Introduction to Medical Imaging and Analysis Softwares
Professor Debdoot Sheet
Department of Electrical Engineering
Indian Institute of Technology Kharagpur
Module 3
Lecture No 15
Deep Learning for Medical Image Analysis (Contd)

So let us continue with from where we left on the earlier class on Deep Learning and I would now be discussing about basically a bit of historical perspectives and what we call as a Family Of Deep Learners over here. Now for that let us look into how deep learning came into its origin and how it was growing up. So now for that the very simple term is that this is not something we just came up in the recent past, it has been there for quite some time down.

(Refer Slide Time: 0:52)

Indian Institute of Technology Kharagpur | Department of Electrical Engineering

Deep Learning, origin and growth

- Around 1950 – NN age
  - Neural Nets (McCulloch and Pitts, 1943)
  - Unsupervised Learn. (Hebb, 1949)
  - Supervised Learn. (Rosenblatt, 1958)
  - Associative Memory (Palm, 1980; Hopfield, 1982)
- 1960
  - Discovery of visual sensory cells that respond to Edges (Hubel and Wiesel, 1962)
  - Feed Forward Multi Layer Perceptron (FF-MLP) (Ivakhnenko, 1968)
- 1980 – Neocognition
  - Convolution + WeightReplication + Subsampling (Fukushima, 1980)
  - Max Pooling
  - Back-propagation (Werbos, 1981; LeCunn, 1985, 1988)

Deep Neural Networks [Debdoot Sheet] 14

So we started with around the age of 1950s, when we had this neural networks come up with the initial days and that is when they started with actually McCulloch and Pitts model, so somewhere around 1943 from there we came up with Hebb rules in 1949 till Supervised Learning by Rosenblatt in 1958, from there to Associative Memory in 1980 and this Associative Memory concept is something which is much more ramped in used today and that is what plays down a role into how these neural networks can lean and also understand how to associate certain patterns and with what an outcome is or where it would be in terms of a deep neural network where it will be learning as to understand what an image for a particular class will look like.

So it is going to associate these image appearances to class labels itself. Now, in 1960s was when a lot of neurosciences and biological neurosciences were coming with mathematical discoveries and both the communities were benefiting each other from each other's work and that is where a lot of work came down on to how Feed Forward Multi Layer Perceptron happens and can we use them in order to understand how in our own visual perceptions how we perceive objects.

So do we it is something like does our eye look at everything as a camera censor or a CCD censor or a CMOS censor as in a camera or does it actually have some sort of a processing pipeline within the whole process. So does it sense instead of images of pixels, is it sensing a array of different directed ages and gradients and then is our brain reconstructing the way we look at the outer world together.

So this was a work which went down around 1960s and from there till 1980s we had this new term called as Neocognition and what that meant was a new sort of network started coming up which were called as Convolutional operations network. If you look at the network which we were looking in the earlier class so we just were connecting to all neurons to all neurons. Now instead of connecting every neuron to all the other neurons in the subsequent layer, we can define some sort of a convolutional operation. So say, I have a weight vector which can translate in sort of a sliding window fashion. So now on my first input node if I have say something 441 such nodes and I want to connect it to my second hidden layer to my first hidden layer which has about 400 nodes.

Now instead of how many connections I can draw down is basically 441 connections to each of these nodes. And similarly so this will form down a matrix, so 441 cross 400, which is the translation which goes down. Now see, instead of that I take down a small set of vectors and this is called as a convolutional kernel. So I have just a convolutional kernel of size 5 and say I can convolve this and pass it over the whole thing, so I can define the output of my first input nodes to the hidden layer in terms of a convolution.

Now the number of weights I will to look over here is actually much lesser, because it is no more all the connected weights over there but the number of weights is basically 5, which is left down. So the computations go down and again the beauty of convolution which is making it space invariant comes into over here. So in terms of understanding whether in image has say for our case if it is a image of a cell, so now whether the cell is present on the top left corner or the bottom right corner it is invariant to each of them because I will just be

convolving with these kinds of weights over the whole image, I will be able to find out the features of the image wherever the object be located over there.

Now these were the new concepts which are coming down around in 1980s and from there came down another interesting concept called as Max Pooling which is to reduce the complexity of networks and from there we came down to another interesting concept called as backpropogation, which is about how can we take the error back behind over there and do it. So this is what we were using in the first lecture on neural networks itself for training a neural network and coming down to what is the optimal weight combination.

(Refer Slide Time: 5:15)

Now sleek from there to 1980s from 1980s to 2000 is when a lot of research went into understanding simple low complexity problem solving with these kind of networks that led to the birth of Recurrent Neural Networks, then Long learning Feed forward Neural Networks, there was Advanced gradient descents, Sequential Network Construction which is how they can be constructed one by one at a time and from there to a very famous thing called as Auto-Encoder and from there a backpropogating CNN, which was a very fundamental discovery and in fact the exercise we do later on will have a backpropogating CNN itself for solving the same kind of a problem.

(Refer Slide Time: 5:49)

So sleek from there to 2000 and, onwards and this is where we land up beyond the era of deep beyond the era of neural networks standard neural networks and into the era of what we called as deep learning and where it started was with one of these kind of deep neural networks winning the NIPS 2003 challenge on Feature Selection and LeCun very famous model of the digit recognizer called as a LeNet and similar like what we are going to use later on for our problem as well.

From there to Deep Boltzmann Machines and Auto Encoders, so right from there around in 2006 onwards we saw the major boost in the way deep learning was being done, because if you look over here typically, these networks have a lot of connections between them. So the number of multiplications, the number of nonlinear operations you are going to do is actually seriously large, it is not a very small number, so you need a good amount of computing power available at your space to do it and save with a desktop computer it would never be possible, finitely possible, you would actually either be needing a lot of RAM space and in anyways your process power is always limited to one single thread or may be for certain machines which have hyper threading support on them you can have multiple threads up to 4 threads or 8 threads in total.

Now even with them the total time you would be taking is going to be finitely seriously large. Now with advent of these GPUs and what NVIDIA got down in the recent past is they upgraded their gaming cards with cores present over there using specific library called as CUDAs where you can use all of these courses as compute course for solving the purpose. So

we have specific libraries called as CUDNN libraries these days which help you in creating a neural network on the gaming card itself.

So you can just buy a simple gaming card for less than 10,000 Indian Rupees and use that as a super computer attachment on to your desktop computer and train down networks much faster. So we will be doing exercise later on in the day, where I am going to use a similar gaming quality graphics card as a GPU alternative and we are going to show basically how fast we can learn the whole network.

