



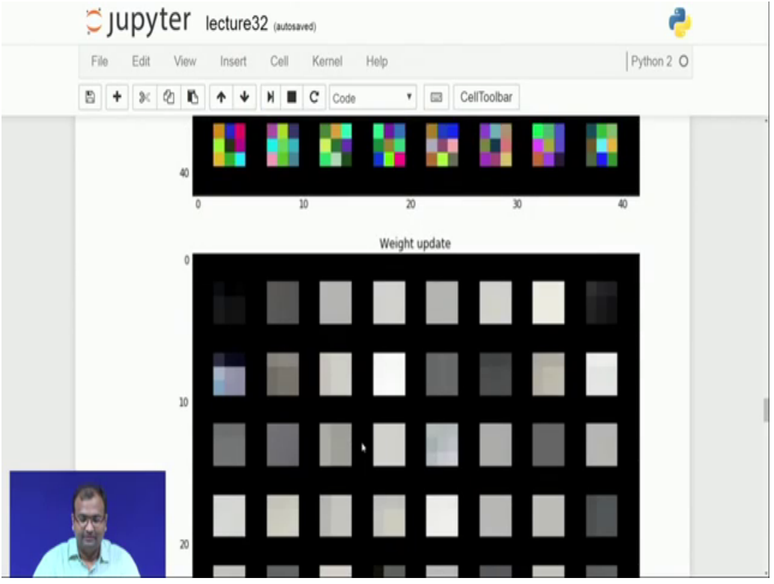
So, my initial weights something which looks like this ok. Now, you need to keep something in mind that we had actually created a model. So, if you look into this model over here and, the first layer which is connecting down any input which comes down into 3 channels into 64 such kernels over there and, then since it is a 3 channel input over

there my convolutional kernel is of the size of 3 cross 3 cross 3 my 3 cross 3 is the special spared over there and the number of channels is also 3 ok.

So, this is technically as if for 3 cross 3 rgb made fixes what we are going to look down into and, there will be 64 of them those unique numbers. Now, let us come down into the visualization. So, if you go over here and then the matrix which is forms has 1 2 3 4 5 6 7 8 number of kernels on this axis. On other side also you will be seeing down that there are 8 number of rows, which down and that together make it as 64 number of kernels which are visualized. And each is a 3 cross 3 in special split and has 3 channels that is why you get your color visualization coming down over here. And that is quit conformal to since we have the input in terms of a color images itself ok.

Now, these were the weights at the initial part of it and, then once you have train out the whole process, then you get down your weights which look down quit similar to this one ok. Now, for most of you it might not even make sense because, it look as if had not much of a change.

(Refer Slide Time: 19:41)



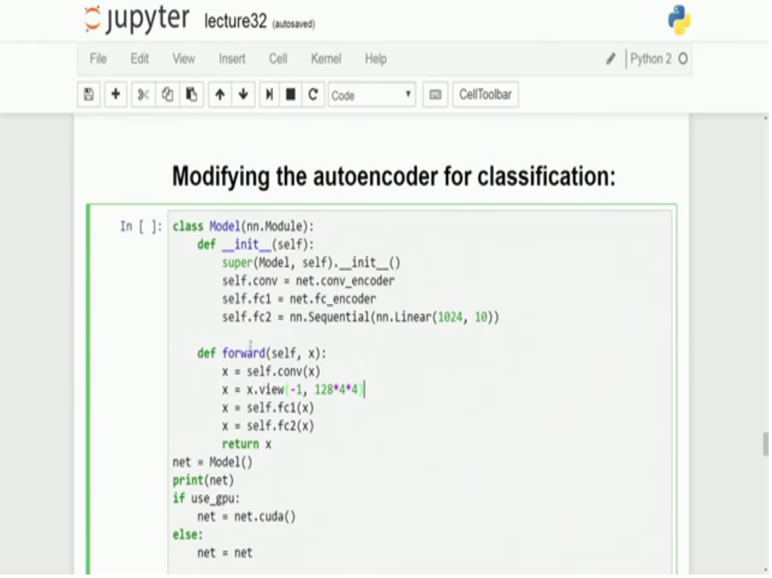
Now, if you look down into your weight update, then you can pretty much make update this updates which have happened over here are sort of like which appear more of scalar in nature, like whatever was the initial randomization that has been just some sort of (Refer Time: 19:53) shift happening over there. Now, that is not unlikely; however, for some of them there has been a different wearing nature of the ship which comes down

and on over here you see that there has been a different shift in each of these color planes ok.

