

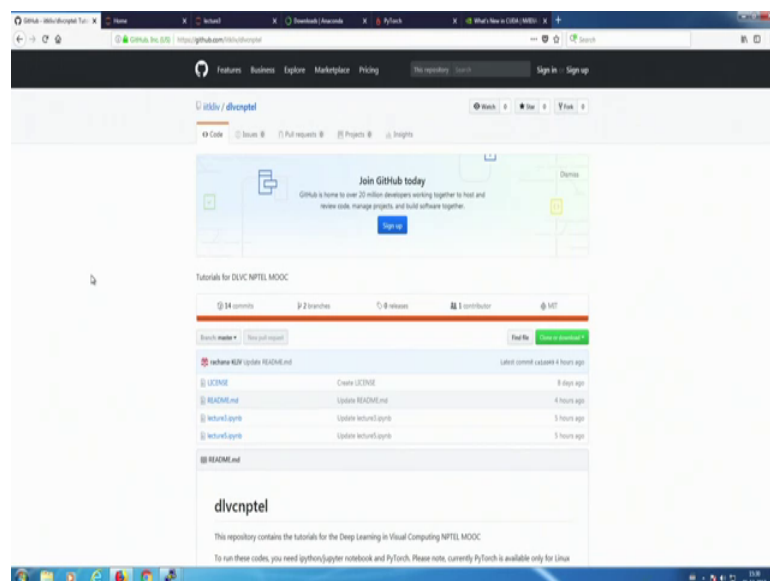
Deep Learning for Visual Computing
Prof. Debdoot Sheet
Department of Electrical Engineering
Indian Institute of Technology, Kharagpur

Lecture - 03
Feature Extraction with Python

So welcome. And while in the last two lectures you have been introduced to various aspects of what is this course about, and just a brief overview and visual computing from the classical way. So, now what we are going to do is while you have learned down about the classical techniques of how to extract out features and then use those features in order to classify. So, the first part is basically getting down your first bits of the code study in order to extract out features.

So, what we will do today is actually a lab session, it is a hands on mechanism and I would go through down the process of how to set up your whole system a pc on your own side, which you can use for doing these basic exercises. And since our codes are going to run down on python and that is the predominant modality on which we work and for deep learning purposes we will be making use of PyTorch, which is a very specific deep learning library within python, so without much of a delay let us get started.

(Refer Slide Time: 01:17)



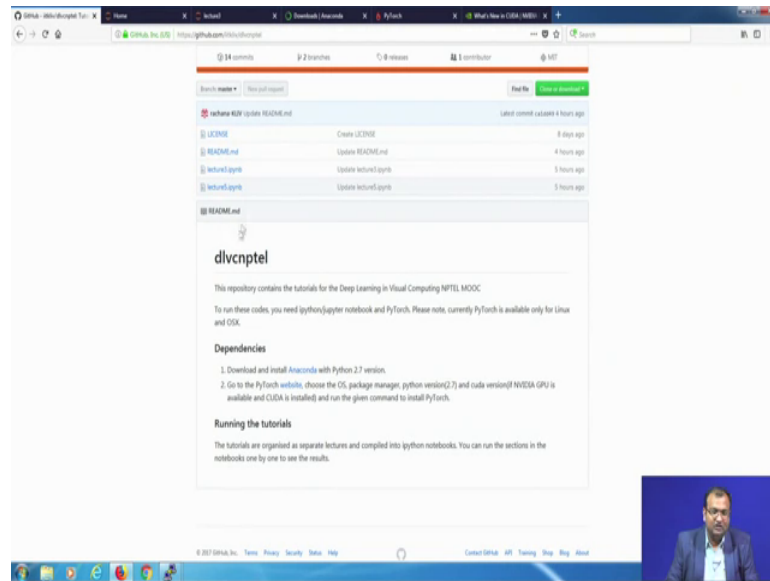
So, we have put down everything onto 1 GitHub page, which helps you get down 1 single consolidated window into all of your tutorials which will be conducted over here. So, if you look over here. So, the address for this 1 is on GitHub, so it is GitHub point com slash iitcliff slash dlvcnptel, so it is just as mentioned over here. So, once you go down over here the simplest stuff which you can on anyways do is basically clone or download this 1, so give you a basic reminder a please ensure that you are using a Linux based system in order to do that because, like we are going to make use of a very specific library called as PyTorch which I will be showing you subsequently.

So, PyTorch supports only on Linux base place system, so unfortunately you cannot do it on a windows based system, though I mean you would be seeing that this desktop which is getting recorded on is a window based system because, we are connected down to another server system remotely in order to do all of this.

So, I will not be complicating that part of the stuff because, most of you I assume will not getting access to direct dedicated servers and for none of the exercises which we are going to do in this learning based curriculum over here, you will not even be needing access to a full scale server to run down your codes. So, that that does not hesitate and you can pretty much run it down on your laptops as well, there is no very specific requirement that your system needs to have access to a GPU and though it does help a lot for accelerating and doing it faster, even a very bare 1 core I5 based CPU machine with barely about 8 gigs of ram on board should be enough to do maybe a bit slow, but it will finish down within say much shorter than the life time.

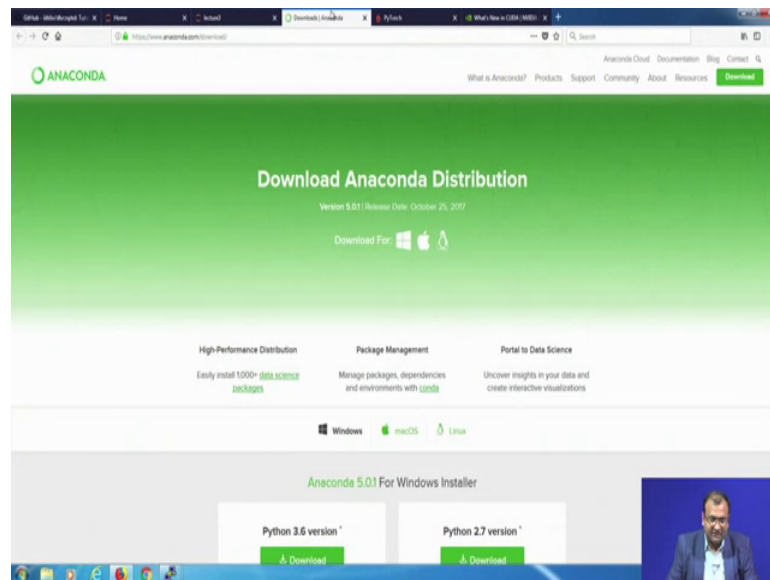
So, something which might be taking down a couple of seconds with a GPU system, so in case of without a GPU on a CPU system it might take down a few minutes, but you will still be able to see the results. So, this is the first foremost point is to go on this GitHub page and this is where you do.

(Refer Slide Time: 03:24)



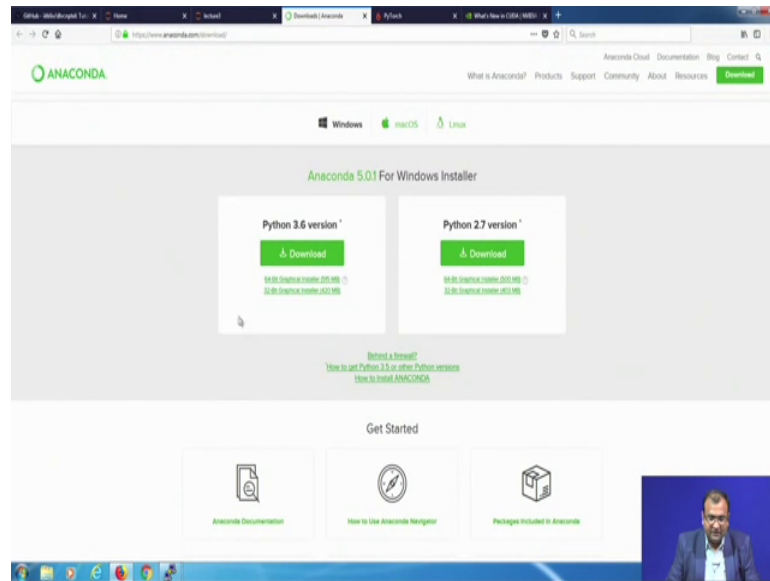
So we are going to populate it incrementally with all the lectures coming down 1 by 1 as of now today you would be seeing that just lecture 2 3 and 5 which is placed over here. So, what we will be making use of is something called as jupyter notebook and that is supported on anaconda Python.

(Refer Slide Time: 03:41)



So, the python distribution which we make use of within these tutorials over here and which you are also suggested to you make use of is the anaconda python distribution.