So from there till 2009 we got down this deep belief network on the GPU, which was a seminal contribution to Max-Pooling and CNNs on the GPU which required a significant amount of efforts in terms of a VLSI design constraints within how GPUs are designed. And in 2012 with Image Net winning the Image Net challenge from Alex Krizhevsky with the Alex Net is where it actually made the whole community come down to a conscientious that deep neural networks are what is going to rule down substantially the feel for a longer period of time now.

There is a sort of a framework, which can be used as very feasible solution to a lot of problems to we were facing, so instead of each of us trying to work on a different area and arguing across as to which is more important whether feature extraction is more important or classification is more important or who is at a much more superior phase. The whole community now comes to a conscientious that let us understand that it is a hierarchical problem, let us put the whole hierarchy into one single place and let us take one single model, which can learn the hierarchy and solve the problem and this is what we are going to speak.

(Refer Slide Time: 9:24)

Indian Institute of Technology Kharagpur | Department of Electrical Engineering

Families of Deep Learning

- Fully connected networks
  - Autoencoders
    - Autoencoders
    - Stacked Autoencoder
    - Sparse Autoencoder
    - Denoising Autoencoder
    - Convolutional Autoencoder
  - Belief Networks
    - Restricted Boltzmann Machines
    - Deep Belief Networks
- Convolutional Networks
  - Conv-Nets
  - LeNet, GoogLeNet, AlexNet
  - U-Nets
  - Res-Nets
- Recurrent Neural Networks
  - Long short-term memory (LSTM)

Deep Neural Networks [Debdoot Sheet] 17

Now, if you look into deep learning you basically have a family of learners over there. Now the first family is called as a fully connected network family and this is similar to a multilayer perceptron. So what happens is that each neuron on any layer is connected to every single neuron on the subsequent layer of the hierarchy. So the first model is an Autoencoder which learns to basically encode and decode, so to put in very simple terms it is basically you take an image patch you feed through this network at the output over there you should be able to reconstruct the same image file.

So if I have a 5 cross 5 patch on which I see an apple, on the output also I should be seeing the same apple coming down over there, this sort of a network is called as an autoencoder, it makes up of one part which is called as encoder and another part which is a decoder and since it is encoding itself that is where the name comes down. So then we have a stuff called as Stacked Autoencoder, and what does this is basically you stack down one by one autoencoders together and you create a much deeper network and that is where is gets its name.

Next one is a Sparse Autoencoder, where what you have is basically that we had seen that particular problem in the last class where we were trying to solve down pixel labeling problems on the Optical Coherence Tomography images in order to find out which pixel belongs to what kind of a tissue. And on the second layer over there we could see that most of the values were 0s, there were some values which were just 1s or - a very high negative value

over there. Now this kind of a Sparse Autoencoder is what is going to help you in finding out imposing that kind of a sparsity constraint over there.

Then there are Denoising Autoencoders which you can use as denoisers actually. So, if you want to remove noise from a certain kind of an image without you knowing that what is the distribution of noise or what is the basis function of the noise, you can actually learn them using a neural network and very easily remove. And in fact, you can remove out noise of very complex nature, so there can be Gaussian un-speckle noise, there can be Poisson and Multiplicative noise together coming down and you can de-noise these kinds of images by learning them appropriately.

From there is a Convolutional Autoencoder which is molt between convolution and neural networks and an autoencoder. So basically each layer of the autoencoder is a convolutional function itself, so that does not typically qualify as a fully connected network but again since it comes from the family of a autoencoder, we just keep it over here for the same purpose.

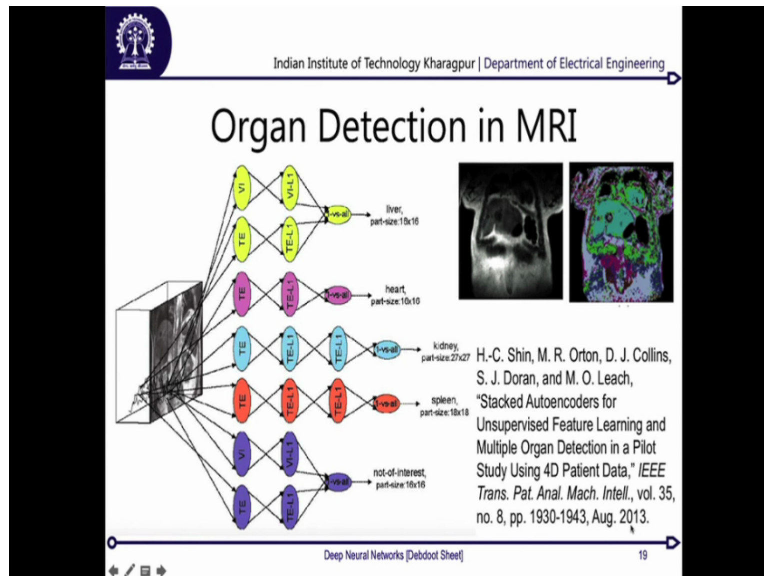
From there the second one on fully connected networks are Belief Networks, which consists of Restricted Boltzmann Machines and Deep Belief Networks. So a Deep Belief Network is basically sort of a stack version of the Restricted Boltzmann Machine itself and as a name suggest a Boltzmann, so this is what has a Boltzmann distribution imposed over there and you can have very good amount of memory associativity within these networks. Now you can read about details of them in much specialized literatures, but for our stuff we will just keep them to the minimum.

The next family is Convolutional Neural Networks. In Convolutional Neural Networks you can have generalized Conv-Nets or very specific models which have come out in the recent past so some of them are called as LeNet, GoogLeNet and AlexNet. So LeNet is again from the (()) (12:25) challenge with Yann LeCun was solving in 1998, and so they have a much improvised version on top of that from Google and that is called as a GoogLeNet and the ImageNet Challenge being solved by Alex Krizhevsky, what is called as the AlexNet. Then there are U-Nets and Res-Nets, which are basically networks where you can put a whole image and get down the segmentation of the whole image instead of doing it as a pixel by pixel manner and every single layer over there is implemented as convolution functions.

From there we have Recurrent Neural Networks and one very famous component of that is Long short-term memory, which can try to encode temporal patterns itself within a network

so if you are looking at say temporal data which is say there are scans of cancer patients every 6 months on of the same portion, so if want to relate between the temporal side of it the same voxel at the same space at the same patient but every six months and you want to do a follow up, so you would be using some sort of a temporal mapping and that is where you would be making use of LSTM networks.