If you keep on running this over a longer period of time as we have done in earlier cases with adjusting trying to use down convolutional networks of the very basic nature, you can actually see the trend setting in it consider we just train down for 5 epoch for sake of time within this video it trains out, but you do not see such a major difference coming down over here. If you set the training with learning update tools and oven longer number of epoch and maybe something like 100 epoch over there and wait for some time, there is each for me this is taking down about 25 seconds..

So, for you it might vary based on whether you are using a CPU only or GPU kind of a mechanism, but it is it is a nonetheless one thing you need to remember is that these mechanism of using a deep learning network, they do consumes a lot of time for training and, you will have to just wait and watch nothing more than that, a good method really excellent one, then the time consumed is also large. .

(Refer Slide Time: 21:00)



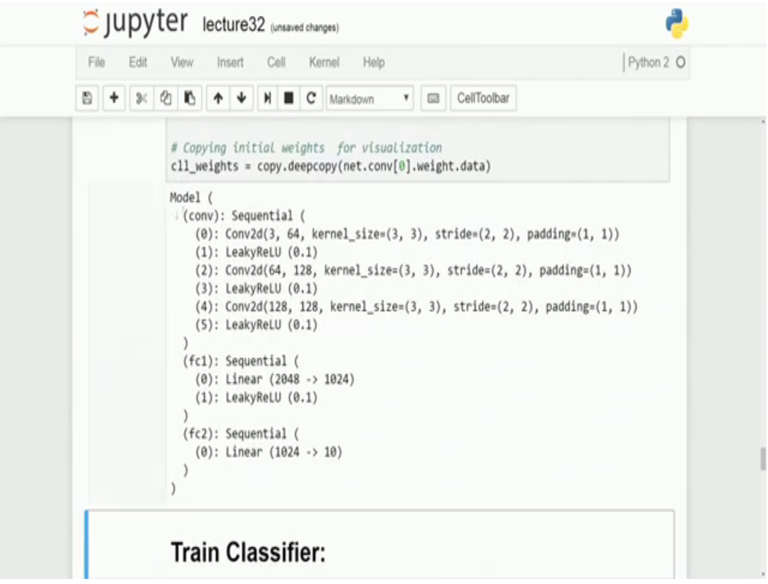
So, this ends of the first part of it, which is just with learning looking into what representations it has done, but then the objective of actually learning this representations was so, that I can have a classification network which is very accurately initialized onto what features makeup the data ok. Now, the next part is to modify this auto encoder part

which has a convolutional part over there, in order to make this into the classification network..

So, I technically do over here is that I define a newer kind of a model, which is just going to take down my initial part of the convolutional encoder and, my fully connected part of the encoder and then what is whatever it is the resultant over there which is 124 just connect it down on to 10 classes over there, via a linear connection there is a only change which comes down and due to this change you have your forward function also changed over here. .

