Deep Learning for Visual Computing

Prof. Debdoot Sheet
Department of Electrical Engineering
Indian Institute of Technology, Kharagpur

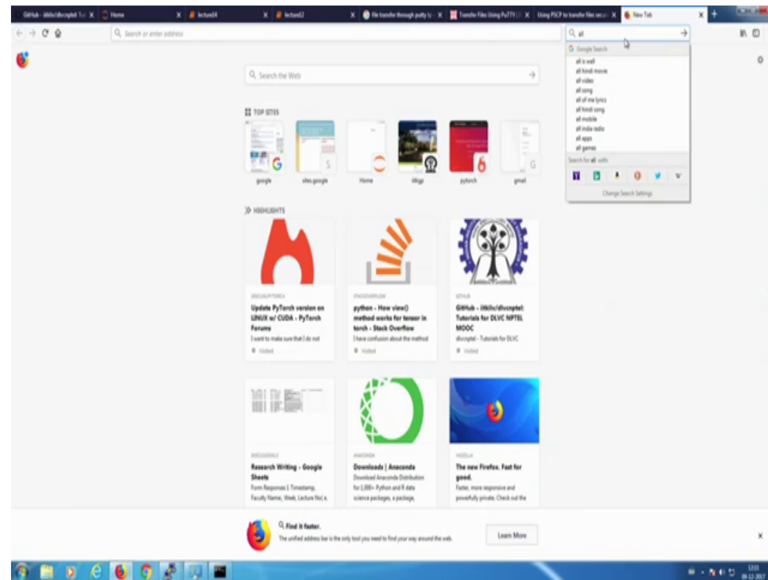
Lecture - 14
ALL-IDB Classification using autoencoders

So, welcome and today we are going to do basically using an autoencoder for batch wise classification, but then, this is a bit different from what we had done in the earlier class and here we were going to deal with color images and that is for the first time that we are doing it. So, earlier everything was on grayscale images where it was quite easier because you just had integers over there and then these were used in order to linearize out them and it is just a 1D matrix, but then when it comes down to the matter of color image, the point is that you need to have some sort of a 3D matrix on which it would be now working on.

So, you have each plane of your color as the third dimension of the matrix over there. So, for making it simpler, although you had been exposed to over to the cifar 10 classification problem, but then cifar is a data set which is much more complicated to be handled down using simple autoencoders. And in order to, though we will be touching on them for our convolutional neural networks.

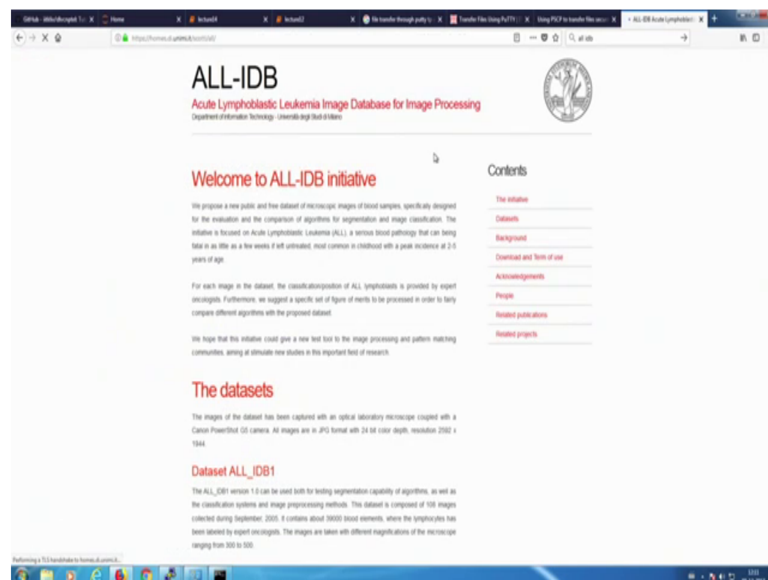
But then for the purpose of autoencoders, we are going to stick down to a much simpler one and that is using something on the medical domain and these are all microscopy images. So, we are going to use, make use of the ALL-IDB1.

(Refer Slide Time: 01:38)



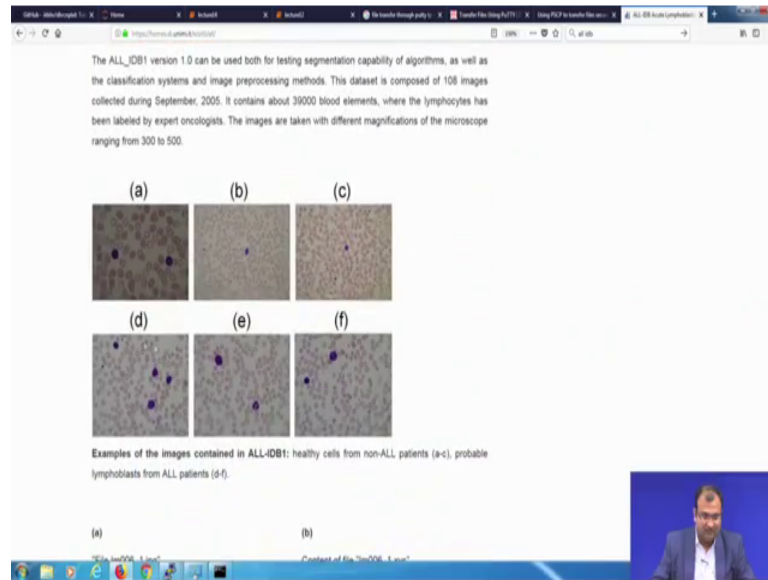
So, let us show you how to get down the ALL-IDB. So, you can just search down for ALL-IDB data set and this is for Acute Lymphoblastic Leukemia images. So, you end up coming down to a page like this.

(Refer Slide Time: 01:48)



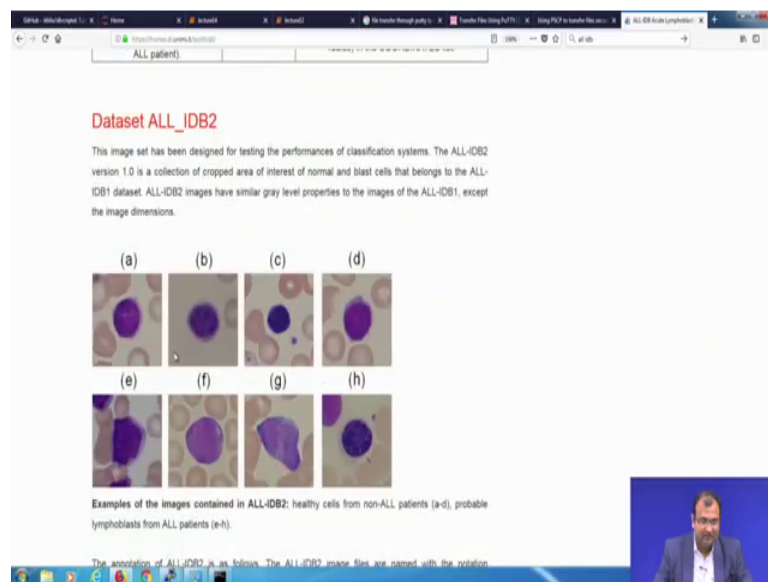
This is a publicly available dataset not much of an. So, these are basically the one.