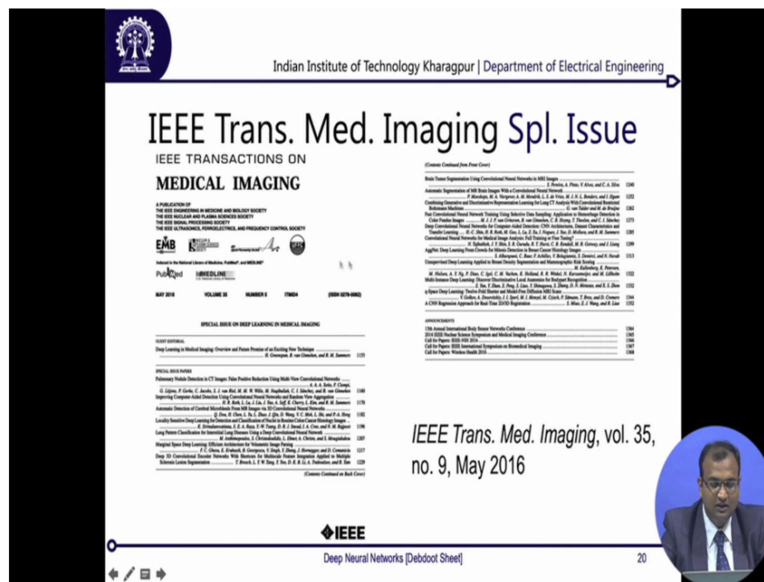
(Refer Slide Time: 13:49)



So let us get into a few of these places where deep learning is playing a big role today for medical image analysis and that is what is the whole agenda of our discussion is, the first one I would like to point out is this particular paper from PAME in 2013 on one of the special issues on deep learning and this was about organ detection in MRI and very clear case in which the idea was that can each pixel over an MR sequence, so if you have one of the slices in say a T1 sequence or T2MR sequence, then can I label each single pixel as to it belongs to which particular kind of a tissue or which particular kind of an organ.

So there were manually annotated examples which they were using and the solution was not a very computationally complex solution, so it was just a very simple Stacked Autoencoder, which was doing unsupervised feature learning and finally at the final stage it was mapping them down on to which particular organ it belongs to. So this is a very seminal contribution which came up and the way this whole learning was done without having to handcraft and hand engineer features were seriously wonderful contribution at that point of time.

(Refer Slide Time: 14:47)



There was IEEE transactions on medical Image Special Issue, which came out in May 2016, which speaks about all on deep neural networks. So this was just a special issue which had all papers were deep neural networks were being used to solve medical image analysis problems in total. So that ranged from microscopy to MR to brain to histologist to robotic surgeries everything over there.

So it is a good collection of papers which I would definitely add as a pointer for you to go through if you are trying to make sense of where and how people have been using these kind of techniques in medical image analysis itself. Now few snapshots from the issues are where on brain tumor segmentation using convolutional neural networks, then on using deep 3D convolutional encoder networks which shortcuts for multi scale feature integration for Multiple Sclerosis Detection and Lesion Segmentation in brain MRs.

(Refer Slide Time: 15:52)

Indian Institute of Technology Kharagpur | Department of Electrical Engineering

Some Snapshots from the Issue

S. Pereira, A. Pinto, V. Alves, and C. A. Silva, "Brain Tumor Segmentation Using Convolutional Neural Networks in MRI Images", *IEEE TMI*, vol. 35, no. 5, May 2016.

T. Brosch, et al., "Deep 3D Convolutional Encoder Networks With Shortcuts for Multiscale Feature Integration Applied to Multiple Sclerosis Lesion Segmentation", *IEEE TMI*, vol. 35, no. 5, May 2016.

Q. Dou, et al., "Automatic Detection of Cerebral Microbleeds From MR Images via 3D Convolutional Neural Networks", *IEEE TMI*, vol. 35, no. 5, May 2016.

Deep Neural Networks [Debdoot Sheet] 21

And then there is another one on Automatic Detection of Cerebral Microbleeds from MR images using 3D convolutional neural networks. So this is where you are no more limited to images or 2D, but since the data is inherently 3D in the voxel space so we are going to use all of the voxel information and design a neural network which is also operating on the voxel space.

So earlier if we had a small patch of 5 cross 5, then I just have 25 pixels and my convolution operator may be another smaller one which is operating only on the special scale. Over here I will be having now instead of 2D scale I will be having a 3D volume, so I will have a 5 cross 5 cross 5 voxel space, which is 125 voxels. So now I can define a very small convolution operator say may be 2 cross 2 cross 2, which is just 8 elements over there which can move in the voxel space and do convolutions over there.

So this is what brings in the beauty and some novel changes which the community itself had to do. If you look at computer vision community as such that is not where we operate on 3D data day in and day out, its majorly 2D data which has a time access to it, but over here it is majorly 3D data. Although at some certain point of time when you are trying to do a dynamic MRI kind of stuff you would also be getting a 4D data, where it is 3D data over time so you have 4 dimensions.

(Refer Slide Time: 17:24)

So that is where you would bring down much more noble contributions as to how can these modeling be done, may be with 4D neural networks or something even beyond that and much more radical and interesting. So from there I would point out to one of the earlier work which we had presented sometime early in 2016 and this was about concept called as Distribution Preserving Autoencoder. Now the beauty over here is when we were telling about this autoencoders and the concept was that I wanted to always preserve the same patch so whatever goes in my input the same has to be created on my output and that is what I would like to look at it.

But if you look at speckle images, then you would not necessarily have a one to one mapping, because speckles are necessarily especially non stationery. So they do not remain stationery in space when you are taking it over multiple points of time. So but however, the only thing which is constant for them is the distribution they come down from the same sort of a distribution.

So what we wanted to do was can we create a network which can learn to encode in a way such that the output has the same distribution as the input. So this is what we were doing with Distribution Preserving Autoencoders which paved a very great way in solving a challenge about Plaque Detection in Optical Coherence Tomography Images. So we could very effectively deal in plaques and this is one of the 3D visualizations.

So we have this whole model running on a frame by frame basis over a whole pool back of Cardiovascular Optical Coherence Tomography images and which we find out this plaques

and then how the whole 3D topography is mapped down over there. So you can have a careful look over there as well and this is a model which is pretty simple to implement and does not take much of compute time requirements during testing, however the cost functions and everything is definitely something where we had to put in lot of way in which how the data transfer happens and how the networks (()) (19:01).