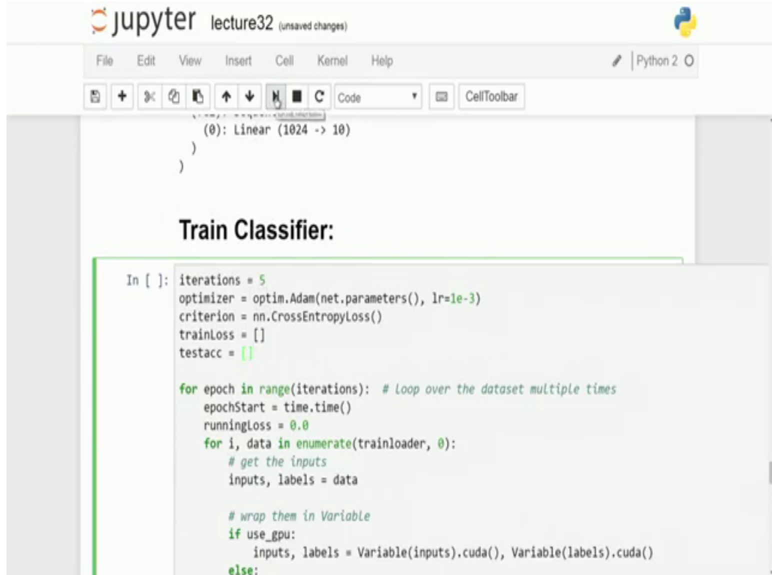
Now, if your GPU is available just convert it to the GPU and, then you can just copy your weights and keep it for your purpose. Now, we have a model which has already been trained an initialized to do a good amount of representation leaning.

(Refer Slide Time: 22:00)



What we are going to do over here is basically yeah, this is what your network looks like, you have your first part of it which is just convolution, the second part of it is the sequential and, the third part is what were just now introducing in terms of classification network ok. Now, we are going to train down this final classifier..

(Refer Slide Time: 22:20)



So, whatever has been optimized earlier with unsupervised mechanism, we are just going to retain all the weight over there and, then do it this will be trained over 5 iterations. So, let me just keep it running and then I can do the rest of the discussions..

So, now, over here if you look so, my optimizer still Adam not much of an issue; however, since my task has changed out and it becomes a classification task. So, the criterion function of the cost I use is a cross entropy loss ok, the rest of it stays as the same; however, over here for the first time I am going to make use of my labels.

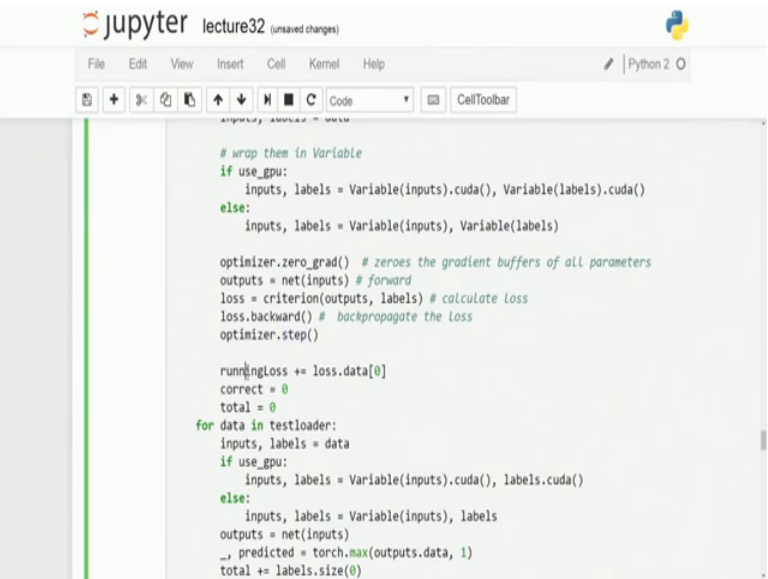
(Refer Slide Time: 22:54)

```
jupyter lecture32 (unsaved changes)

In [9]: iterations = 5
        optimizer = optim.Adam(net.parameters(), lr=1e-3)
        criterion = nn.CrossEntropyLoss()
        trainLoss = []
        testacc = []

        for epoch in range(iterations):  # Loop over the dataset multiple times
            epochStart = time.time()
            runningLoss = 0.0
            for i, data in enumerate(trainloader, 0):
                # get the inputs
                inputs, labels = data

                # wrap them in Variable
                if use_gpu:
                    inputs, labels = Variable(inputs).cuda(), Variable(labels).cuda()
                else:
                    inputs, labels = Variable(inputs), Variable(labels)

                optimizer.zero_grad()  # zeroes the gradient buffers of all parameters
                outputs = net(inputs) # forward
                loss = criterion(outputs, labels) # calculate loss
                loss.backward() #  backpropagate the loss
                optimizer.step()
```

Earlier I was just going to make use of the image data for learning, but I never use labels over here, since it a supervises one in order to classify it in 10 classes so, my labels comes for the first time. So, going down by what we have is you 0 down your gradients do forward over the network and, then you are going to took take a look into how good is the classification over there..