(Refer Slide Time: 03:52)

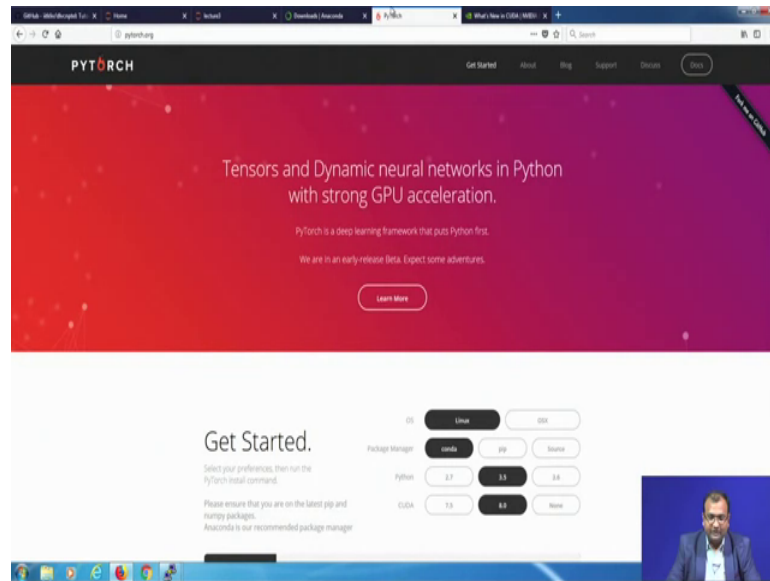


Now, remember that there are 2 versions of it python 3.6 and python 2.7. So, for a much conformal support we would be going down with Python 2.7, which all though to a lot of you might appear to be an older version, but major of these libraries for computer vision and visual computing tasks are what exists with python 2.7 on legacy and this is a time tested 1. So, just first get down your python 2.7 you can download it over here and so I mean based on whether you are on windows or you are on a Mac or Linux and since most of you guys would be on Linux.

So, it is advice that you download the Linux version over here, so you can click on these OSS over here select your Linux and then you get down your installer coming down over here so and then please check out whether you using a 32 bit system or a 64 bit instruction set system, most modern systems will be your 64 bit system. So, please download this 64 bit installer for x 86 systems and so power pcs are a different variant of processors as well which are available just for few computers and generally for servers.

So most of you will not be having access to a power pc, so that does not need you to download this 1; so just ensure that you are on the correct version of anaconda which you are downloading, now once that is downloaded and installed on your system.

(Refer Slide Time: 05:14)



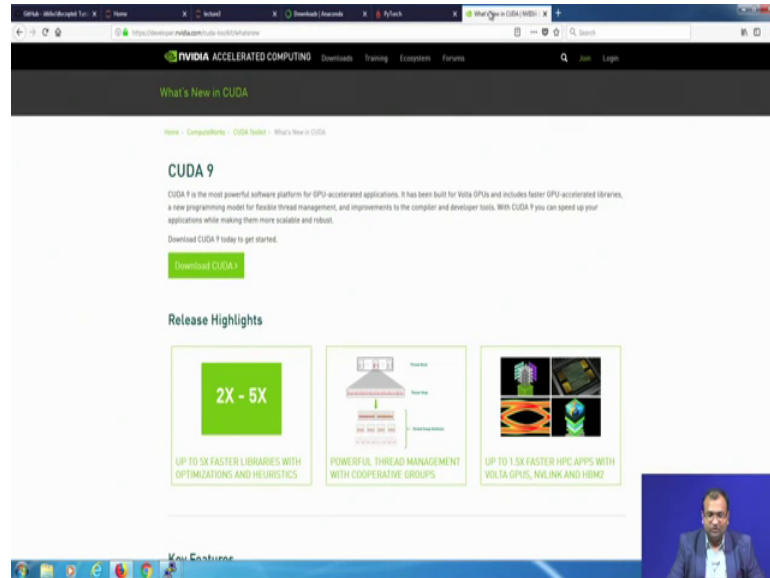
Now, the library which we will be using for our coding practices is what is called as PyTorch, it is quite simple you need to just go on PyTorch dot org and all of these links are actually given down on the GitHub page. So, if you are on the GitHub page you see that there is a dependency list build up over here and there are links given down. So, this is for your anaconda download this for your PyTorch download.

So, once you click on that PyTorch you will be ending up over here, now in order to get started there are few simple questions which you need to answer, so 1 is what is your os you select down Linux, what is your package manager? So, that will be conda because we are going to use make use of anaconda next is what is your python. So, that is 2.7 for us and what is your CUDA version. So, either you can choose 7.5 or 8 or even none, so if you do not have a GPU over there then just click on none otherwise like for me it was 8.

So, I am going to choose down 8 over here and then you just need to run this command, on your come on your terminal for your Linux and then that takes care of the rest of the installation, in case you get into some amount of problems over here. So, you can always put it on the discussions forum on PyTorch that can even be related to installations and they would be the best guys to actually help you out, rather than putting it back to us on our forum because we want always be equipped to solve all problems of third party software provider. Similarly for anaconda also if you get down into any kind of a glitch, so you can just contact out these people or you can just write post it on the blog over

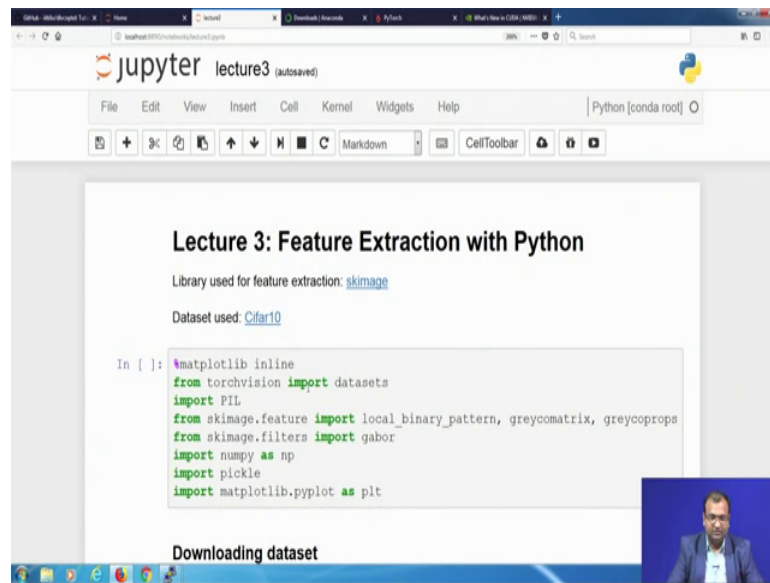
there or on support. So, within the support team they do make sure that they get back to you.

(Refer Slide Time: 06:52)



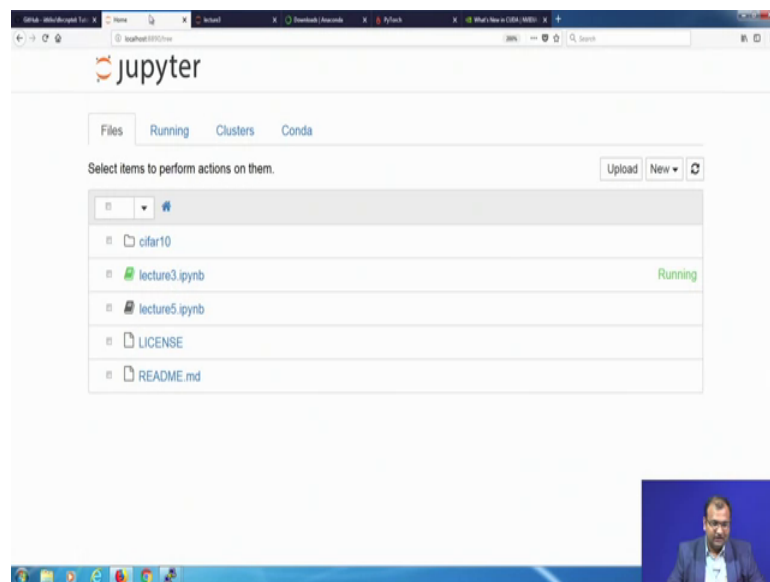
Now for CUDA: so the current version is CUDA9, but you need to keep in mind 1 thing that PyTorch is supported only till CUDA8 and for that reason we will not be using CUDA9, but we will be making use of CUDA8 actually. So, with CUDA it should be good enough to install PyTorch and get started, now given that once all of this is installed over there. So, what you need to do on your terminal is invoke something called as a jupyter notebook.

(Refer Slide Time: 07:16)



So jupyter notebook is basically a web based UI for your coding environment so and then once you invoke your jupyter notebook you get this kind of a home screen coming.