(Refer Slide Time: 01:57)



So, there are 2 datasets over there, one is ALL_IDB1 which is a full scale microscopy image on which you have to identify these leukemia cells.

(Refer Slide Time: 02:10)



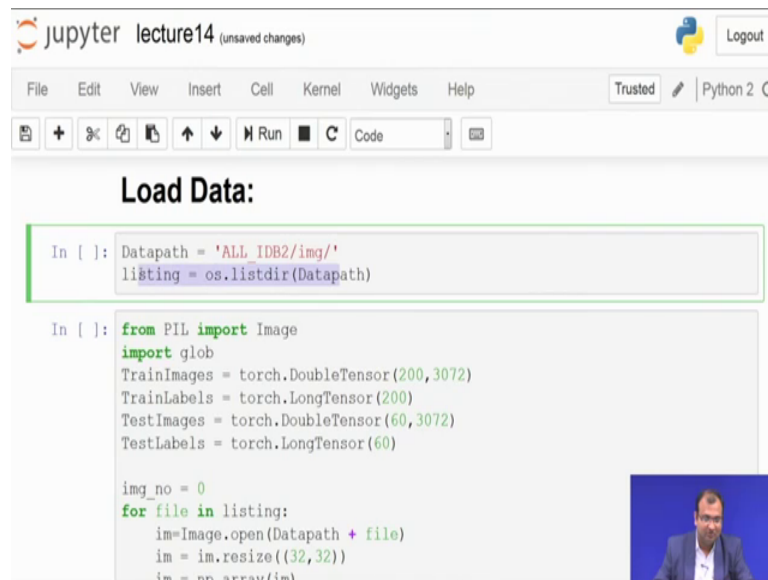
The other one is which is ALL_IDB2 and this is where you have small patches. These are quite small size patches not all of a similar size, but each has just one of these images offered WBC available over there.

Now, some of them are healthy like these ones are the healthy and these were the ones which are Leukemic. And the whole objective over here is to segregate these two. So,

this becomes a 2 class classification problem. The data is obviously of a higher dimension. So, you can just go on this website of ALL-IDB and just register yourself and then get down ALL-IDB2. So, it is a simple form. So, we just go over here. You get this application form over there as a pdf; you can just download write down, sign it off and send it out to the email id which is mentioned over there and get back to you with a link and then using that link you can download the datasets.

So, once the data set is available to you, it will come down to you as a zip file. And what you need to do is, basically get that zip file within the folder where these codes are kept down and then unzip it locally over there itself. So, do not keep it at any other location; just keep it over there and unzip. And then, whatever is a directory structure within the zip just keep it as it is because we are going to use the same kind of open. So, once that is done, the first part is the header structure and that this remains pretty much standard for us across all of our experiments; we are not making much of a change.

(Refer Slide Time: 03:37)



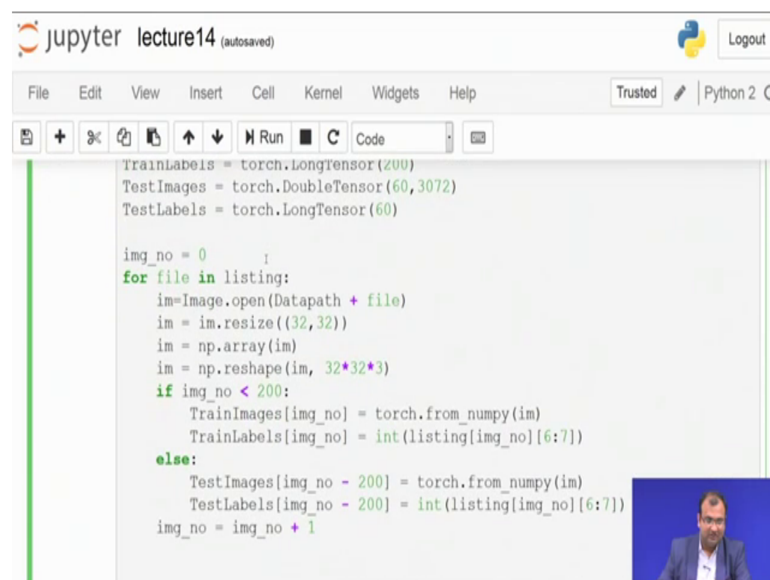
The next is to load down the data. So, as I said that, just keep this one within your same data structure. So, if you are not changing the file names or anything then it will create another new folder called as ALL_IDB2, ALL underscore IDB2 and within that you will have a folder called a slash img which has all of these images and then, there is also another one which is a data path description and classification labels present over there.

So, that is not much of our issues, but just creates this one over there and from there you can fetch down all your file names of images. So, they will be something like img001, 002 kind of labels over there. So from there, the first part is to basically convert all of this image which are available as image files, normal png files in to form a torch tensor and that is what we are doing over here. So, here the objective is just to read down each of them and then convert it into our available Tensors.

Now keeping in mind that, there are just 200 images available for your training and 60 images available for your testing over there as we see and then these images are resized into 32 cross 32, just in order to keep it conformal to a smaller size and there are 3 such channels. So, 3 into 32 into 32; so 32 cross 32 is 1024 and 1024 into 3 is 3072.

So, that is what it just brings down to us as 3072. So, in the earlier case with your MNIST, you had seen 784 because of a 20 cross 28 and they were all grayscale images. So, you did not have the channel concept coming down over there. But here since we have color images, so the 3 channels also come down to play and then we have 3072 neurons going into it.