And for that you have your output which are generated over there, that is output for 10 neurons and then you had your labels which are given an over here for comparison purposes..

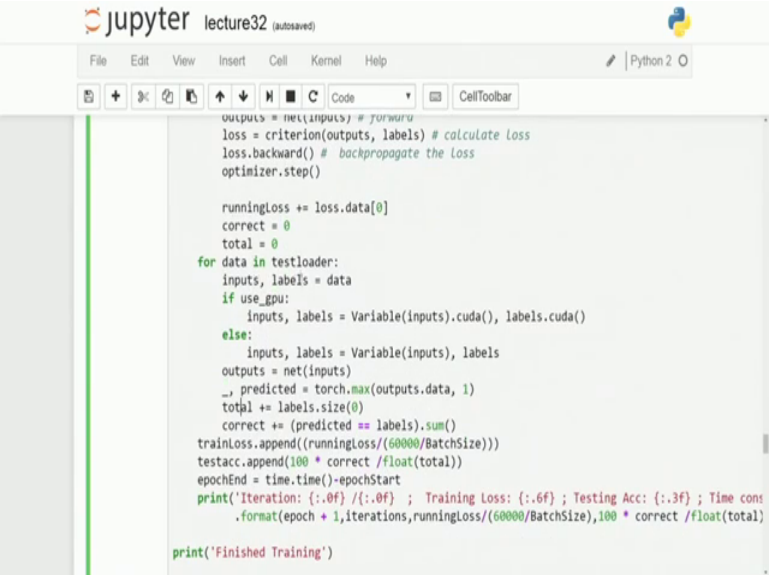
(Refer Slide Time: 23:36)



Now. once your loss is computed you find out nabla of the loss on the gradient over that and, then iterate over the optimizer in each step and that does your back propagation coming down. You calculate out your running losses over all the number of batches which work done over there and, then you can actually use that for the rest of the purpose..

Now, following that what we also try to do is as we were doing in earlier cases is try to load down into how good has this network train in order to do a classification process..

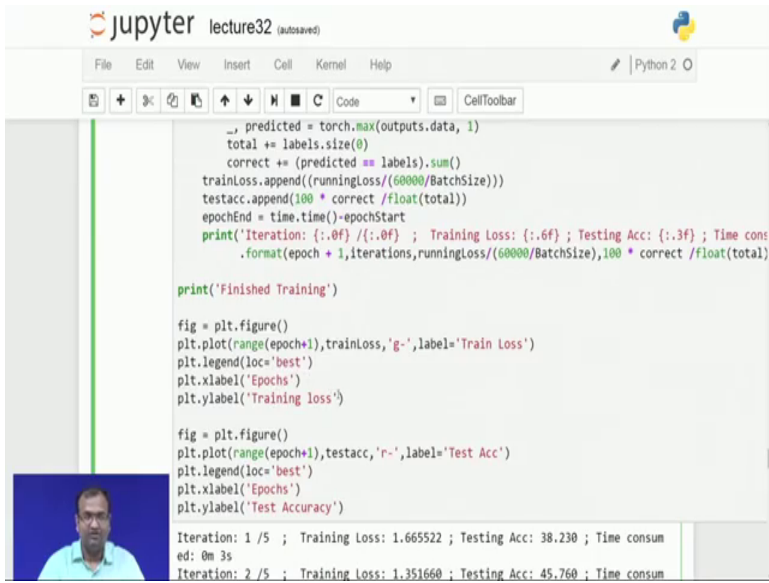
(Refer Slide Time: 23:56)



And, for that purpose what we do is we just try to take down data from my testing dataset which is independent. So, I have not use any of the data in my testing dataset in order to train the full network, I am just going to take that part of the data and, I am I just iterated over all the samples on my test data loader, which I have defined in the start of this whole code itself . And, now if my data is available if GPU is available just get it converted onto your cuda and, then the rest part is quite simple..