So what I wanted to basically say through these ones was that it is not necessary that it will have a one shot single go implementation where you can just take the model and do it, a lot of times you will have to innovate yourself, you will have to create your own networks, own training functions, cost measures as we had done over here. So instead of taking a mean squared error or sum of squared errors over there we thought of taking down a diversions measure so that we mapped down distributions between each of them.

(Refer Slide Time: 19:50)

Indian Institute of Technology Kharagpur | Department of Electrical Engineering

Endnote (almost there)

"It's like in quantum physics at the beginning of the 20th century" *Trishul Chilimbi* (MSR, DNN, Adam)

"The experimentalists and practitioners were ahead of the theoreticians. They couldn't explain the results. We appear to be at a similar stage with DNNs. We're realizing the power and the capabilities, but we still don't understand the fundamentals of exactly how they work."

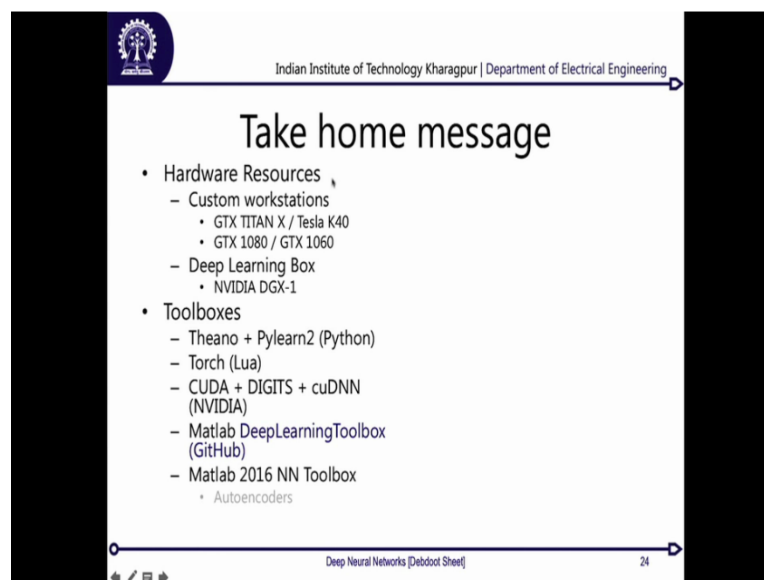
Deep Neural Networks [Debdoot Sheet] 23

So these are new innovations which come down when you are trying to use neural networks and specifically deep neural networks for the area of medical image analysis. Now, with this we are almost at the end for deep neural networks, although it is a very interesting topic and it would have been fun to devote many more hours on to this one which itself is a much more specialized topic but I am just over here to give you an introduction and some user case scenarios where they are working. So here when we come down to an end I have this famous code by Trishul Chilimbi and he says that this deep learning thing is quite like quantum physics was at the beginning of the 20th century.

And I just leave it up to you to figure out why does he says something like this and now the reason why he is saying it is because if you look at quantum physics at it was at the beginning of 20th century, then there were lot of experimentalists and practitioners who were ahead of it and who were ahead of the theoreticians. So it took a lot of time for theoretical physicist to say and explain each and every single stuff about quantum mechanics whereas, experimental physicists were observing it much faster. So today with deep learning it is the same thing, when we look into the theory of deep learning it is really complex to understand and explain over there and without those explanations possibly it is really hard for you to even make a way through in terms of publication or even a single product development over there because until you are able to explain the theory, there is no surety why it is working the way it is working over there.

Whereas in practice we do see a lot of things just what like magic and as a black box and this is a point where I would definitely leave a thought for you that do not just trust it as a black box, try on innovating over there and please make a sense of understanding why your method is working the way it is supposed to be working or why it is not working the way it was supposed to be working which you had decide.

(Refer Slide Time: 21:32)



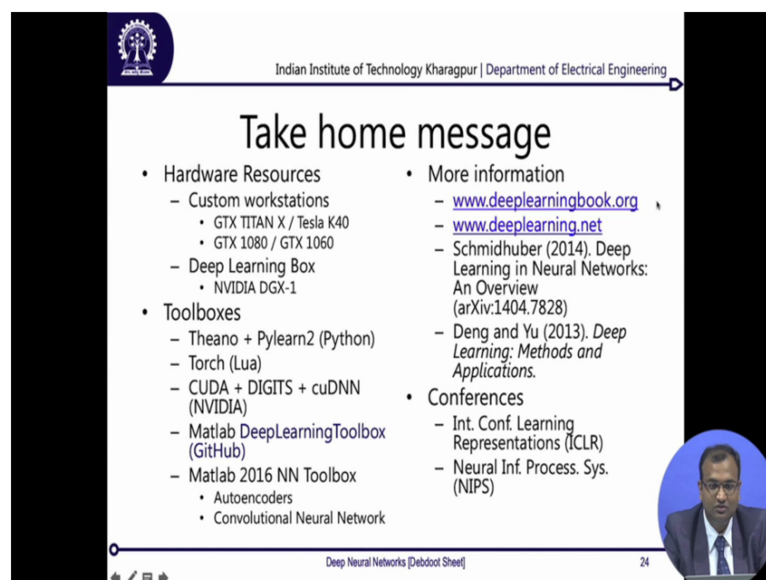
Now that takes us to the end of it and I just have a take home message over there. So as far as working with deep learning and if you want to do some serious amount of work then you would possibly need a custom design workstation which is with a GPU and although I am going to demonstrate due to the very simple gaming grade card but it generally suggested that for most of a serious experiments we do use a GTX TITAN X of card or Tesla K40 and so

there is a M40 which is also over here now and you have the P100 GPUs coming out or you can use much more affordable version of them on a GTX1080 or a GTX 1060 as well.

And if you can really afford then DGX-1 is a Deep Learning Box a single box solution which is provided by NVIDIA and it is great for doing it. On the toolboxes side of it you can use Theano and Pylearn2 within Python for deep neural networks, on a separate language called as Lua you can use the library called as Torch. For NVIDIA things you would be requiring CUDA, DIGITS and cuDNN, infact like if you want to call down the GPU through any of these either from Python or Lua, you will still be requiring cuDNN and CUDA.