(Refer Slide Time: 07:27)



So, based on which whichever folder you are located in, so it is going to show you the directory listing within that folder, now what I can do is once I have the directory listing. So, I click on this particular 1, so which is lecture 3 which I want to do. So, once I click that invokes this particular page coming down over here and this is what is something I will be getting my initial access to; now if you look under this page there are some

number of these bits and parts of python codes, if you are quite used to python encoding or if you are new to python coding then it still should not be much measure of a problem, because I am going to explain you what each of these steps actually mean now.

So, there would be these codes which go down and that are exactly how we are going to solve the first part of it and that is our first exercise which we are going to solve. So, you have already finished off classical methods of feature extraction which included textures and then majority of them were just texture based features you had wavelets as well. So, we will be making use of those and here is when I get you to show on them.

So, what we are going to do is let us do a very brief walkthrough of what is there, so if you look on this let us just zoom like really big. So, this should be a font size which is enough for you to understand, so the first thing is that what I am doing is you have this first line which is from torch vision import data set. So, what it technically does is that as in any of your pager records when major or languages you would be having a set of dependencies called as libraries. So, here also in python you have your libraries and these library since it is an object oriented.

So, they are hierarchically packed with in more containers as well, so torch vision is basically your container within which there is another library container which is called as data sets and the point is that I do not want to import complete torch vision over there. So, torch vision this torch thing comes down from the library PyTorch, which we make use of and there is a separate base for it which is called as torch vision and this is just for the vision pipe lines over the standard computer vision ways of doing it with torch, now from there the only thing which I would need for this particular exercise is dataset.

So, instead of loading that complete library and eating up space on my ram, I am just going to load down the data sets part over there. Then the next part is to include down is to import python imaging library this is just for reading and writing images and showing it out in a perfect way. Next is, we will be making use of cycads actually and this is the cycads image processing toolbox which will be making use of.

So, from there I would just need certain functions which are within the sk image to add features, now within that the first thing which I would be making use of is get down local binary pattern. So, this is a python function a dot py file, which I need to input and have

it ready for a lookup within my resources path. So, I am just going to input this particular file the next 1 is for gray level co-occurrence matrix.

So, that is a glcm which we have already studied and within gray level co-occurrence matrix, once you have that matrix coming out you cannot use make use of that complete matrix because, say you have a 8 bit image and you and you went to get down across all of these bits you will get down some 256 cross 256 sized matrix. So, instead of going through that 1 we typically would like to compress that and get down certain uniquely describing features and they are your gray g l c m properties and that is what this particular function is going to provide you with.

The other 1 is Gabor filter and since Gabor filters are typically within a filters toolbox. So, I would just be inputting this Gabor function from sk image dot filters more of what I do is I import this function called ass pickle and pickle is basically going to help me in storing my data. So, this is a container so if you are more of used to mat lab, so you would be storing down your data structures some sort of matrices or arrays, which you have intermediately computed in terms of our dot mat file; here what if we would be doing is in terms of a dot pickle file and for that reading writing part of it I would need to import this particular library. Then I would also need access to my numpy library which is the numerical programming library for python and this will help me in defining my arrays the data structure of arrays and everything and in order to avoid writing down.

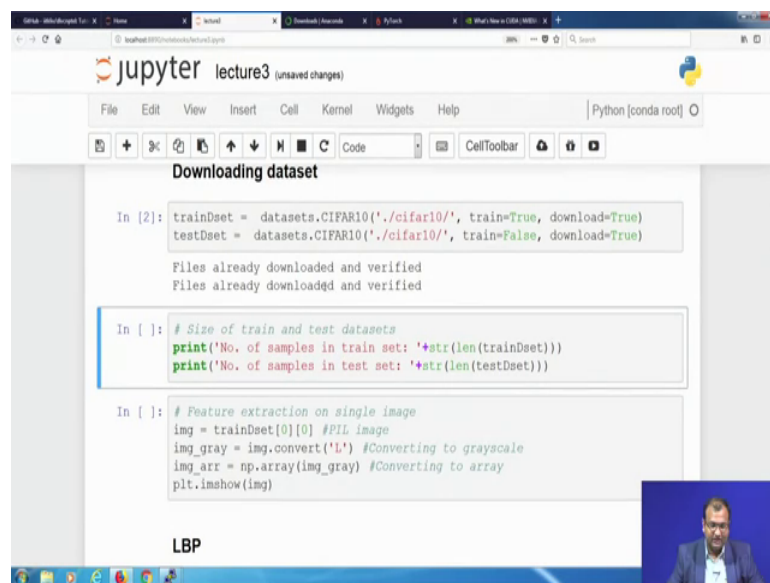
At every single point numpy dot some function call, I would just be providing a small acronym for that 1 for numpy and that is called as np. So, every time I just write down np dot something that will expand it in terms of numpy dot something and also in order to plot down my figures I would be making use of matplotlib dot pyplot and then that would help me in doing it out also I define certain macros over here.

So, the first macro which I defined out and that is more of to be consistent with jupyter environment itself, if you are using some other environment then you might not need to write down this first, 1 which is just if I am doing a plot then let the plot be an inline plot, so that it appears on my html itself. So, this is 1 part of the code block which let it get this 1 running. So, the simple part is just go over there and then you have the run selected cell. So, we select that 1 and run it. So, while it runs you would be you had seen down that small asterisks coming.

So, that is when this all the codes within this block are running and it has not yet completed and once it is completed this is the first time step of the run which has been done and you get this 1. So, this is just a comment, so if you can choose to run it, but it does not actually do anything. So, the next part over here if you see what I am doing is I am creating 2 data sets over here. So, this data set is the CIFAR 10 data set. So, CIFAR is basically a small images data set which is used for.

So, there are 10 classes of items images which are divided into 10 classes there are small thumbnails of the 32 plus size and you would be make basically classifying them into these different kinds of classes over there, now the point is since we are dealing with visual computing, so we need to take in images itself and these are all color rgb color images. So, that is available directly within the torch vision data sets. So, now we had imported torch vision data set over here.

(Refer Slide Time: 14:02)



```

jupyter lecture3 (unsaved changes)
Python [conda root]

File Edit View Insert Cell Kernel Widgets Help

Downloading dataset

In [2]: trainDset = datasets.CIFAR10('./cifar10/', train=True, download=True)
testDset = datasets.CIFAR10('./cifar10/', train=False, download=True)

Files already downloaded and verified
Files already downloaded and verified

In [ ]: # Size of train and test datasets
print('No. of samples in train set: '+str(len(trainDset)))
print('No. of samples in test set: '+str(len(testDset)))

In [ ]: # Feature extraction on single image
img = trainDset[0][0] #PIL image
img_gray = img.convert('L') #Converting to grayscale
img_arr = np.array(img_gray) #Converting to array
plt.imshow(img)

LBP

```

Now, I can go into data sets and then from there I input the CIFAR 10 data set, now the point is when it imports locally, so its either imported somewhere earlier and then you can just fetch it and keep it within your folder called a CIFAR 10. So, this CIFAR 10 is basically another folder which is created within my local directory.

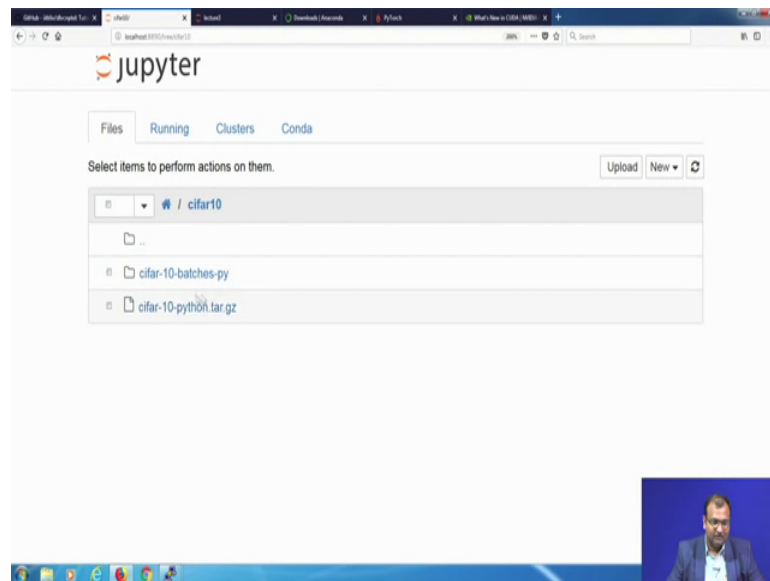
So, you see your local directory over here where within your jupyter environment. So, if you go down into this part you would see another folder called as CIFAR 10, so this is what it needs to create. So, initially when you are downloading this whole thing from

GitHub you do not see that directory anyways because we did not upload the data set, that is a huge bulky file to be uploaded and we just do not want to upload all of those because, they are available from a secondary source as well to come down.