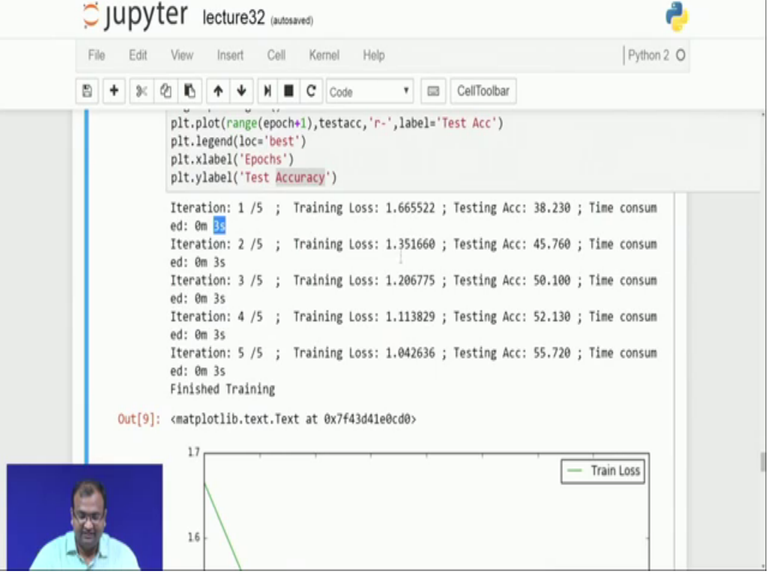
So, do a feed forward over the classification network, you get down whatever it is that predicted index location which has the maximum probability which comes out and then, you see whether the predicted index is correct in terms of just matching it down whether it matches down the label for that data or not. And then you normalized it over all the batches, which are running down and then you can just have a good look on to the whole thing by doing this 2 plot.

(Refer Slide Time: 24:47)



So, one of them is going to plot down the lost in terms of classification learning, the other is going to do an independent accuration of the independent estimation of the accuracy on the test dataset which is available to you. .

(Refer Slide Time: 25:00)



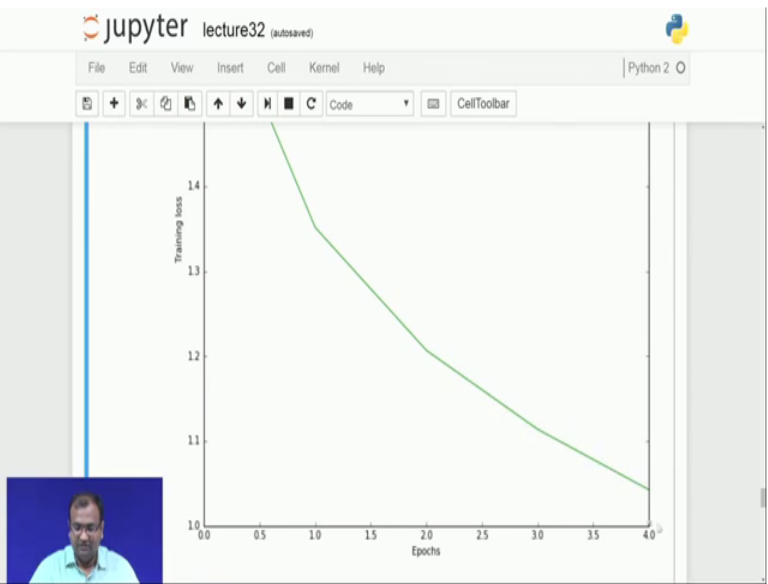
So over here this takes lesser amount of time as you see it takes down about roughly 3 seconds in order to take the whole network and, that is quite common because in the earlier case it was taking more time since you had the convolutional block which was there in the encoder, you had a convolutional part of the connection in the decoder and,

you had fully connected layers as well coming down. You have you have actually chopped it down significantly and, then we reduced some of the major costly operation which are in terms of using bilinear interpolation to up sample it out.