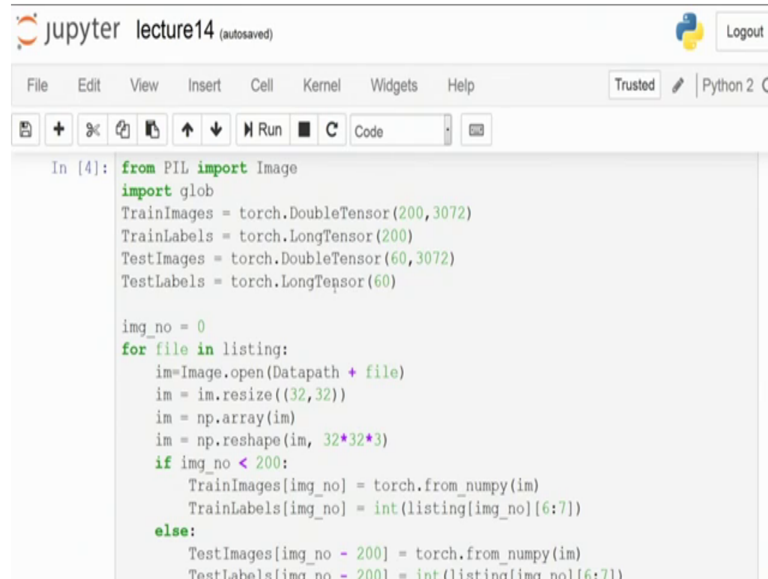
(Refer Slide Time: 05:31)



So, that is about getting your images ready and where you see the change coming down over there. Now, in order to divide it into training and testing, the idea has been implied something of this sort that if image number is less than 200, you are just reading it down

from the directory structure over there. The first 200 images will go into your training and the rest of the 60 images will go into your testing space over there.

(Refer Slide Time: 06:01)



So, we run that and, here now that you have your data listed down on your data listing over and available to you. The next part is basically to read down your images. So, here what it basically does is that your images are available into multiple different sizes. They are not necessarily 32 cross 32 or say 28 cross 28. But, here we will be taking down the advantage that we will make use of 32 cross 32 resized versions of images. And since it is 3 channels, you are effective number of neurons which had to be there on the first layer of your auto encoder becomes 32 into 32 which is 1024 and 3 channels, so into 3, that makes is 3072 neurons over there.

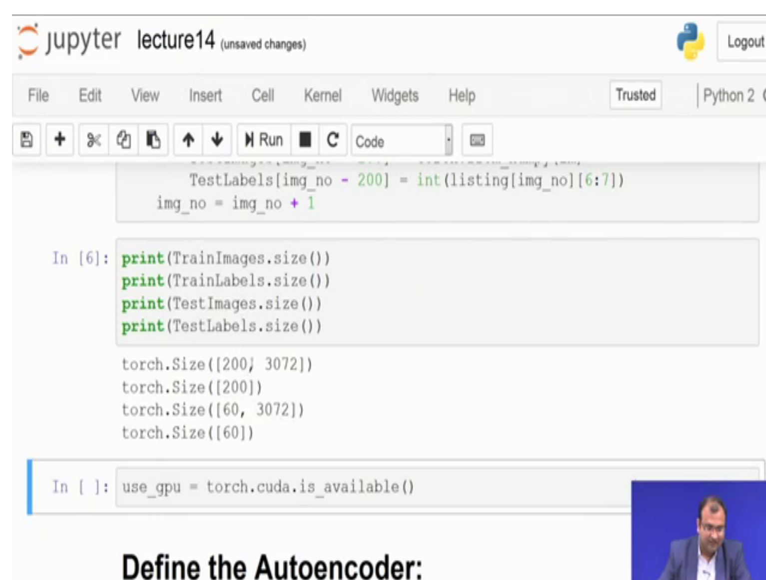
So, we decide to take 200 out of 260 images as your training images and 60 of the remaining as your testing images over there. So, this is just to define down how to handle down your tensors within torch for your training data and labels and for your testing data and labels. And then, here what we do is basically, we open up each image over there and then resize it into a 32 plus 32 patch because each image itself has a different dimension and they are not actually always square in it is own way. So, they can be of higher order set, 200 cross 200 or even 200 cross 250 sized and each image is quite varying because they were just random clips from the main microscopy slide where they were taking down.

So, here the objective is to get down these into a standard form and then reshaped up into one single 1D or linearized out array. And then for the first 200 images, we create the training set and for the rest of the 60 images, we create my testing set over there. So, this is a particular problem where you are using a much smaller corpus of data because for your MNIST and fashion MNIST when you are doing, you are using 60000 for training and just 10000 for your testing.

Here you see that your training has really lesser with 200, so obviously it does have a downside that you are prone down to over fit and memorize out samples, but then you need to keep one thing in mind that the number of samples which you will be needing for any kind of a supervised learning problem has a direct dependence on what is the diversity of the space you are trying to learn. If your space is not very diverse, you would not be requiring too many numbers of samples. You can pretty much make do with lesser number of samples.

So, here is one of these cases where the space is not so diverse. So, you do not need a lot of samples; you can pretty much do it with lesser number of samples. So now, the point is that we can just run this part over there and then try to look into what is my training and testing size.

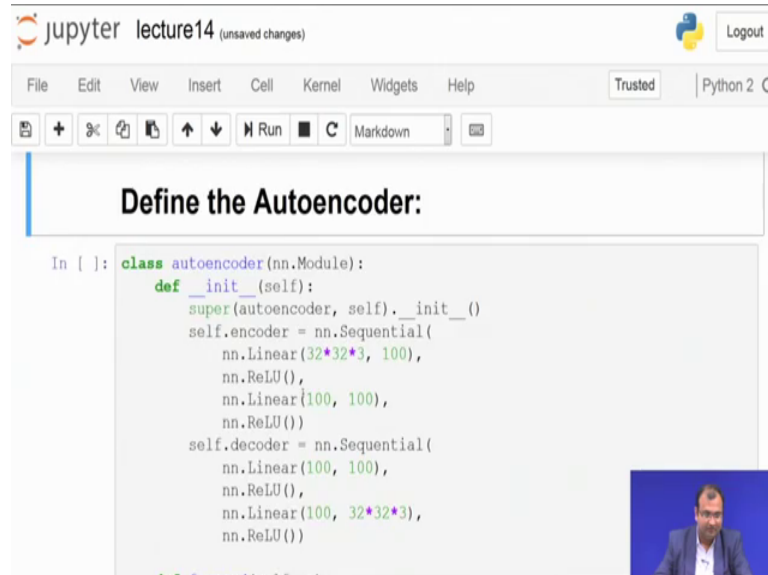
(Refer Slide Time: 08:38)



So, you can clearly see that I have 200 samples for my training and 3072 linearized neurons over there which represent each single image that is one way. Now this is my

flag to check out whether my gpu is available and then we start by defining our autoencoders.