So, once first you download this whole thing you will not be getting this folder called as CIFAR 10, now when you run in this line this is the first time when you will be getting none your CIFAR 10; now the reason I have this already running and this folder present is because, I really want to make this faster, so if you have it already downloaded.

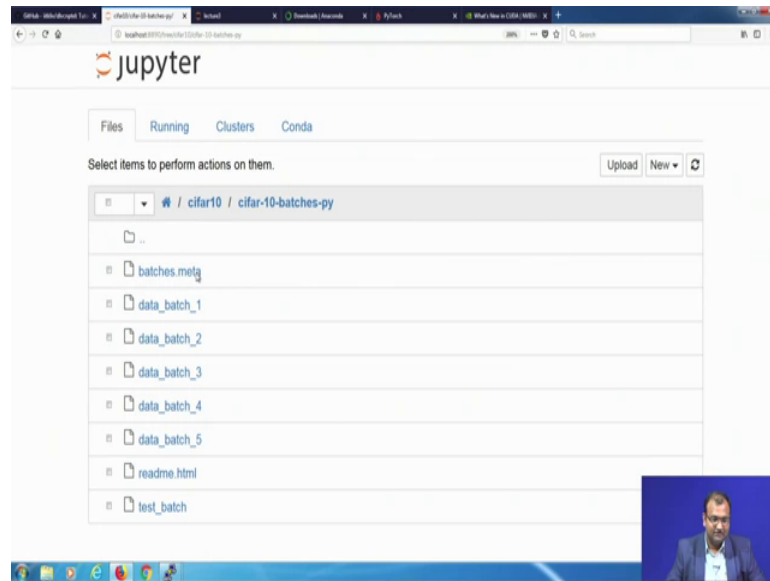
Somewhere you can put it within the folder called as a CIFAR 10 and that solves the purpose otherwise you need to download it from scratch. So, here like what it would do is it just goes over there and sees that files are already downloaded and they are perfectly. So, they will just be creating these 2 small data sets for me over there.

(Refer Slide Time: 15:20)



So, I get down on CIFAR 10 and within CIFAR 10 I see these 2, so this part over here CIFAR 10 python dot tar dot tg said this is my actual file which downloads over there, from the source and then within CIFAR 10 batches, it will be creating my training and test batches over here.

(Refer Slide Time: 15:33)



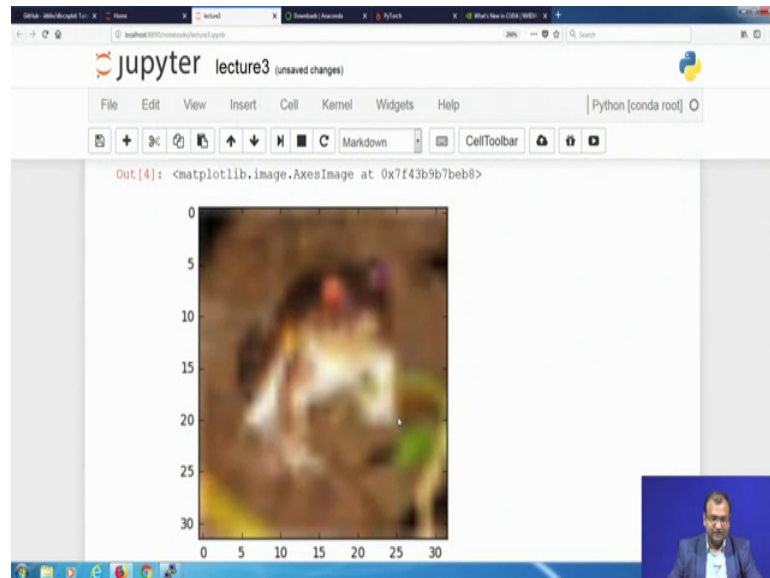
So, now once that is done, so what I can do is I move back on to my main directory over here and let us go to the next part of it. So, here what I am trying to do is get into the data set and try to find out what is the length of the data set or how many number of training samples and how many number of testing samples are present over there. So, it is quite simple. So, it says basically find out use the Len function which gives you a length of array within the train data set and within the test data set, so just run that part of it.

So, these 2 pointers are already over there, so we just find out what is the length over there and then it just converts it to a string and prints it out over there. So, you know that your training data as it is of 50000 images and training and testing data set is of 10000 images, now once that is done the next part is we come down over here which is feature extraction on a single image. So, initially what we will be doing is let us see what these images look like, so what I am doing is I take down 1 of these images which is at the 0 comma 0 location.

So, this is the first image present on my training data set and then I stored that as a variable image, now what I am doing is I convert this image to a grayscale map which is image underscore gray and then I convert this into a numpy array. So, this is whole thing which stays as a grayscale integer format. So, that will typically be coming down as some sort of a container with me, now that container I want to convert it to an numpy

and then let is just plot that 1, so if I run that you would be seen that this is sort of an image which you get done.

(Refer Slide Time: 17:10)



Now it is really fuzzy to understand, but these is basically if you like really go far off and then try to see into it, so this is the image of a frog; so the first data which we were looking down was just a frog.

(Refer Slide Time: 17:23)

```
In [ ]: # Finding LBP
feat_lbp = local_binary_pattern(img_arr,8,1,'uniform') #Radius = 1, No. of r
feat_lbp = np.uint8((feat_lbp/feat_lbp.max())*255) #Converting to uint8
lbp_img = PIL.Image.fromarray(feat_lbp) #Conversion from array to PIL image
plt.imshow(lbp_img, cmap='gray') #Displaying LBP

In [ ]: # Energy and Entropy of LBP feature
lbp_hist, _ = np.histogram(feat_lbp,8)
lbp_hist = np.array(lbp_hist, dtype=float)
lbp_prob = np.divide(lbp_hist, np.sum(lbp_hist))
lbp_energy = np.sum(lbp_prob**2)
lbp_entropy = -np.sum(np.multiply(lbp_prob, np.log2(lbp_prob)))
print('LBP energy = '+str(lbp_energy))
print('LBP entropy = '+str(lbp_entropy))
```

Now once that part is done, next is we would like to compute out lbp features for local binary patterns for us, now for local binary patterns what you are going to do is you

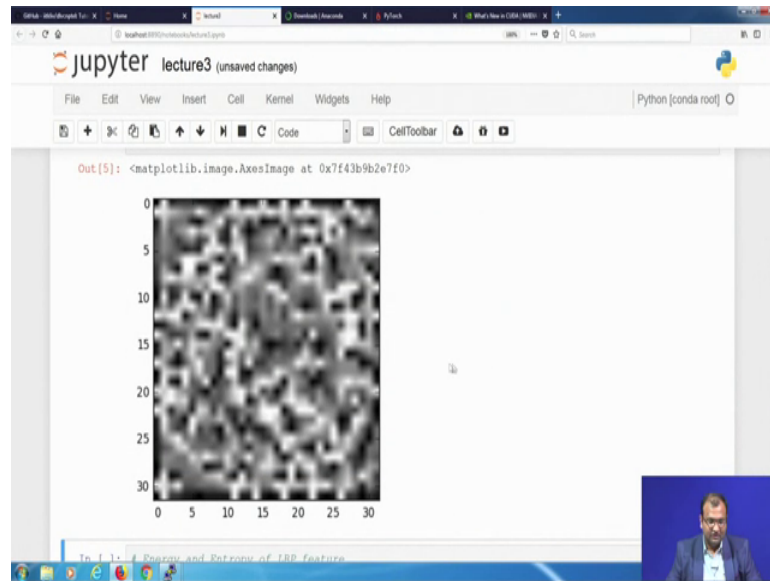
would need the main image array. So, that is present over here as a numpy array and this lbp is over here they will be computing it in terms of a numpy. So, you remember that we had already imported from `sk image features` this particular file called as `local_underscore_binary_underscore_pattern` for computing lbp. So, the arguments which go into this is the image array which is a grayscale 1, then this number over here is basically the number of points you would be taking around the central point.

So, you remember clearly from our earlier discussions on from in the last class on lbp, where you take 1 single point and then if you are looking into it is 3 cross 3 neighborhood you would be getting 8 such neighbors along that point which are at a distance separation of 1 pixel and then what extra is added over here is what is the kind of a sampling you would do.

Now what it allows within these functions is that you can choose down any number of neighbors you can choose 4 5 6 7 typically for the 3 cross 3 that that would not be a uniform pixel kind of a distribution, but you can interpolate and go down to those kind of forms. So, what we choose to do is we take a circular neighborhood with a uniform sampling over there and that is what these arguments go down, you can get down further more details if you just search out on the help file for this function called as `local_underscore_binary_underscore_pattern` and that would give you all the details over there, next the idea is basically once you get down these lbps over there is to Convert this lbp matrix onto a 8 bit format over there. In order to get down what are the what is the range of these lbps which come down.