Now, if you are just linear interpolation over there on nearest linear interpolation; it is much easier and less costly operation. So, these are places where just by introducing some changes, or by in case you accidentally take a in effective mechanisms you are just going to increase the total time complexity over there. So, the moment you changed it out because, the rest of the down sampling is something which happens in much smaller time, then the amount of time needed to do linear interpolation over there.

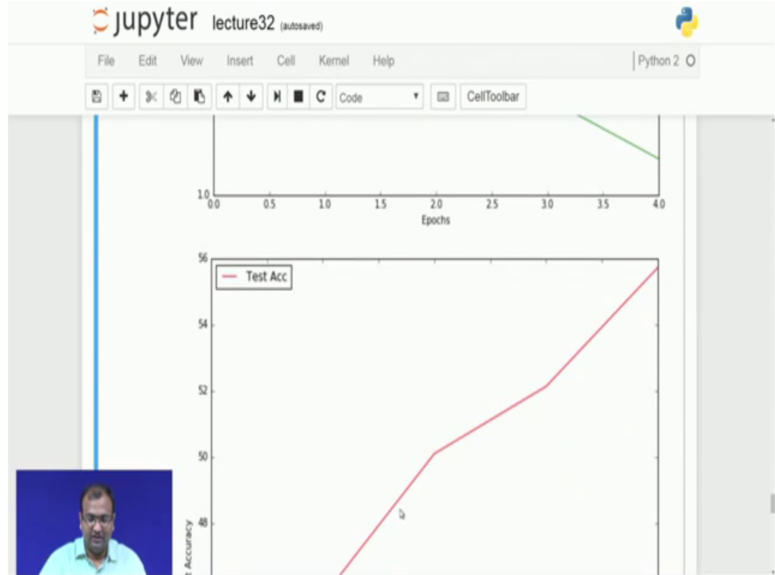
Now, from there we look in to it that your loss which was maintain in terms of cross entropy, it starts somewhere around 1.66 and then it goes down at the end of 5th epoch to 1.04 that has not been much of change to say practically and looking at the end of this curve.

(Refer Slide Time: 26:13)



Keep it running for 100 epoch it would definitely hit down, the saturation in a much better accuracy.

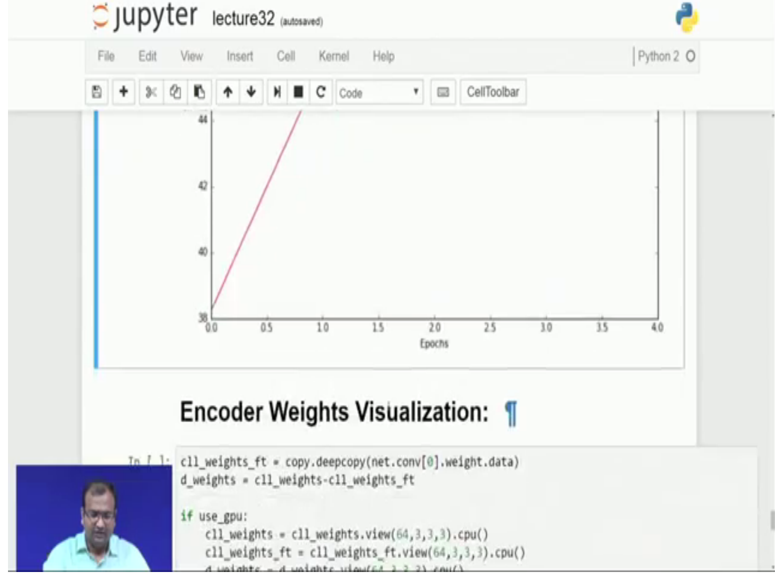
(Refer Slide Time: 26:19)



Now, if you look into the accuracy part over there it starts somewhere very low at about 38 point some percentages and, then at end of it goes down quite high to about 56 percent. Now, not an impressive figure, but definitely the is a factor of about 20 percent 18 percent roughly is an increase which you get down just by training for 5 epochs..