If you are not a much fan about these large object oriented programming or complex languages, you can still choose to stay with your very homely comfortable environment of Matlab. So just git hub for the deep learning toolbox and you will be getting down a very sweet link coming down over there which you can use for learning deep learning and in fact Matlab 2016 onwards within neural network toolbox, you also have autoencoders and CNNs as a standard package and feature available, so do feel free to visit them and make use of them

(Refer Slide Time: 23:21)

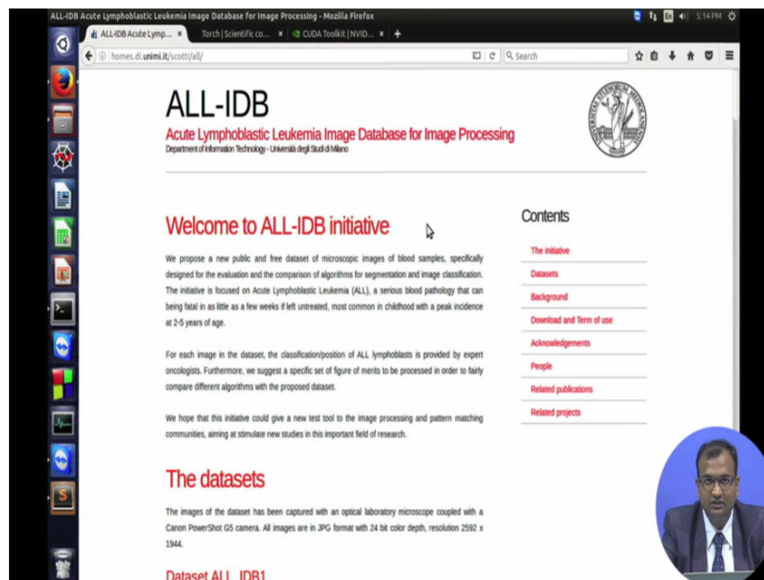


And for reading more about deep learning, there is a website where we have a eBook for deep learning available now. So that is written by the major players on the field you can visit deeplearning.net, which is a major website where we socialize and all new postings in the field of deep learning come up and beyond that I would suggest that you read this, you can

have a good read over this overview paper, which provides most of the material which we have been using for these lectures.

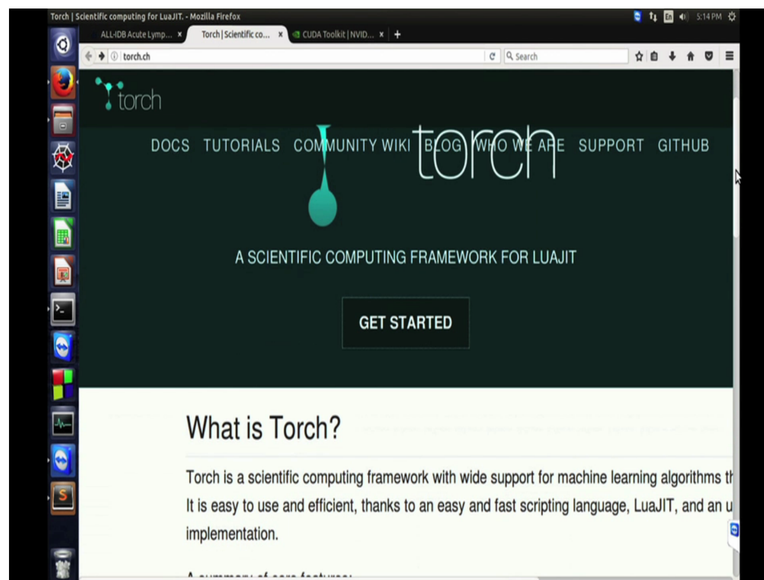
And you can also have a look through this particular book on deep learning methods and applications. And for looking at Conferences ICLR and NIPS are the places where majority of stuff related to deep learning are published. So with this I come to an end of the theory class and we would have a small demonstration with using with solving the same ALL problems on using Torch on Lua, for our deep neural network implementation.

(Refer Slide Time: 24:13)



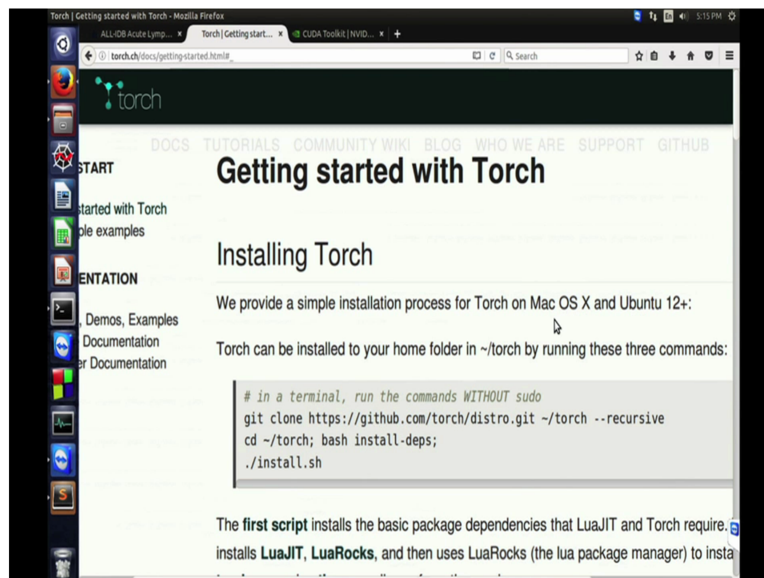
So we get back to the practical demonstration of using a deep neural network and we will be using an architecture called as the LeNet in order to solve this problem. So what we would be doing is we are again going to use the same ALL-IDB images, which we had downloaded in the last example as well so that I can explain you the whole concept over here. Now on top of that our deep learning purpose will be solved using Torch, which is a scientific computing framework which is based on Lua just in time compiler environment.

(Refer Slide Time: 24:38)



So you can just go down to the website called as torch.ch and you would be ending up on this page, only reservation over here is that this is available for Linux based systems and Unix based systems it is very easily combined. So you might need to have a Linux based system available over there.