So, once you have this lbp converted over there, the next point let us like see what this lbp feature on a point to point basis looks like, so we compute this 1 and this is more or less what is the lbp for that frog image, which we get to see over here.

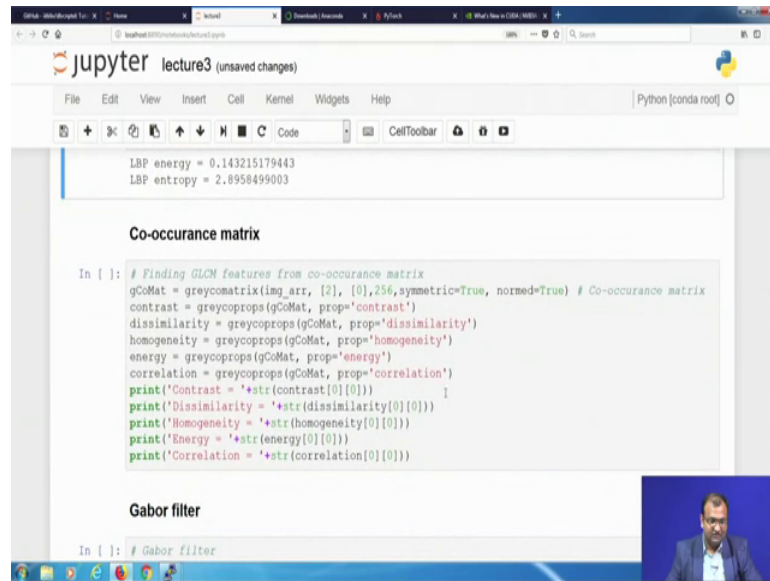
(Refer Slide Time: 19:29)



It is really hard to actually find out whether there is a frog or something or not from. So, many points over there which are so distinct these right; now from an lbp what you can get down is you can compute out the histogram of lbps you can which is a consolidated feature rather than feeding in this complete matrix of 32 plus size, the next is you can get down.

So, once you have this histogram computed you can get down the probability pdf over there from this histogram, then that would help you to get down the energy and entropy as well now once you have all of these you can basically use energy and entropy as 2 different distinct 1 dimensional features to in order to represent this; because, the main purpose is that this whole image needs to be represented in terms of 1 single scalar value and a set of those multiple number of scalar values which will be your features which describe this image. So, for that what we do is we just evaluate this part over here and I get down that lbp energy of this much and lbp entropy of this much is what defines all of this together present in this image.

(Refer Slide Time: 20:35)



```
lbp energy = 0.143215179443
lbp entropy = 2.8958499003

Co-occurrence matrix

In [ ]: # Finding GLCM features from co-occurrence matrix
gCoMat = greycomatrix(img_arr, [2], [0], 256, symmetric=True, normed=True) # Co-occurrence matrix
contrast = greycoprops(gCoMat, prop='contrast')
dissimilarity = greycoprops(gCoMat, prop='dissimilarity')
homogeneity = greycoprops(gCoMat, prop='homogeneity')
energy = greycoprops(gCoMat, prop='energy')
correlation = greycoprops(gCoMat, prop='correlation')
print('Contrast = '+str(contrast[0][0]))
print('Dissimilarity = '+str(dissimilarity[0][0]))
print('Homogeneity = '+str(homogeneity[0][0]))
print('Energy = '+str(energy[0][0]))
print('Correlation = '+str(correlation[0][0]))

Gabor filter

In [ ]: # Gabor filter
```

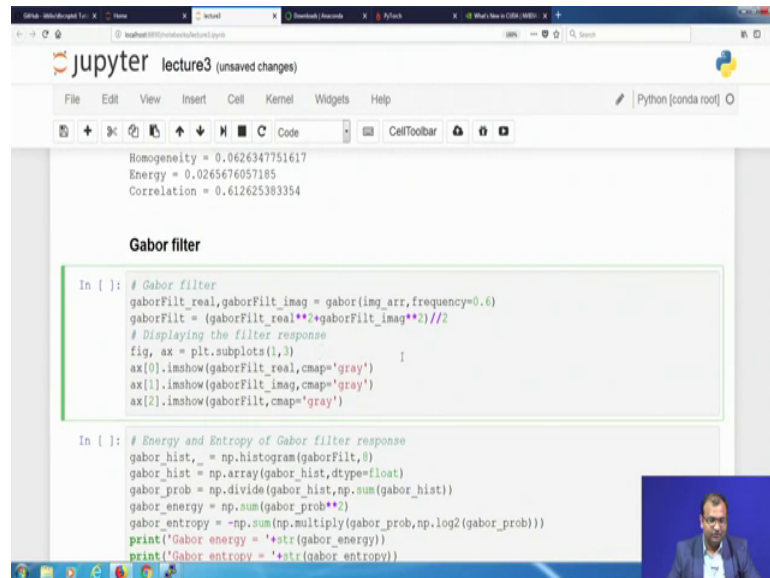
Now, once that goes down the next part is to find it out on the co-occurrence matrix. So, in a co-occurrence matrix what I need to do is I need to get my image over there. The next point which you need to do is you need to find out, so basically at how many neighbors would you be looking. So, would you be like taking down only 1 directional of it or 2 directional of it, the next argument over there is what is the orientation of your vector whether it is at 0 degrees 45 degree 90 degree.

So, based on whether it is east pointing it is 0 degree, if it is north east pointing it is 45degree, if it is not pointing then it is 90 degree so and so far, this number 256 is basically the number of gray levels; you have in your gray level co-occurrence matrix to be computed and this other parts on symmetric and norm are basically to show down how to handle down the boundary conditions present over there. Now once you have done down this 1 from your goal gray level co-occurrence matrix what is computed is another matrix of the size of 256 cross 256, which is present in this a variable called gCOMat.

Now my point is that I cannot again make use of this 256 cross 256 bit matrix, which corresponds to just the 32 cross 32 image; but I would try to get down more scalar values from that 1 the first scalar value is basically to get done contrast, second scalar value is to get down dissimilarity, third is homogeneity next is energy next is correlation and for all of these.

We make use of this separate function called as greycoprops, which takes in 1 of the arguments as a gray level co-occurrence matrix and then whatever property you need to calculate it out that is another like scalar disk this. So, this is a small label descriptor which goes into the function. So, that it knows what to compute and give you, so this help in getting and this are the different measures for that 1 particular image.

(Refer Slide Time: 22:29)



```
Homogeneity = 0.0626347751617
Energy = 0.0265676057185
Correlation = 0.612625383354

Gabor filter

In [ ]: # Gabor filter
gaborFilt_real,gaborFilt_imag = gabor(img_arr,frequency=0.6)
gaborFilt = (gaborFilt_real**2+gaborFilt_imag**2)**2
# Displaying the filter response
fig, ax = plt.subplots(1,3)
ax[0].imshow(gaborFilt_real,cmap='gray')
ax[1].imshow(gaborFilt_imag,cmap='gray')
ax[2].imshow(gaborFilt,cmap='gray')

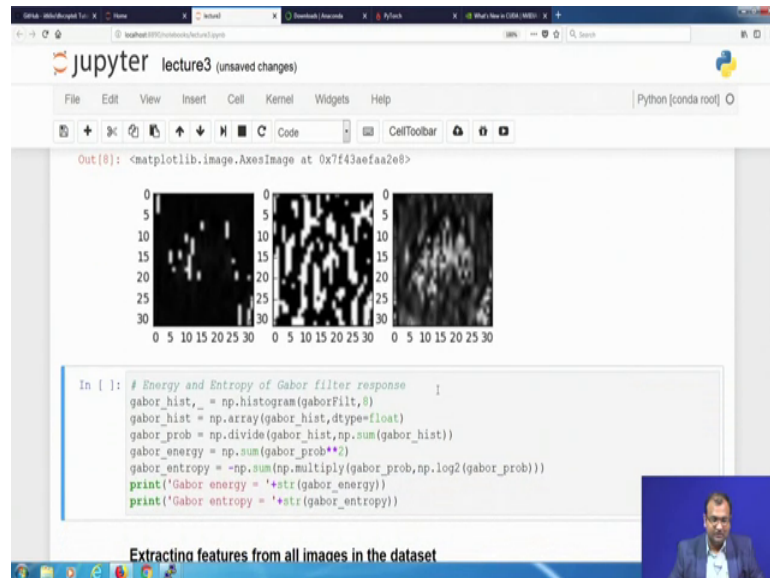
In [ ]: # Energy and Entropy of Gabor filter response
gabor_hist,_ = np.histogram(gaborFilt,8)
gabor_hist = np.array(gabor_hist,dtype=float)
gabor_prob = np.divide(gabor_hist,np.sum(gabor_hist))
gabor_energy = np.sum(gabor_prob**2)
gabor_entropy = -np.sum(np.multiply(gabor_prob,np.log2(gabor_prob)))
print('Gabor energy = '+str(gabor_energy))
print('Gabor entropy = '+str(gabor_entropy))
```

Now, from there the next 1 is to get into wavelets and do it, so, for we choose to do it with Gabor filters now as you remember from your Gabor filtered equations in the last class. So, there would be 2 different things which you need to take care of within a gabber filter and that is like what is the frequency and then what we are typically doing over here, that is it will rotate and generate out. So, the first part is that you need to have your image given down over there, as well as what is your frequency at which you would like to operate.