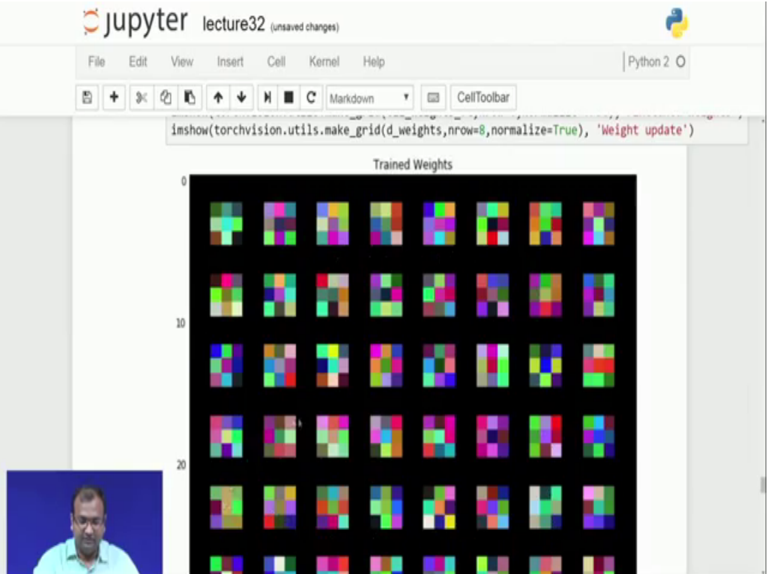
Now, if your initialization in the earlier case say, you had kept your auto encoder training over 100 epoch and whatever you had initialized, you use the same thing for this classification over here, you would have started with a much higher accuracy because your features are much better. So, it is it is it is in the same line as we had leant out with our standard auto encoders in the fully connected because, in the same kind of concepts of learning and better initialisation do have a benefit cumulatively transferred over here as well.

(Refer Slide Time: 27:07)



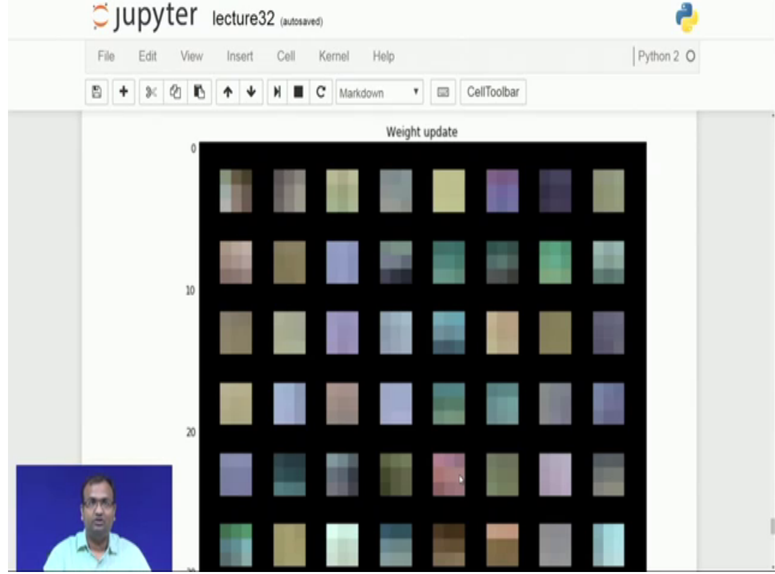
Now, once that part is done let us look into the weights which come down, at the end of this process..

(Refer Slide Time: 27:15)



So, these were the weights which come down after the end of this whole training process over there ok..