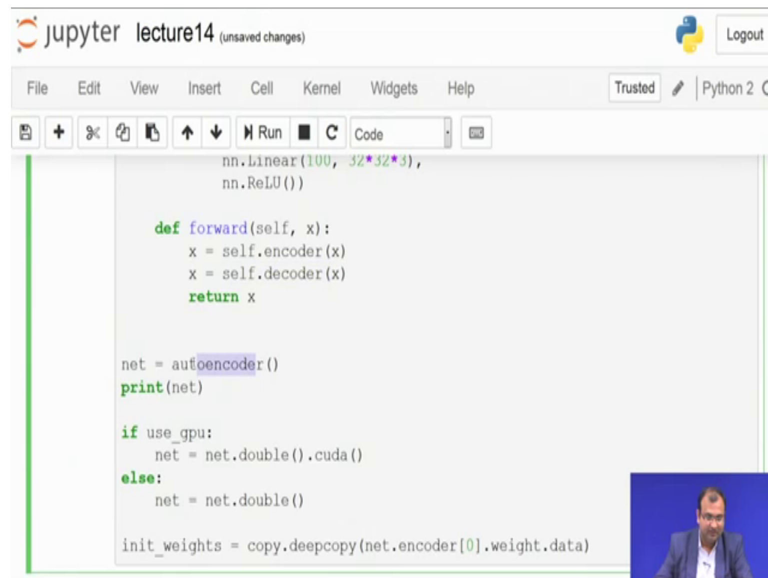
(Refer Slide Time: 08:56)



So, here the objective is that you have 3072 neurons; you scale it down to 100 neurons. Apply a rectified linear unit as a transformation function, from there you connect down 100 neurons to another 100 neurons and then you have a ReLU. So, this is typically my encoder unit where I have 2 hidden layers; the first hidden layer with 100 neurons, the second hidden layer also has 100 neurons. And you just have ReLU as a transfer function.

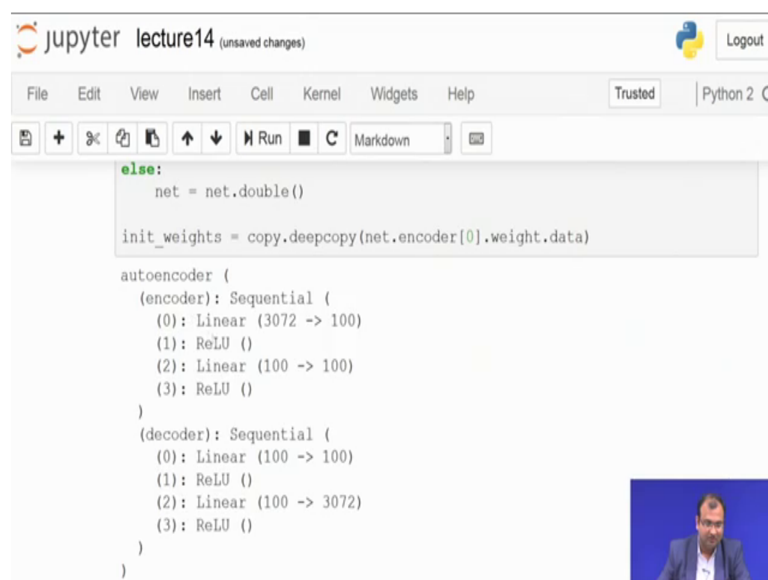
On my decoder side of it, what I do is, from my output of my second hidden layer which has 100 neurons. I go to the next hidden layer which has 100 neurons and apply a ReLU transfer function, from 100 neurons I go down to 3072 neurons; 32 cross 32 cross 3. And that is how I have my auto encoder different. And then my forward pass says, as we have been doing for all of our other networks, it is just a forward pass over the encoder and a forward pass over the decoder.

(Refer Slide Time: 09:52)



And that together defines my auto encoder and then I can print out my auto encoder, I can convert it to a cuda array if I have gpu available and then as with earlier ones where I was just copying down the weights, here also I copied down all of my weights and keep it with me to visualize what is the difference and how much of it has been learned before training to after training.

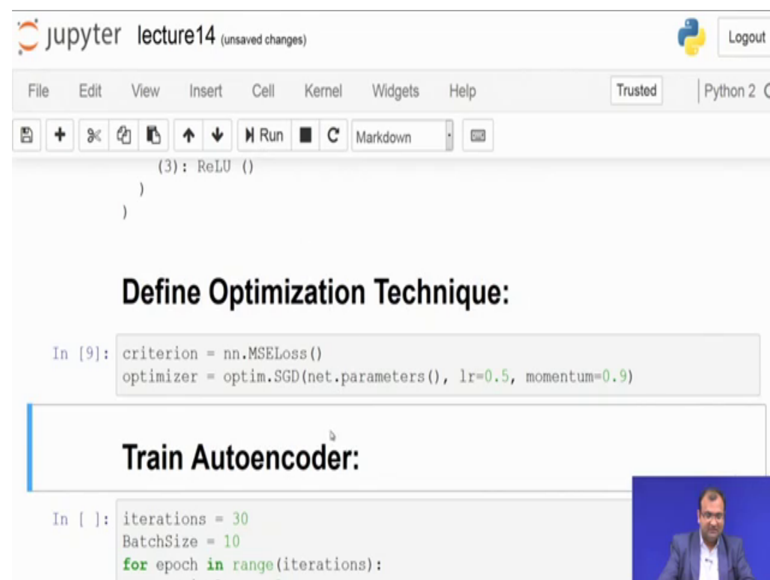
(Refer Slide Time: 10:19)



So, that is where it goes and you see my autoencoder is something which is of this form and it is pretty much defined. Now we go down with the same kind of a use of an

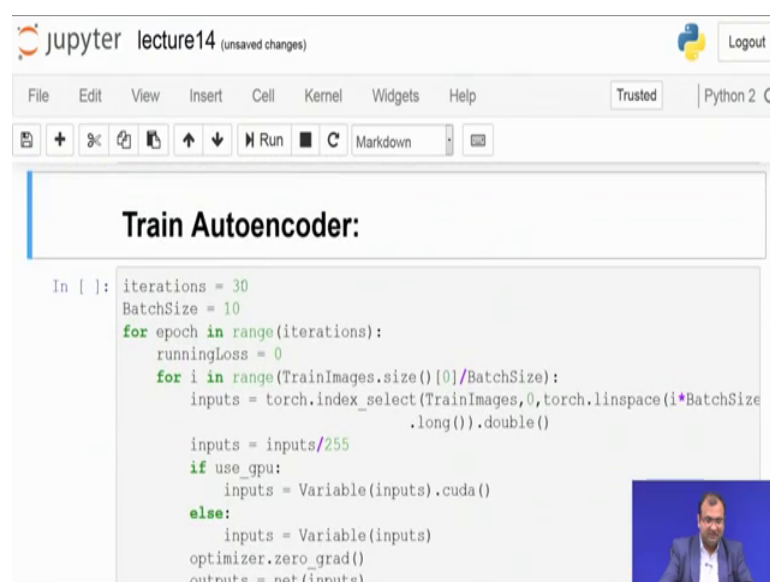
optimizer and here what I do is that since I am using it as an autoencoder mot which is sort of a regression learning problem. So, would be using my l 2 norm or a mean square error loss function over there and then for my optimizer, I am using my optimum package with a learning rate of 0.5 and a momentum of 0.9.