Now, the other part is what is the angle at which it is located and what are the variables over there? So there are some things which are auto tuned within the system or you can also choose to give them. So, you can read down with within the details more over there. Now given that at any point you will be getting down to components of your wavelet decomposition 1, is the real value path, another is the imaginary value path. Now from there we just need to get down the magnitude valued part and so we find out we square

and find out what is the magnitude now once then. So, you can just have a look into this part of it as well.

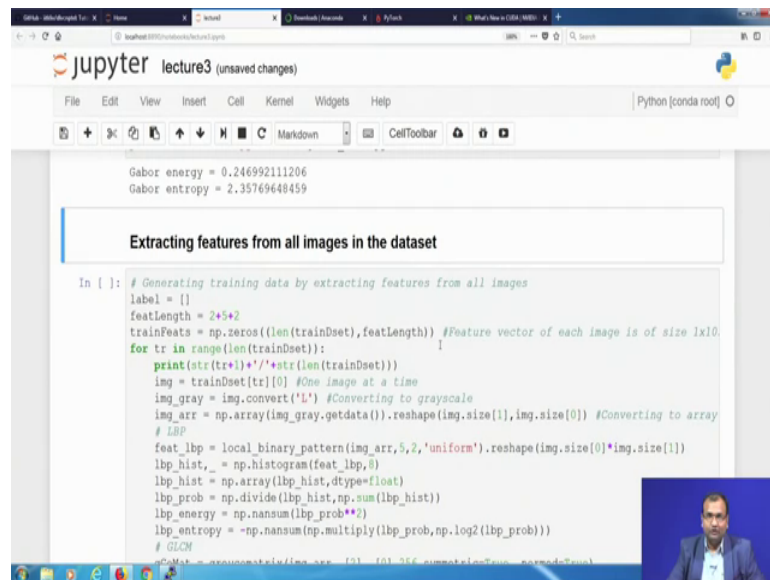
(Refer Slide Time: 23:34)



So, you see that this is the real valued part of the gable filtered output, is the imaginary part and this is basically the consolidated magnitude response over there. The next part is from your Gabor or you would we would like to get down some scalar representations rather than these kind of matrix representation and they are basically your probability energy entropy and then probabilities used within this energy and entropy. So, you can calculate that and get done that the Gabor energy is this much and interface this much.

Now this is till now what we have done was just for 1 of these images which was at the first location within my training data set, now in order to do it for training I would need to do.

(Refer Slide Time: 24:12)



```
Gabor energy = 0.246992111206
Gabor entropy = 2.35769648459

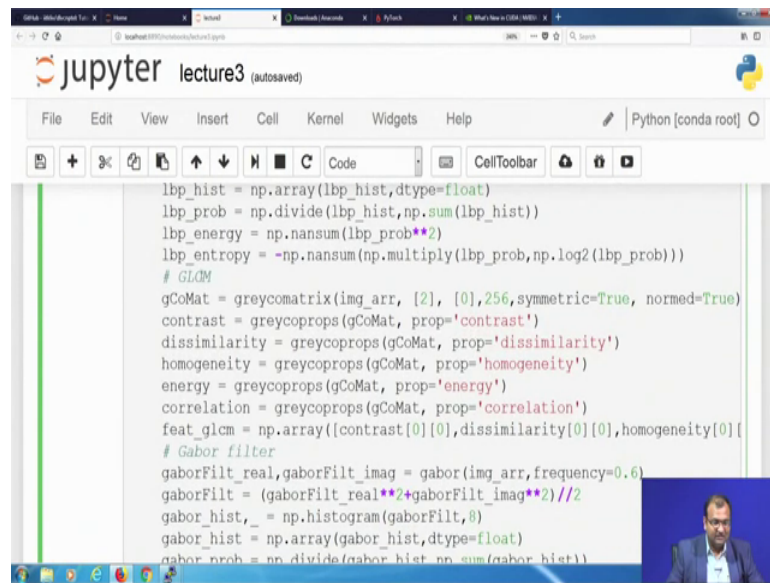
Extracting features from all images in the dataset

In [ ]: # Generating training data by extracting features from all images
label = []
featLength = 2+5+2
trainFeats = np.zeros((len(trainDset),featLength)) #Feature vector of each image is of size 1x10
for tr in range(len(trainDset)):
    print(str(tr+1)+'/'+str(len(trainDset)))
    img = trainDset[tr][0] #One image at a time
    img_gray = img.convert('L') #Converting to grayscale
    img_arr = np.array(img_gray.getdata()).reshape(img.size[1],img.size[0]) #Converting to array
    # LBP
    feat_lbp = local_binary_pattern(img_arr,5,2,'uniform').reshape(img.size[0]*img.size[1])
    lbp_hist,_ = np.histogram(feat_lbp,8)
    lbp_hist = np.array(lbp_hist,dtype=float)
    lbp_prob = np.divide(lbp_hist,np.sum(lbp_hist))
    lbp_entropy = np.nansum(lbp_prob**2)
    lbp_entropy = -np.nansum(np.multiply(lbp_prob,np.log2(lbp_prob)))
    # GCM
    #GCM = gcm.calculate(img_arr, [2], [0], 255, cumulative=True, normed=True)
```

It for the whole data set and that would mean that, I cannot have any further looking into 1 image at a time. So, I need to run down some sort of a for loop within the length of my training data set as in over here, in order to calculate this for the complete training data set because I cannot do it on a 1 to 1 basis. So, what I need to do is define some sort of a matrix which is called as the training features matrix.

So, this is a 2 d matrix which is the number of rows in this matrix is equal to the length of the training data set, the number of columns is equal to the length of features. Now how many features we found out was basically 2 plus 5 plus 2 and that makes it 9 features which we are going to have over here; now for this part what we do is we write down first for loop which basically ranges over the whole length of the training data set, once you get over the whole length of the training data set you need to find out 1 feature at a time. Now once you have 1 feature at a time coming down you need to calculate all of these features 1 image at a time, coming in you will be needing to calculate all of these features the first 1 is your lbp features.

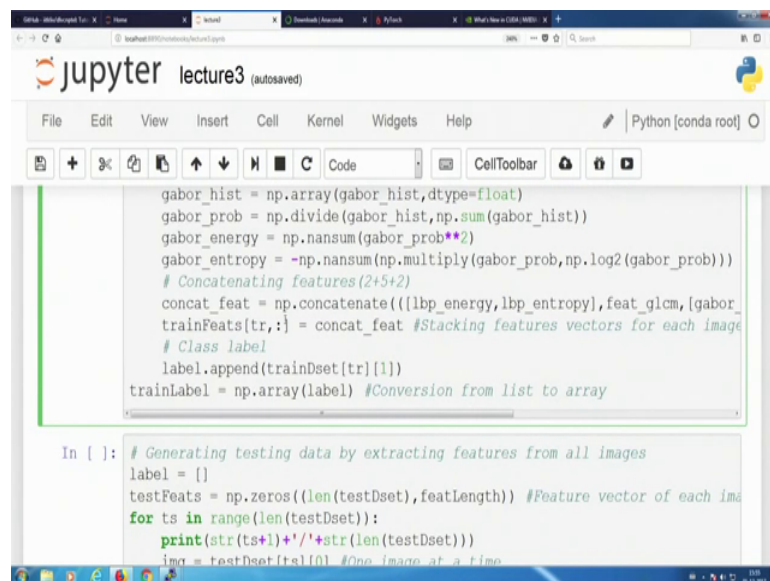
(Refer Slide Time: 25:18)



```
lbp_hist = np.array(lbp_hist, dtype=float)
lbp_prob = np.divide(lbp_hist, np.sum(lbp_hist))
lbp_energy = np.nansum(lbp_prob**2)
lbp_entropy = -np.nansum(np.multiply(lbp_prob, np.log2(lbp_prob)))
# GLCM
gCoMat = greycomatrix(img_arr, [2], [0], 256, symmetric=True, normed=True)
contrast = greycoprops(gCoMat, prop='contrast')
dissimilarity = greycoprops(gCoMat, prop='dissimilarity')
homogeneity = greycoprops(gCoMat, prop='homogeneity')
energy = greycoprops(gCoMat, prop='energy')
correlation = greycoprops(gCoMat, prop='correlation')
feat_glcm = np.array([contrast[0][0], dissimilarity[0][0], homogeneity[0][0], correlation[0][0], energy[0][0], correlation[0][0]])
# Gabor filter
gaborFilt_real, gaborFilt_imag = gabor(img_arr, frequency=0.6)
gaborFilt = (gaborFilt_real**2 + gaborFilt_imag**2) ** 0.5
gabor_hist, _ = np.histogram(gaborFilt, 8)
gabor_hist = np.array(gabor_hist, dtype=float)
gabor_prob = np.divide(gabor_hist, np.sum(gabor_hist))
```

Next is your gray level co-occurrence matrix and Gabor filters.