(Refer Slide Time: 25:03)

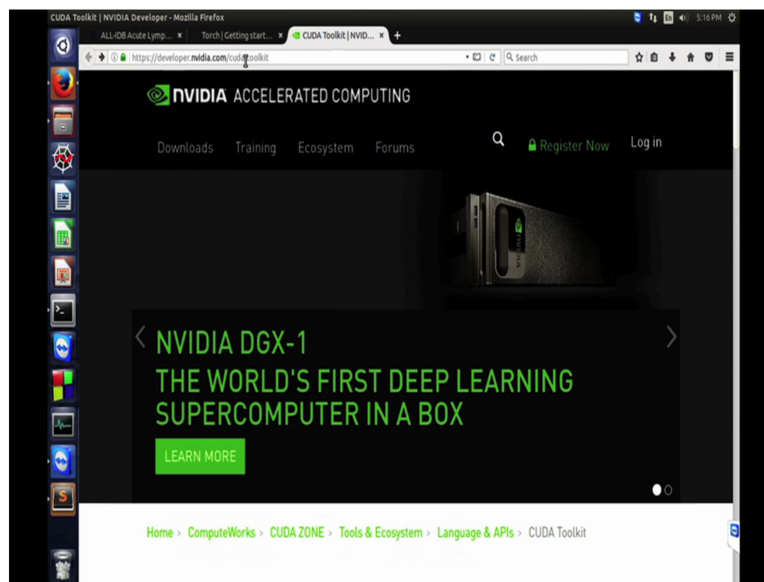


Now they provide a very simple step by step (()) (25:02) if you want to install torch on OS X platform from Mac on Ubuntu as well. What you can do is you can follow down each of these instructions over here and just keep on doing, this would be downloading about 2 Gigs of data through your network, so you need to be atleast available with that much of

bandwidth from your side and once you are able to finish till this bashrc thing and just has a word this would take you some amount of time.

So typically this git clone and the recursive cloning over there and then the installation this takes you roughly 30 minutes based on what kind of a bandwidth speed you are affording over there and then the whole installation completes. Now, once you are over here what you can do is after that if your system has a GPU, which I would strongly recommend is how we are also going to do over here, then you would need access to the NVIDIA CUDA toolkit in order to use your GPU and make it faster. So, today the CUDA standard is CUDA 8.

(Refer Slide Time: 26:12)



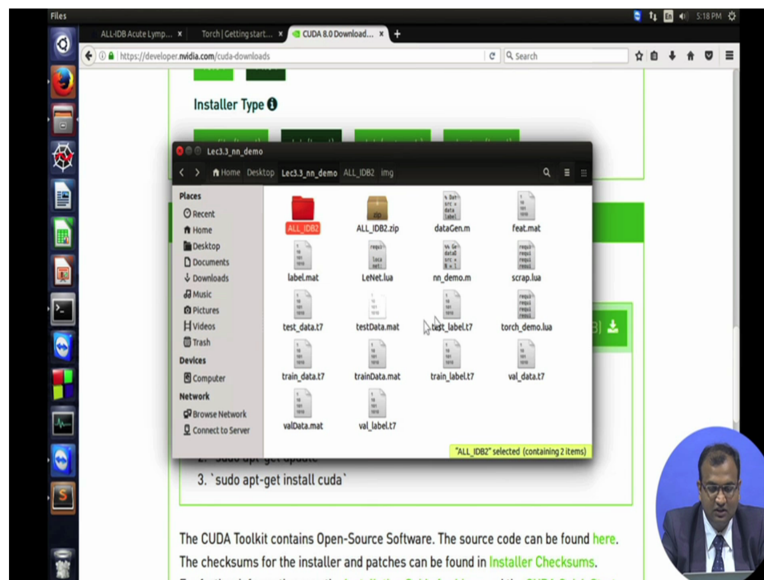
So you can just Google down for NVIDIA CUDA toolkit or just go on developer.nvidia.com/cudatoolkit and you would be getting access to that. So, on the CUDA 8 toolkit you can just press on download over here. Now this goes down to another page on which it would ask you basically a few set of questions you can select your system say Linux and then it would be asking you another set of question as to what is your processer architecture for most of us it would be x86 or a 64 system. So I click on this and then it asks me another question as to what sort of OS I am using, most likely we are going to use Ubuntu atleast for my system it is that.

I prefer using the 14.04 because of a lot of support issues around with software pipelines as well. So I have this 14.04 and I also prefer using the debian distribution file on a local one instead of a run-file or any of them. So just click on this and you would be directed to a download link, so now you can download this 1.9 Gigs over here and once the download is

complete you can just do a dpkg and load this one on your apt-get repository and from app-get now you can basically install CUDA.

So this is also going to take you so apart from this 1.9 GB during the update process it would be taking about couple of 100s of MBs as well on the download. So and it takes about roughly 2 hours of a time for getting Torch till CUDA and everything downloaded. Now beyond that you can also install CUDNN if you would like to use CUDNN features as well. Now that is something you can just Google and find out more of details as well.

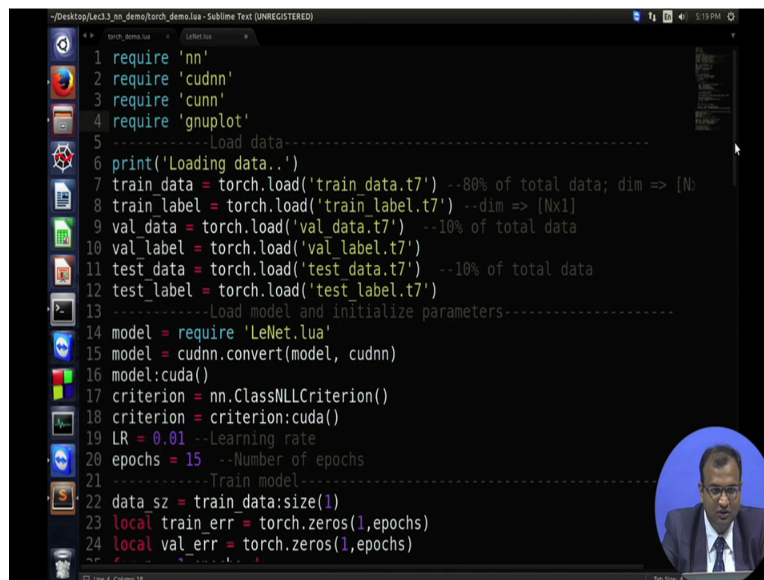
(Refer Slide Time: 28:08)



Now, let us get down into the actual part of the stuff, so what we have do is we had our dataset over here, you had seen these images over there. Now all of these images we have actually packed them onto some sort of files which are called as dot t7, which is a standard extension for Torch version 7. So we will be loading down our preform data, so it generally takes a longer amount of time to load each image at a time, so what we have done is we have transferred that into some sort of a matrix and stored that as a data file, I will just load one file and I will get an array of all the matrixes, in Torch we called it as a tenser.

