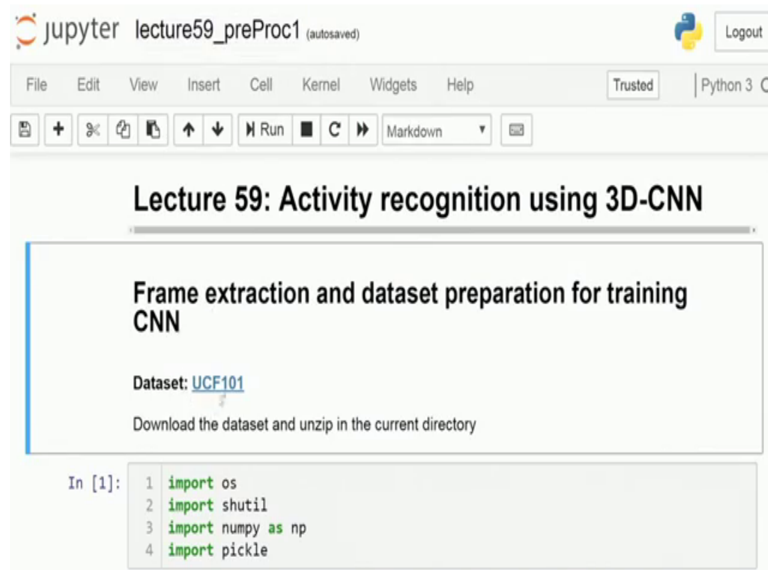
**Deep Learning for Visual Computing**
**Prof. Debdoot Sheet**
**Department of Electrical Engineering**
**Indian Institute of Technology, Kharagpur**

**Lecture – 59**
**Activity Recognition using 3D-CNN**

So, welcome to today's lecture and this is basically on activity recognition. And as you have done with the last 3 lectures classes on which we have understood about how to deal with these videos in terms of a tensors. And then what will be a possible kind of a network which you can do when we had studied about 2 different ways of doing it.

One was where you could actually take in this video as some sort of a 3D representation where you have the number of channels or the color channels has one of these axis. Then you have your time axis and then your x and y axis over there. And now using a standard mode of a CNN now instead of a 2 D CNN and the CNN over here becomes a 3D CNN.

(Refer Slide Time: 00:54)



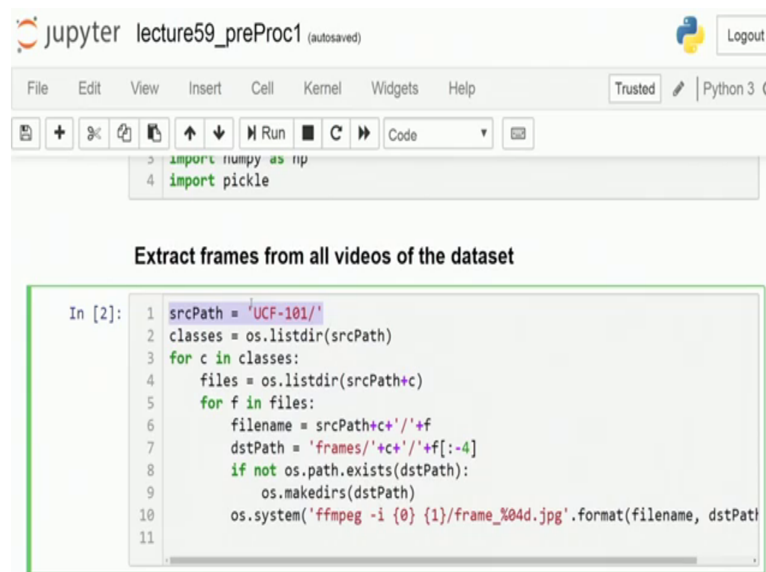

Now implementing this 3D CNN you can do classification on this video volumes as well. Now this is one of the ways the other way is where you basically try to represent your image in terms of just a simple 1D tensor. And that can be done by running a simple 2D CNN has a feature extraction on top of it. And once you get this 1D tensor over there; then you can do a temporal modeling with this 1D tensor. Now this second form of it is

what we are going to do in the next lecture. And today's lecture is more of to understand about this whole concept of how can be implement a 3D CNN working down on a video.

Now first and foremost where we start is that there is a lot of processing stuff as such to be done before you can get really started onto the word because videos. How they are packed and available in a data format. And how these CNNS are going to take them is a bit different and this is what we had discussed in the last lecture also. So what all axis you will have to flip and change and what will be the ordinality in the these axis coming down in as your tensor that also has to keep on changing over there. So that is what we get started over here.