(Refer Slide Time: 25:21)

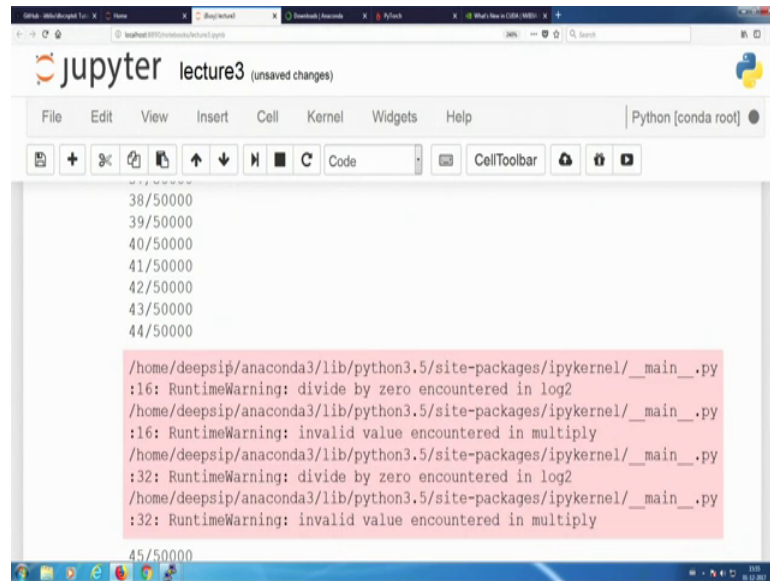


```
gabor_hist = np.array(gabor_hist, dtype=float)
gabor_prob = np.divide(gabor_hist, np.sum(gabor_hist))
gabor_energy = np.nansum(gabor_prob**2)
gabor_entropy = -np.nansum(np.multiply(gabor_prob, np.log2(gabor_prob)))
# Concatenating features (2+5+2)
concat_feat = np.concatenate((lbp_energy, lbp_entropy), feat_glcm, [gabor_energy, gabor_entropy])
trainFeats[tr, :] = concat_feat # Stacking features vectors for each image
# Class label
label.append(trainDset[tr][1])
trainLabel = np.array(label) # Conversion from list to array

In [ ]: # Generating testing data by extracting features from all images
label = []
testFeats = np.zeros((len(testDset), featLength)) # Feature vector of each image
for ts in range(len(testDset)):
    print(str(ts+1)+'/'+str(len(testDset)))
    img = testDset[ts][0] # One image at a time
```

Now, once you have all of them you need to concatenate that into 1 row matrix and then you keep on concatenating 1 below the other and you get your 2 d matrix coming down, so if we run this part.

(Refer Slide Time: 25:34)



The screenshot shows a Jupyter Notebook window titled "lecture3 (unsaved changes)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with various icons. The main area displays a code cell with the following content:

```
38/50000
39/50000
40/50000
41/50000
42/50000
43/50000
44/50000

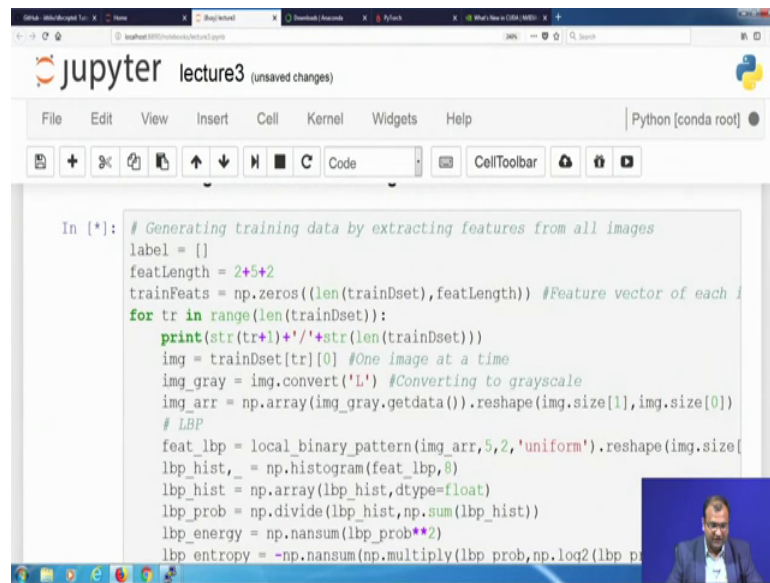
/home/deepsip/anaconda3/lib/python3.5/site-packages/ipykernel/_main_.py
:16: RuntimeWarning: divide by zero encountered in log2
/home/deepsip/anaconda3/lib/python3.5/site-packages/ipykernel/_main_.py
:16: RuntimeWarning: invalid value encountered in multiply
/home/deepsip/anaconda3/lib/python3.5/site-packages/ipykernel/_main_.py
:32: RuntimeWarning: divide by zero encountered in log2
/home/deepsip/anaconda3/lib/python3.5/site-packages/ipykernel/_main_.py
:32: RuntimeWarning: invalid value encountered in multiply

45/50000
```

You see this verbose commenting coming down and then it keeps on running; so together that would finish it off there might be certain warnings at positions and you just need to escape it out. So, it would take quite some time because it needs to do it over 50000 of those, but if you look through it; so it is pretty much fast because there is this like the rate at which it is running down is not.

So, tidy slow as well in the duration of where we are speaking you can already see this quite going on. So, we just have a verbose command given down over there. So, if you would like to get rid of this part then the simple task is that you do not keep 1 printing this part over here which is your print statement.

(Refer Slide Time: 26:26)



```
In [*]: # Generating training data by extracting features from all images
label = []
featLength = 2+5+2
trainFeats = np.zeros((len(trainDset), featLength)) #Feature vector of each
for tr in range(len(trainDset)):
    print(str(tr+1)+'/'+str(len(trainDset)))
    img = trainDset[tr][0] #One image at a time
    img_gray = img.convert('L') #Converting to grayscale
    img_arr = np.array(img_gray.getdata()).reshape(img.size[1], img.size[0])
    # LBP
    feat_lbp = local_binary_pattern(img_arr, 5, 2, 'uniform').reshape(img.size[1], img.size[0])
    lbp_hist, _ = np.histogram(feat_lbp, 8)
    lbp_hist = np.array(lbp_hist, dtype=float)
    lbp_prob = np.divide(lbp_hist, np.sum(lbp_hist))
    lbp_energy = np.nansum(lbp_prob**2)
    lbp_entropy = -np.nansum(np.multiply(lbp_prob, np.log2(lbp_prob)))
```

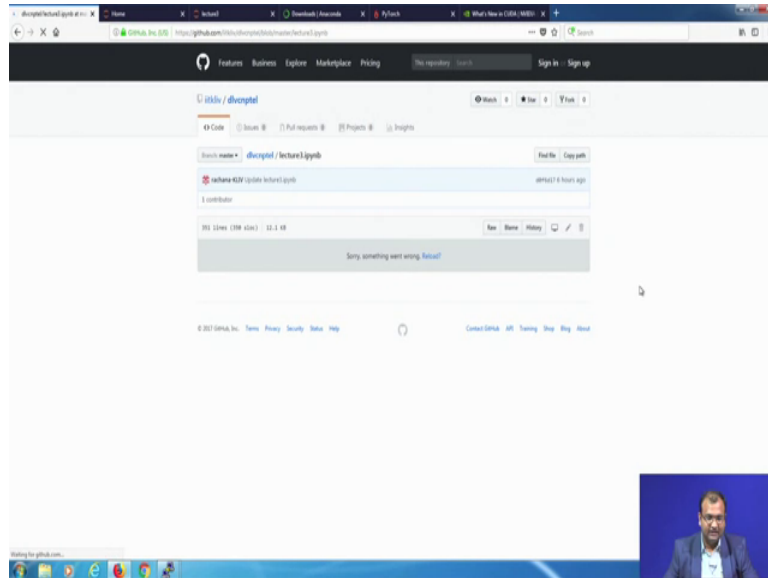
So, we will not print this part then it is not going to show down how many of them are done and then you just need to wait till it is completely gone off; now once that part is done the next part is basically to find it out on your test set as well. So, I just have to wait for some more time for this 1 to get over. So, let us just do a basic revision in that case, so what I did was I have my pre defined precursor coming down over here make sure I take down to my data set and then I can decide to print on the type of the data set or not.