(Refer Slide Time: 10:28)



And this is a standard stochastic gradient descent operator which is being used. So, once that is done the next part is to train down my Autoencoder.

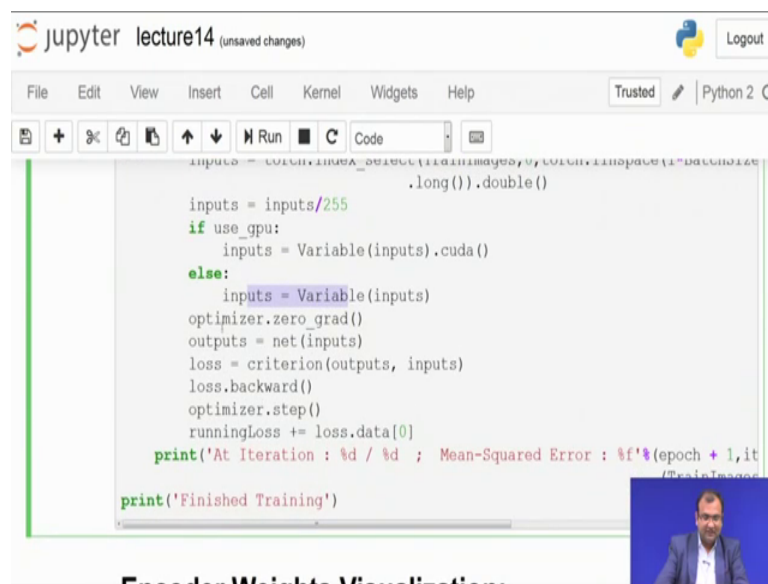
(Refer Slide Time: 10:54)



So, let us keep it short and simple. So, I will train it down just with over 10 iterations. Batch size as I had introduced batching in the earlier examples as well, to make it computationally much more attractive though I will be covering down the theory on batch sizes a bit later on when I, once I go over cost functions and eventually over to learning rules and batch sizes and how to handle them.

So, batch size over here is 10. So, I take down 10 images in a batch and then keep on running them. So, my first part is basically to get down my inputs and my inputs since it is in RGB color spaces that is a full scale of integer space. So, 0 to 255. Now the objective is to convert that 0 to 255 and bring it down to a range of 0 to 1 and floating point range. And that is what is achieved over here. Now once it is in a floating point value, we see if cuda is available then just convert it on to cuda and push it out otherwise these inputs are just defined as variables and left down as floating point variables.

(Refer Slide Time: 11:58)

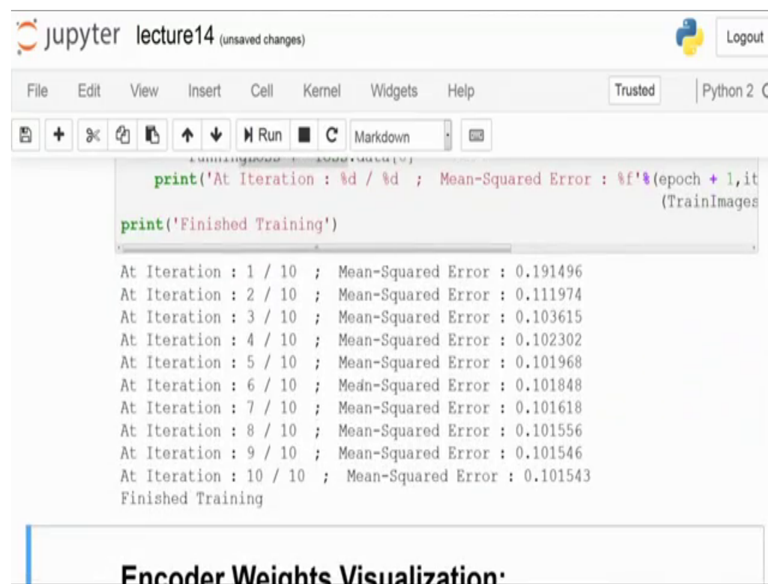


The next part is pretty simple. So, you have your optimizer 0 grad, which means all the gradients inside has zeroed down. Within your training function, you do a forward pass of the inputs over the network and get down your output.

You have the loss being computed by using your criterion function which was defined earlier and the loss is defined in terms of it is relationship of outputs and inputs and then you have the backward which is a gradient. So, that is a gradient computer part over the loss or the del del w or jw. And then you have your optimizer which is your update rule

of w of n plus 1 is equal to w of n minus of eta times of del del w of jw of n. So, that is with your optimizer dot step and here I am just finding out my running loss and eventually if I keep on running this one for my number of iterations, I will be getting down my training happening.

(Refer Slide Time: 12:55)



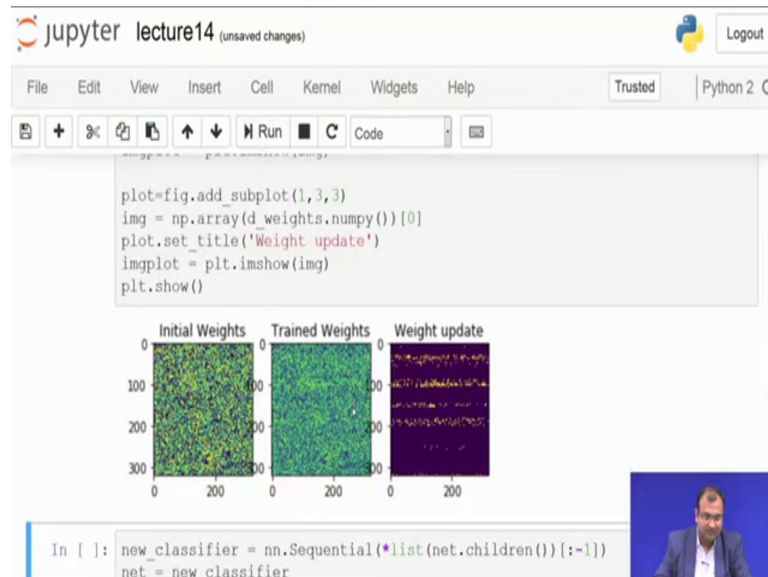
Now one thing which you can clearly see is that this finishes up pretty fast and in fact, so that does put me to the point then let us train it over 30 epochs and see what happens. So, you can see that for 30 epochs, it is not taking much of a time and the gradient is also converging, though it is converging down at a much slower pace itself. And so what happens typically in this kind of cases, since your number of samples is much lesser. So, the forward pass is taking lesser amount of time to compute.