So this first code over here which is preproc 1, so the whole objective of this pre processing 1 is basically to extract out frames from a whole video. And now these frames need to be in a particular order; so once extracted in a particular order and stored as a tensor that is the main job which we are going to do. So the first part is we just make use of a few of our utility files over there as our header. Now the whole point over there you do not see anything coming down as torch and that is the reason that we are not get started with the training, this is just the data pre processing part over there.

(Refer Slide Time: 02:32)



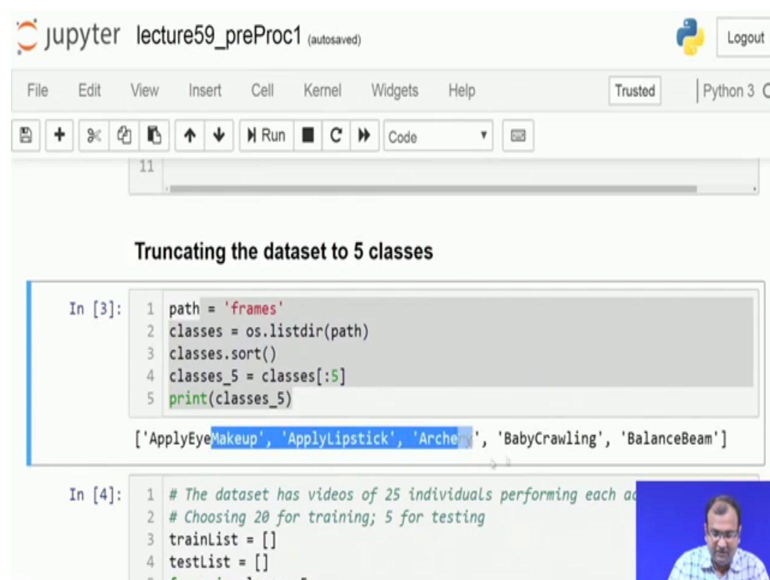

Now the dataset which we are using is UCF 1 o 1 you have the link given down over here. So this is basically a small clip videos for 101 different types of actions over there,

you have multiple videos which demonstrate 1 action and you have your train training set.

And your test set over there, but it is a 101 class classification problem. So typically there would be some small kind of activity say running, jogging, walking, combing your hair or drinking tea kind of activities which are denoted over there. And then you have to classify whether this small video snippet was actually denoting that particular activity which is being shown ok.

Now it is it is not a frame by frame classification in any way, you would need to classify the whole volume in one single class label over there. Now ah; so what we do initially is we have this path where my videos are stored down over there. Now my first part is basically to find out and scheme all the file names of the files which are present over there. Now once you get your file names for all of these files coming down over there now what you can do is; you would need to just get down only 5 different classes present oh sorry one thing.

(Refer Slide Time: 03:37)




So once you scheme out all the file names over there the next part is basically you need to extract out the individual frames.

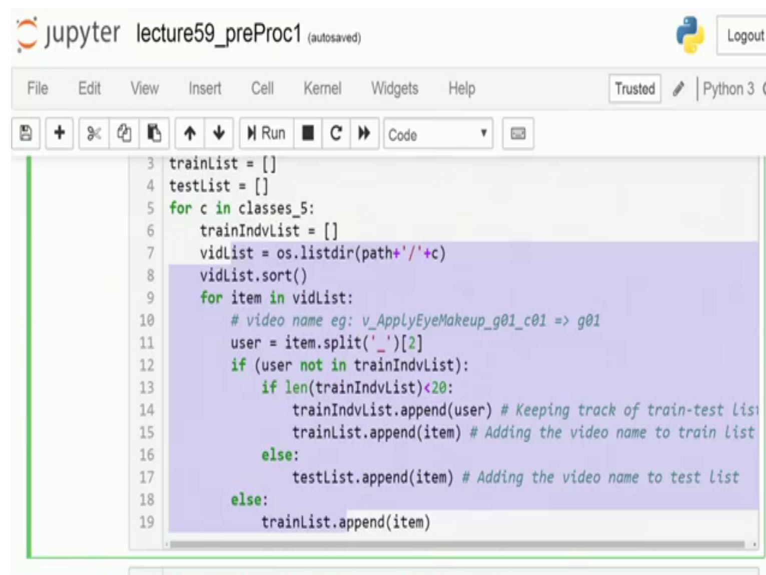
So a typical video how it is represented is your first axis is basically your time, your second axis your color channel, the third axis is x and fourth axis is y. Now you are

supposed to find out these 2D frames and extract it out and keep it over there. So that is this first part of it which is doing. So if there is a valid video which comes down over there then using this ffmpeg decoder unit. So it is it is available typically on a mp mpeg decoder encoding format over there. So we use this library over there in order to convert these videos onto my image frames and then store it down as a jpeg format over there.