But say if you are writing a fully fledged code over there, you will not need to get these parts of the code running down this way just for your explanatory purpose. So, you do not need to get down as to view 1 data 1 image from your data set then compute each of these features and then try to see them. So, these are parts which you can completely comment out and then just start directly from extracting features from all images in your data set. Now if you do not want to look into what is getting extracted, then you can just try to comment out this line or even delete this line that is not much of an issue.

So, this is just for printing out your status and you see that this still keeps on running over here. So, let us see how far it should be quite close to finishing it off because we are all most at 33000 try invoking another instance of so basically till then when we have done. So, the next part is that you just need to wait an observed till this part gets over for some more time, now once your features are extracted the next part of your code is

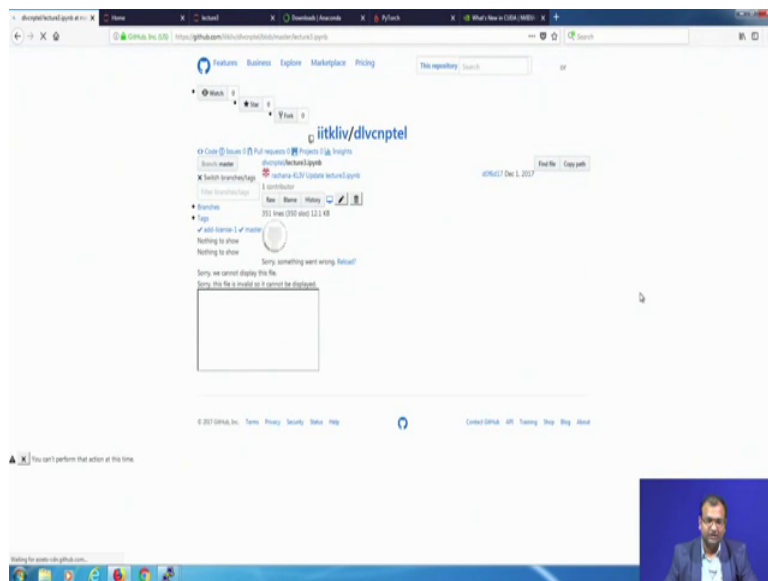
basically to go and look into what is the later part which is about Just saving out your feature.

(Refer Slide Time: 28:30)



So, I am just going to show it down on this offline part, so that we do not just keep on waiting till that part gets. So, 1 is you have your training data set on which your features are extracted.

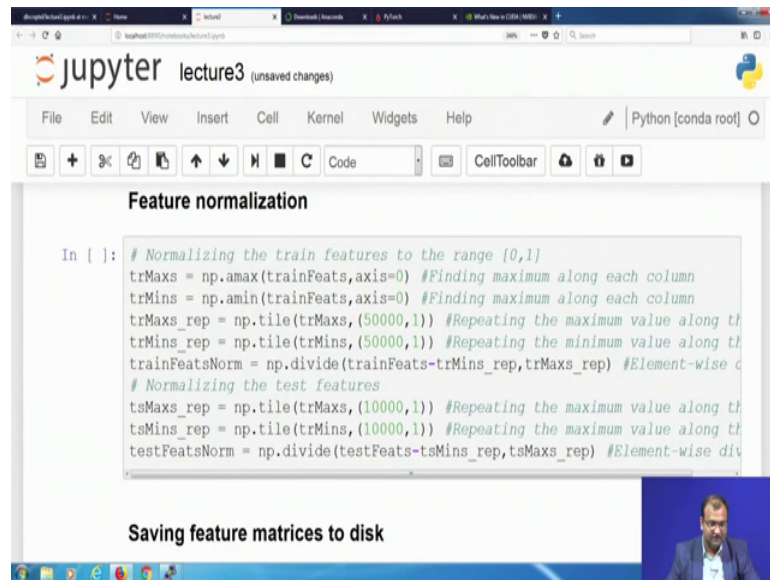
(Refer Slide Time: 28:43)



The next part is to go down on your test data set and also extract out features and completely show it and then eventually you can go. And basically save down all of those

features in terms of a pickle file as well and then that is what goes down in the end part; so, now this is over and the next part of it is basically to get down your testing ones features extracted over the whole dataset.

(Refer Slide Time: 29:11)



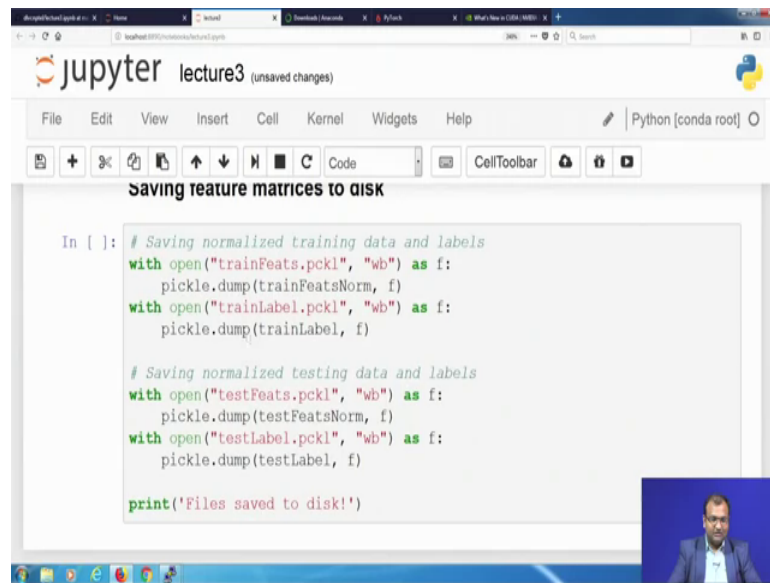
The screenshot shows a Jupyter Notebook window titled "lecture3 (unsaved changes)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations and execution. The main content area is titled "Feature normalization" and contains a code cell with the following Python code:

```
In [ ]: # Normalizing the train features to the range [0,1]
trMaxs = np.amax(trainFeats,axis=0) #Finding maximum along each column
trMins = np.amin(trainFeats,axis=0) #Finding maximum along each column
trMaxs_rep = np.tile(trMaxs,(50000,1)) #Repeating the maximum value along th
trMins_rep = np.tile(trMins,(50000,1)) #Repeating the minimum value along th
trainFeatsNorm = np.divide(trainFeats-trMins_rep,trMaxs_rep) #Element-wise d
# Normalizing the test features
tsMaxs_rep = np.tile(trMaxs,(10000,1)) #Repeating the maximum value along th
tsMins_rep = np.tile(trMins,(10000,1)) #Repeating the maximum value along th
testFeatsNorm = np.divide(testFeats-tsMins_rep,tsMaxs_rep) #Element-wise div
```

Below the code cell, the text "Saving feature matrices to disk" is visible. In the bottom right corner of the notebook window, there is a small video inset showing a man in a suit speaking.

And finally is where this part comes into play and that is where you are normalizing out the features. The whole reason for normalizing out all the features is basically to get down get each feature, dynamically varying within your training set in the range of 0 to 1. Now what you need to keep in mind is that whatever is the normalization range you apply within your training set, the same thing has to be applied within your testing set otherwise the nature of normalizations are going to quite vary.

(Refer Slide Time: 29:40)



```

Saving feature matrices to disk

In [ ]: # Saving normalized training data and labels
with open("trainFeats.pkl", "wb") as f:
    pickle.dump(trainFeatsNorm, f)
with open("trainLabel.pkl", "wb") as f:
    pickle.dump(trainLabel, f)

# Saving normalized testing data and labels
with open("testFeats.pkl", "wb") as f:
    pickle.dump(testFeatsNorm, f)
with open("testLabel.pkl", "wb") as f:
    pickle.dump(testLabel, f)

print('Files saved to disk!')
```

Now, once that is completely done the next part is that you would like to save all of your feature vectors for being used in the subsequent stages. So, we just save down your training features and training labels as well as test features and test levels in terms of a pickle file and then just print it all. So, once this part is complete you need to get down extract features for your training 1 and for your testing set then run the feature normalization and save it. And this should be good enough to get you started for the next lecture, which we will be covering down on classification with a very simple neural network.

Thanks, and stay tuned.