Your number of batches which also come down because your batch size is now kept at 10 and so it makes it just 20 batches whereas, in the earlier case with MNIST and fashion MNIST, when your batch size was 1000, you had 60 such batches to do a pass and each batch was also quite computationally expensive because there was 1000 examples to forward propagate during each of this pass. So from there, we see that lesser number of data it does make it easier, but then the tweeting tendencies are also higher in these cases. So, we have those major discussions in the theory parts intermingled in between as well.

So, we will be touching down upon learning rule dynamics in a bit later on classes where I could be revisiting on these examples and showing you by changing down smaller

factors and getting down newer kind of rules, how you can make things converge much better. So from there, I enter into my earlier example as in to show down your weights.

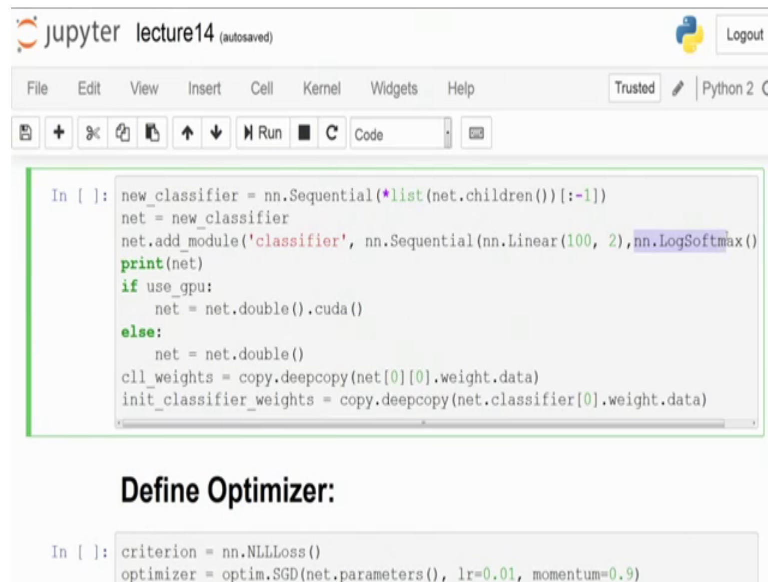
(Refer Slide Time: 14:20)



So, here if you clearly see, one is that you see your weights in colors now and that should be some sort of intriguing to you. Now the point why they come down is because whatever weights you had, they were just some sort of relationships between your color pixels to your next hidden layer over there. And that necessarily brings us to the point that my first layer of weights will have, 1 set of bits which are just associated with my red channel, 1 set of weights which are associated by my green channel and 1 set of weights which are associated with my blue channel. And all of them together go and create down your hidden layer over there.

Now, initially I had just had randomized weights over there. And then while we trained it over these 30 epochs where we had done. So, these weights change and this is the amount of change which you would be seeing down across all the different channels in terms of your weights. So, that is pretty much what is happening out over here.

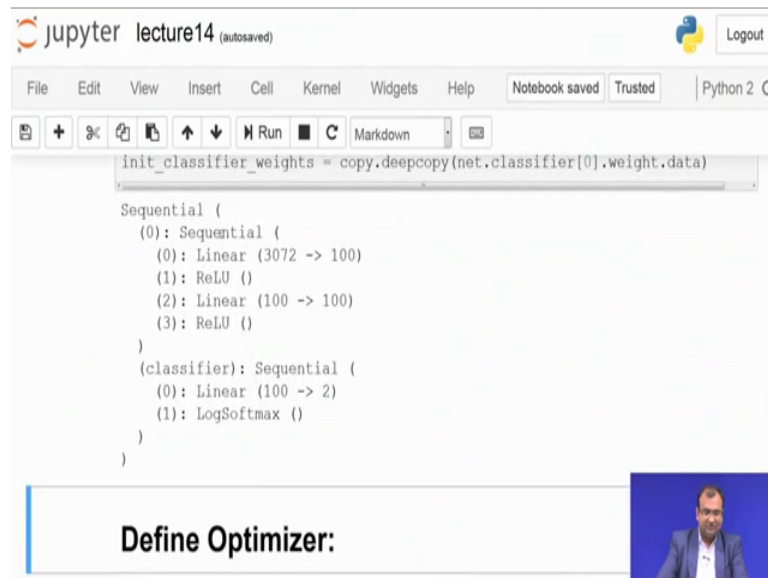
(Refer Slide Time: 15:19)



Now from there, let us get into the classifier part over there. So, as in with our earlier examples with MNIST and fashion MNIST, you have seen that for a classifier what we do is, you have your linear part of, the encoder part of your network gets preserved. And then beyond that you use your separate part of the network which just connects down a classification layer. So in the earlier case, we were connecting down a linear part which was connecting hidden layers and there we had just 100 neurons in the hidden layer. Here also we have 100 neurons in the second hidden layer.

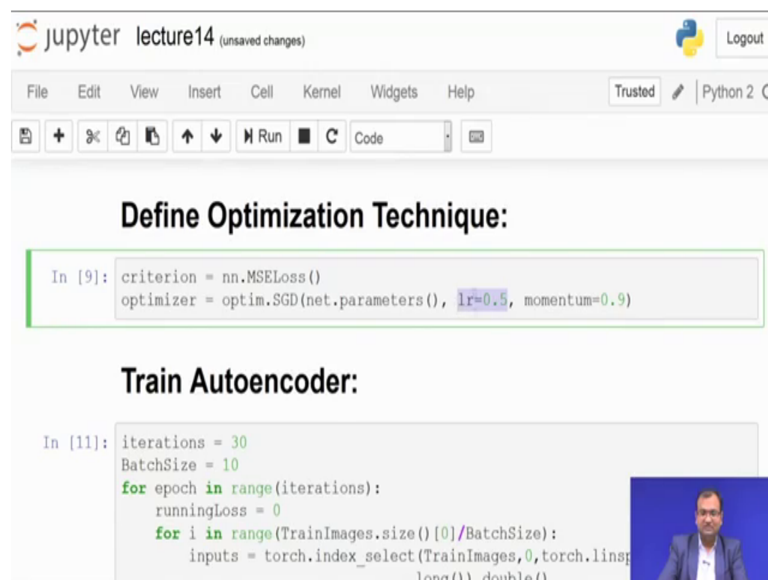
But there I had a 10 class classification problem. So, it was connecting it down to a 10 cross 1 unit of neurons on my output. Here I just have 2 classes to classify. So, it connect connects down to just 2 neurons on my output. And then my non-linearity is a LogSoftmax. Now that I have cuda available on my system, I can with the gpu availability, I can convert everything to a cuda tensor type and get down acceleration with the gpu. And I just decide to copy down my weights and keep them available for future use as well. So, these are just the initializations over there. So, as in you had seen this initialization because this was copied down before the training started down and this is the result after training. So, here also we do a similar kind of a thing. So, you see this is my network which gets defined over here.