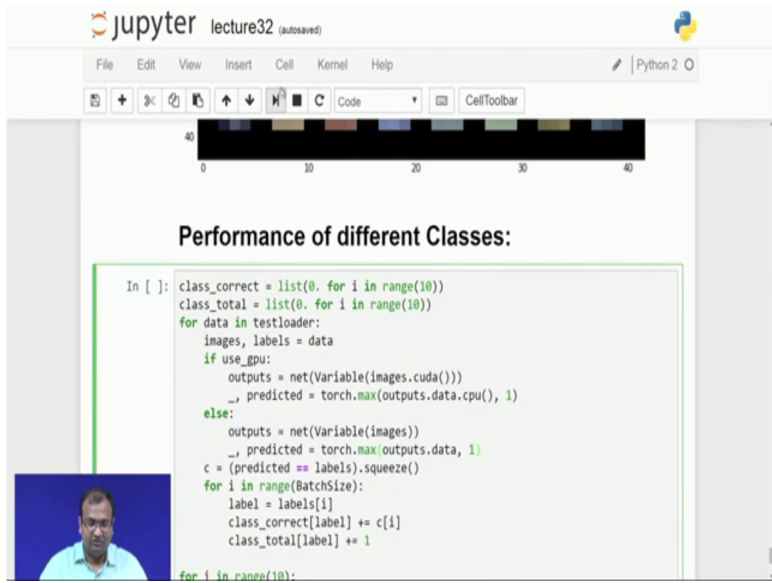
(Refer Slide Time: 27:23)



Now, if I look down over here so, you would be able to see down your weight updates which come down, now you can see pretty much these weights update are where you see a change in color coming down. And this was happening because one phase of it was just trying to represent the data itself, but the next phase is to get down more and more better kernels which are better for classification. So, some of these kernel which were redone then (Refer Time: 27:43) says that whatever be the class of the objective should always be firing up in the same way they are the ones which get modified to the maximum extender in to increase your accuracy over there.

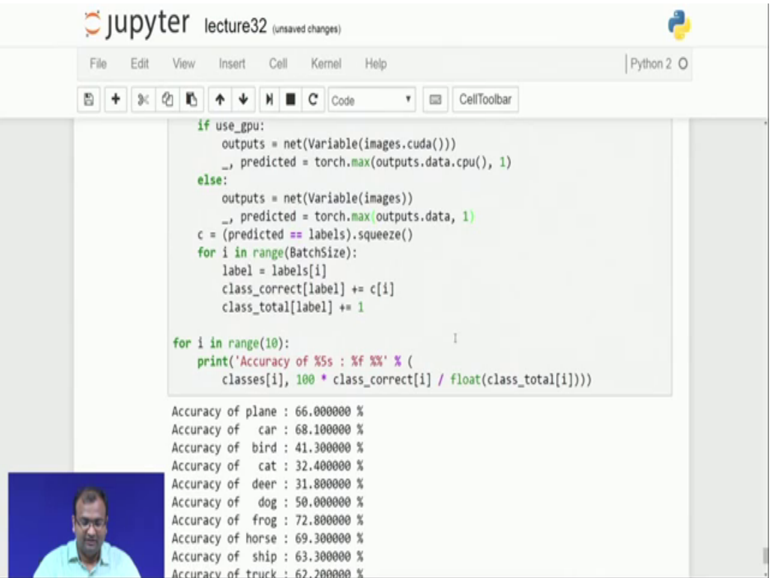
So, just these are for your own understanding and then how you would be getting it done over all the other source of data coming down for your work..

(Refer Slide Time: 28:08)



Then next part is what we strive to look into the performance of classification of each of these class.

(Refer Slide Time: 28:10)



So, now, there were 10 classes which range cross plane car bird cat and then this is the average kind of a performance which goes down, you see that frogs get classified in the best possible accuracy ah, deer has the worst accuracy coming down over here, cat has a bit (Refer Time: 28:26) better than the deer, but most likely it might be confusing all cats

with deers and, that is pretty much because it is a small size image over there of size 32 cross 32.

Now, this is just to give you the connections and how we are going to make use of even unsupervised learning, as in auto encoders within trying to get on a convolutional network as well. So, in the next classes when we get into more deeper convolutional practical networks like VGG nets and others, you can still use this kind of similar convolutional encoder decoder network in order to initially train them though they are typically not trained in that way given the computers complexity. So, just keep on waiting for the next lecturers and till then goodbye.