So we just had a 4D tenser in which my first dimension is basically the image number which I am going to get, the second dimension is each of these color plains, the third dimension is the rows and the fourth dimension is the columns. This is how Torch natively handles it out and (()) (28:49). So we will be providing you with small Matlab codes on how to create this your t7 files as well and I am not discussing it any further now.

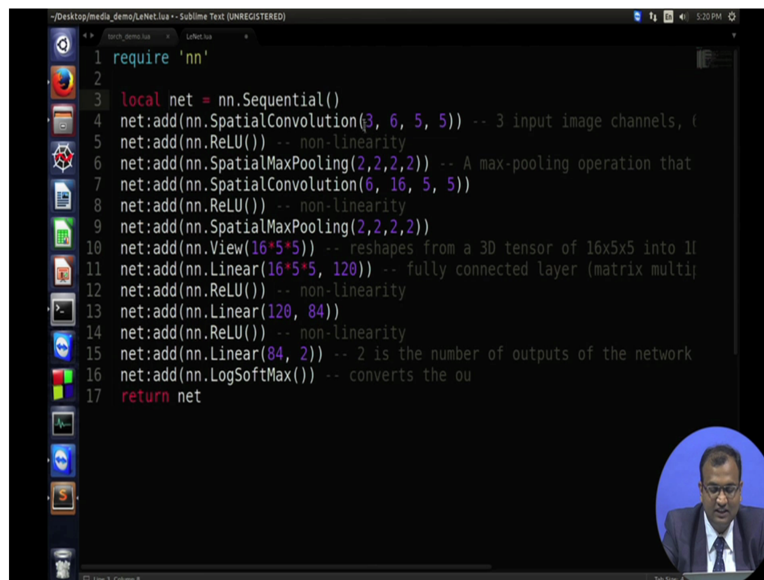
(Refer Slide Time: 29:03)



Now from there let us move into our network over here, so this is a standard code on Lua, how it would be looking like. So you have certain kind of files which you would need to call, so we are using a CUDNN library as well. So I am calling down NN, which is for defining neural networks and then I do require for CUDNN because I want acceleration for this CUDA and with the CUDNN library itself. And gnuplot is for basically I want to plot those fancy epoch versus error reads one. Now this is a place where I am just loading out all the data, so I have my training set, I have a validation set and I have a test set. This training set is where I take all the training data on which my errors would be computed and I would be doing the backpropogation.

I take a validation set in which what I would do is basically that is a set which is not used in the training but every epoch I push this set in order to find out what is my error and I can use that error in order to find down to a closing point or where I need to stop training any further. And test dataset is what I would be using subsequently after that. Now from there what I do is I have my network defined in the separate file which, is called as LeNet.Lua, so this is where my whole network is defined.

(Refer Slide Time: 30:18)



So what I have is special convolutions over there, so I have 3 channels which are my 3 color channels mapped onto 6 channels over there. So there are basically 6 convolutional operators which I am going to define over here which takes in 3 dimensional input and does it and each of them is a 5 cross 5 operator. So I have a 5 cross 5 cross 3 sized convolution kernel and 6 of such kernels being defined over here.

Then I do a nonlinear transformation in terms of ReLu, which is called as rectified linear units the only difference it has is basically if a input to the ReLu is negative below 0, then it climbs it down to 0, if it is positive it just keeps the same positive value passing, this is also another sort of non-linearity. Then we do a Max Pooling over there over a 2 cross 2 cross 2 cross 2, which is which means that I took a take a 2 cross 2 area and then I shift push it down onto one area.

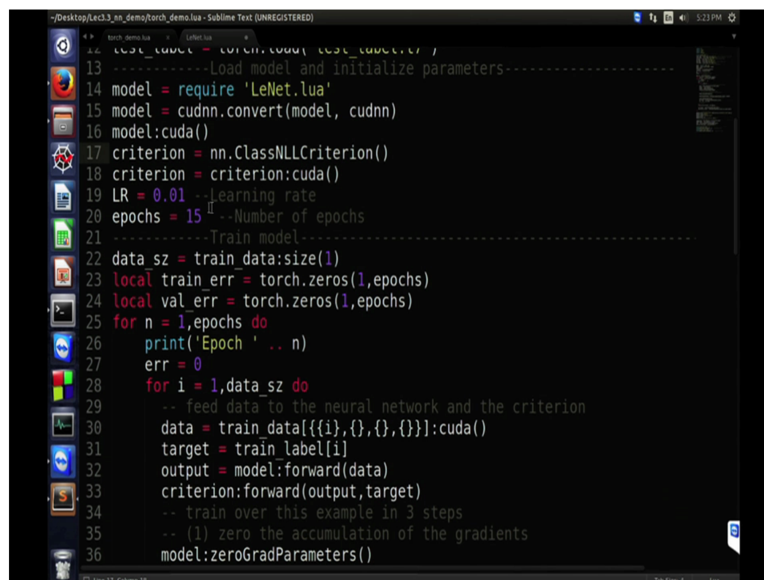
So 2 cross 2 basically 4 pixels, whatever is the maximum value in this 2 cross 2 element, this is what stored into 1 single pixel or a region over there. And then I make this one move by a factor of 2 cross 2, such that I have non-overlapping Max Poolings coming down. So on this output over there I will again do a special convolution, so what this does is all the 6 channels because I had 6 channels output from here, so I will be doing a 5 cross 5 cross 6 sized kernel convolution and there are 16 of such kernels which I am going to learn over here.

So similarly I do all of those operations from Max Pooling and everything and here finally what I have is, since I have 16 layers over there and this 5 cross 5 is again dependent on what was my initial size of the image and how it comes down. So what we find is basically there

are 16 cross 5 cross 5 such, so it is basically what comes down and the end over here is 16 such 5 cross 5 patches. And I will just want to linearize all of those neurons into one single layer.

Now after having done this through view, I have a linear connecter between this number of neurons to 120, so I mapped down to 120 hidden layers over there via a ReLu. And there is a fully connected no more convolutional operations over here and then from those 120 to 84 and from there again to 2 and then I do a log of SoftMax as a nonlinear transformation over there and this is what defines my LeNet architecture totally.

(Refer Slide Time: 33:14)



So this is what I load over here, you can actually copy paste the whole code over here and define it and do it, but we do it just to keep the training part separate and the model part separate and it helps us in much better book keeping. So now what I do is I covert this model to CUDNN and type cast this as CUDA, so that I have all the operations available as cudn module and on CUDA memory space as well, I define criteria which is my loss and this is a negative lock likely hood criteria, this is my classification error how do I define it.