(Refer Slide Time: 16:36)



So my initial part of the encoder that stays as it is. And then over here, I just have my final decision layer added down in terms of a 100 to 10 neurons mapping down.

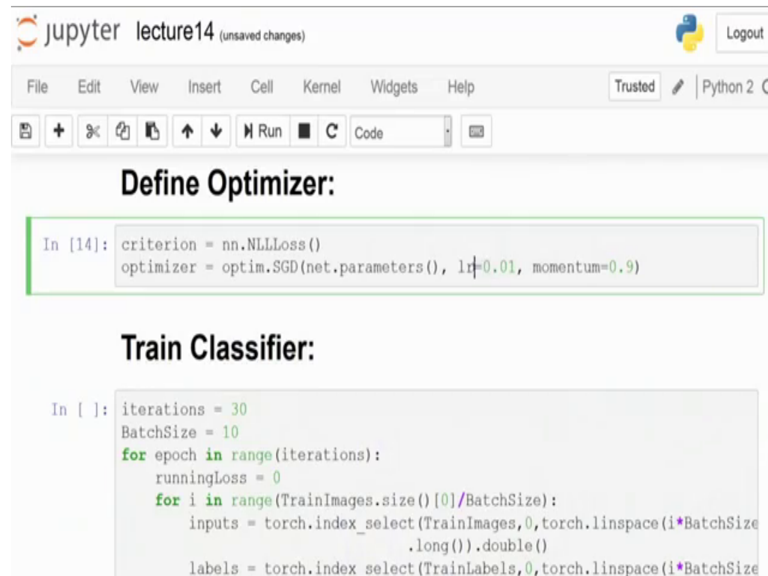
(Refer Slide Time: 16:52)



Now from there, I get into my optimizer and my criterion function over here being classification, it is defined as an NLL function, negative log likelihood as a function and my optimizer is still a stochastic gradient descent. Whereas, what I have changed is my learning rate changes down to 0.01. And that is again based on the dynamic range of the gradient coming down, because in the earlier case, my learning rate where I was doing

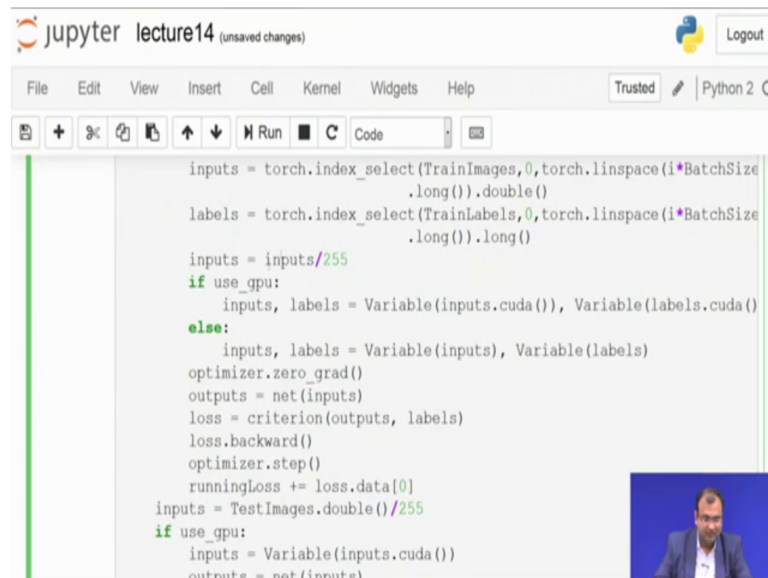
just an Autoencoder for mean square error functions was 0.5. And my momentum was 0.9. So, here that I am using NLL or negative log likelihood for classification, my cost, my gradient of the error which comes down that has a different dynamic range than the dynamic range of gradients which I was getting down when I was using a mean square error for my Autoencoder.

(Refer Slide Time: 17:42)



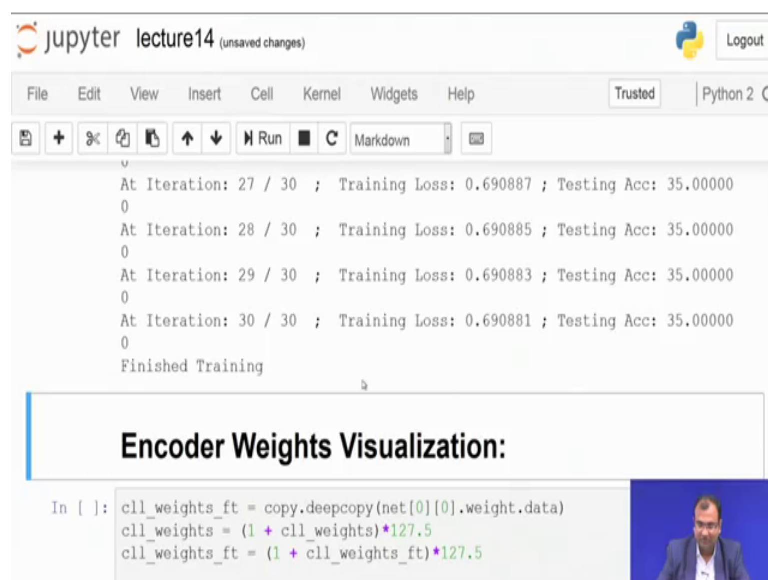
So, here my learning rate is appropriately suited because of the value of this being higher as compared to the weights. So, I put down my learning rate at 0.01. My momentum interestingly remains the same. So, I do not make much of a change to the momentum over there. So once this is done, we can go on training the classifier. So, here also I take 30 iterations and a similar kind of a batch size. The inputs are now resized.

(Refer Slide Time: 18:05)



And then, what I would be doing is, my loss is no more between my input and output, but it is now between my outputs and classification label because I am just trying to solve a classification problem. And here I have my gradient computation of the loss and then the update rule defined as well. So once that is done, you can pretty much see how blazingly fast it was working down.

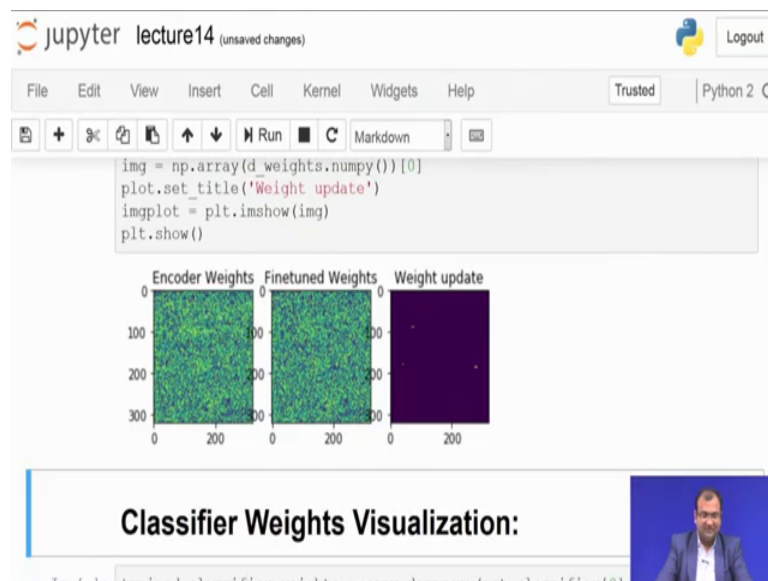
(Refer Slide Time: 18:32)



And accuracy is somewhere around 35 percent. Not a great accuracy I would say and there are sometimes varying like, sometimes it goes on to 78, sometimes come down to 65, 68.

So, this does mean that my network has not yet converged to be very sure. So, it is just twiddling down between all of them and every time you do; from my experiences, this data is actually much less like significantly less to train converge outer network, but for the sake of simplicity we are using this. And it is much more intriguing to train down color images for a deep neural network. So from there, we get into the Weight Visualization.

(Refer Slide Time: 19:15)



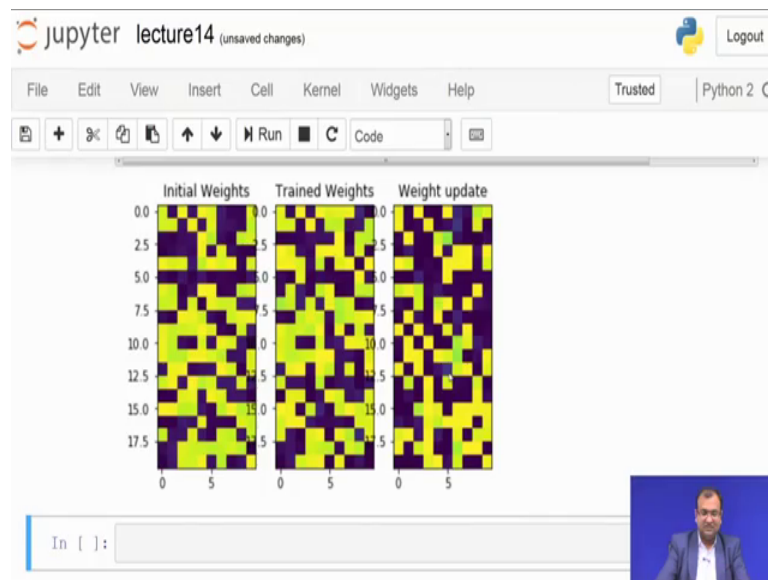
So, if you look into over here, but you can see is that the encoder itself when it was just auto encoding, it had actually learned down weights much better over these 30 iterations because the fine tuning also looks similar and when we look into the weight updates over there, then they are pretty much same what comes out.

So, that brings us to the point that there has not been much of a change as such when looking into them. So, one important fact which you can realize from this one is that, in the earlier cases for say MNIST and fashion MNIST, where you were seeing down a significant amount of change happening when your final fine tuning was being employed. One reason may be that, when you were training down initially, we trained it

for lesser number of epochs; then we have done it over here. So, there we had just trained it for 10 number of epochs. Here I am training down for 30 number of epochs.

So, the number of times the network sees these representations are much better in this case than it was in the earlier case. Whereas, the other reason may be that, here the number of representations which it has to learn is much lesser and much cohort. So, it can learn down these representations much more easier than it would be taking down amount of time to learn the same kind of representations on that kind of a varied data as MNIST. So if the representations are learnt out very good, then there are not much of a change which do appear on the weight update before and after the final classification rule as well.

(Refer Slide Time: 20:48)



Now here we try to look into the updates for the classifier. If you see into them, there has been a significant amount of change in comparison to what we had observed in the simple case of feature learning with just an Autoencoder.

So, in the earlier case on the first layer and the second layer, there was not much of a change coming down in terms of the weights whereas for classification, we did see a lot of change coming down in terms of the vision. And there is not any definite rule which you can lay down at this point of time though that is that is an active research area and problem, which the community is trying to address and that is about. Can you predict ahead of time for a given sort of a data, What will be the amount of maximum error or

error bounds which can come down for it, how many epochs will it take down to learn this system, then what will be the amount of change in weights which can come down; can there be some wise ways of initializing a network. So, later down the line we will be doing something called as a transfer learning problem where we take down networks which are already learnt.

So, the example is something like this that, if I want to classify these kind of say color images, then can I use a network which was trained on my MNIST data earlier over lot many more images, but on gray scale images; can I use them in some sort of an extension form for my color images. So, that they are interesting to actually explore because we as humans, when we look into it, we make use of our prior experience of having known something, having dealt with the problem and take it forward. Now these machines using deep neural networks which are supposed to be mathematically much more closer to the reasoning as we have within our biological neuron system, the biological neuro visual system, as supposed to match down one to the other.

So, can we have similar kind of a transfer of knowledge and information which we do as a human being on to this kind of network based models as well? So, that pretty much brings us to the conclusion of you trying to use down Autoencoders on patch wise models for doing it. So in the next class, is where I would be touching upon trying to use Autoencoders for even pixel to pixel classification or trying to solve a semantic segmentation problem.

So, that is based on one of our earlier work presented at a conference. And there we will also be making use of, trying to see if Autoencoders can actually be used as some sort of a hybrid feature descriptor, which is, instead of trying to put down a classification layer at the end of it with a LogSoftmax and just a neural classification layer; can we use the features extracted by the Autoencoder in the first few stages and then use another classifier, say a random forest or a support vector machine or some sort of a kernel discriminant analysis as well to do.

So, there what it would mean is that I can use the power of feature extraction from a lot a large corpus of unsupervised data and then match it down to another classifier, which can train with the smaller corpus of data for supervised classification. Or it may be as we see in the subsequent one where we are doing; it will be a class imbalance problem which we

will be dealing with. So, while neural networks are not so great for class imbalance which means that if one of the class is just 10 percent and other class is 90 percent, it often tends to drift itself to start replicate and copy down the 90 percent class. So, can we use another classifier to do it? So, there are multiple ways. In fact, neural networks with the specific kind of a cost function can be made to adapt itself to solve out these problems as well. So, as and when we go on, there will be multiple such tips and tricks and experiences from the field which we would be sharing with you. So with that, we come to an end to this particular lecture.

Thank you and stay tuned for the next one.