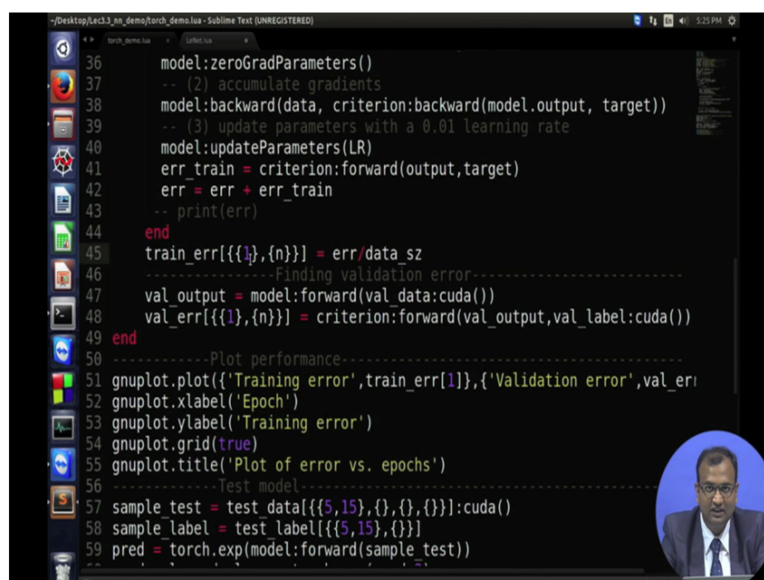
And then what I do is I also need to evaluate this criterion on the GPU itself, so that is why I am going to type cast it as CUDA so that my parameter space exists on the GPU memory itself. Then what I do is I set a learning rate over there, which is 10 power - 2 and total number of epochs over which I am going to train. Now putting a very smaller value over here as 10, now what I do is I take my dataset and just look into how many samples are present over there.

So use the size operator to look into how many samples, then I define 2 local variables which are just training error and validation error and eventually what I do is, over 1 to total number of epochs I am just going to iteratively process this one. So for the first epoch what it would do is initially it would take since it has multiple numbers of images over there, so it is going to take one image at a time and then compute and then reiterate this whole process the way which we are learning over there.

So it takes in the data then it does a feed forward through the whole thing and evaluates what is the error which is what happens over here. So you do a forward of the model so you have certain weights over there you put in the data you are going to get down certain output. Now what you do is you take this output and you know what is the target, so you can do a forward on the criteria and find out what is the amount of error which comes down over here.

Now what you do is you zero down all the gradients which were earlier computed and then you do a backpropogation through this part over there. So what you do is you backpropogate and find out what is the total derivative across the error and you also backpropogate this one across the model cumulatively. And then you update the parameters of the model using this particular learning rate over there. Now after this has been done you can find out what is your total training error over there and this is where it keeps on cumulating and coming up.

(Refer Slide Time: 35:16)



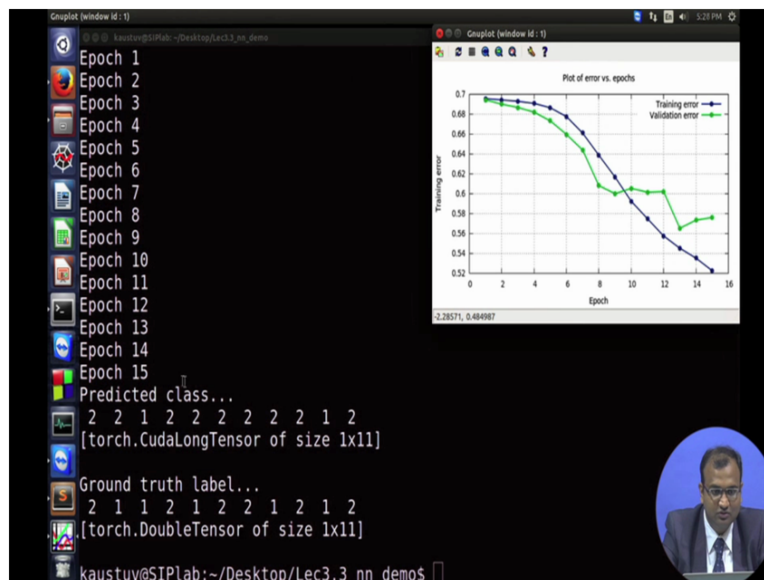
Next what we do is we move a bit below and we find out what is the total amount of training error by just taking an average over the total number of training data over there. On this side of it we are also going to find out what is my total error in validation as well. So I use the

validation set over there, do a forward in the same way and then find out. But this is no more being used in order to backpropogate so I do not have an update mechanism over there. Now once that is done, I will be plotting this is a small module for which gnuplot in order to plot the errors over there, then over here I am going to test it with unknown data. So I am just going to type cast them onto CUDA and then just do a forward of the model.

Now once I am able to do a forward of the model I am going to get down my predictions. Note that since it was a LogSoftMax, so I needed to get the exponentiation of the whole thing that is why I am doing an exponential over it so that I get in the range of 0 to 1. And once that whole thing is done, now I am going to get down a predicted values and predicted class and this is what I am going to print over there.

So let us jump into how we run it down, so typically for running this kind of a code you will be so I have already routed into the folder where my codes are located and my data. So this is where my code and my data is located in terms of this t7 files such that it can load it very easily, over there I will just write down small command called as th which is a function for calling down the Torch compiler itself and then say that this is the file which the compiler has to execute.

(Refer Slide Time: 36:48)



Now once I do that I press an enter so that starts with executing you can see that it is loading the data and then it is able to process out all the epochs over there and on the validation side over there you can see that these are the predicted classes. So the predicted classes since we had put down some 10 samples over there in order to see how the predictions were. So the

first one got predicted properly, the second one there was an error in prediction, the third one was correct, forth one was correct, fifth there is again an error, sixth it is correct, seventh it is correct, eighth correct, eighth there is an error, ninth is correct, tenth is correct and eleventh is correct.

So for eleven of these classes we could see all of them and this is a plot which I got from my gnuplot about the total validation. Now if you see that this curve is actually still falling down, it has not yet saturated, so maybe I can put it down for longer number of epochs maybe 50 epochs, or 100, or 1000 epochs a lot of time. And then if it comes down to a much lower error because the total error over here is still led in the range of 0.5, so we were typically prefer this to be in somewhere in the range of 10 power of - 2 and that is where we expect that a lot these predictions would be correct. So you can just have a play around with these ones when increasing the epochs as well and till then thank you.