Now once that is done the next part is to look down into only a very few specific classes which we are going to make use of. Now we are going to take only the first five classes; which are this and for one of the very simple reason is that if I am taking more number of classes then the granularity of the network is going to be very large. And the time complexity taken for training it is also going to make it complicated. So we are taking a smaller subset over there something in line with the other problems which we have been dealing till now.

So we have larger data sets in access, but we are just taking down a few of the classes only over there. And this is to make it much more easier and conducive for it to work out ok. So I am taking down 5 different classes and these are the 5 classes which are taken down ok.

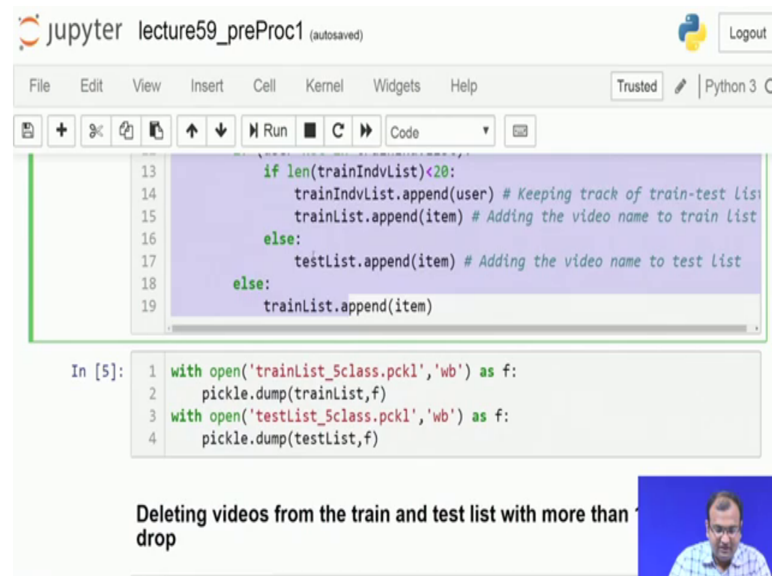
(Refer Slide Time: 05:10)




Now what I need to do is; I need to divide it into my training and test sets over there. And for that I have this part of the routine which is going down. Now what essentially it does is that you are going to scheme out a particular videos, some of these videos are

going to store down in your training set some of these are going to be part of your test set. But nonetheless there are no frames which are shared between the training set and your testing set. So whichever video is there on your training set remains on your training set whichever video is there on your testing set remains on your testing set over there.

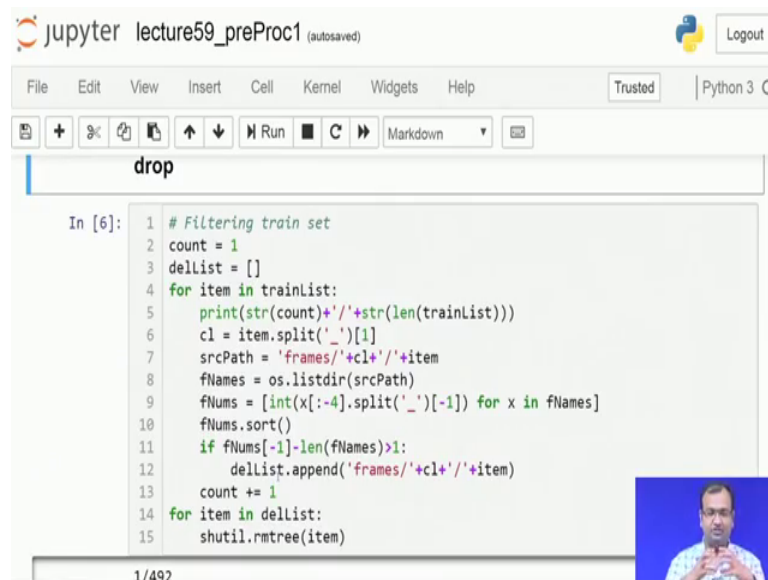
(Refer Slide Time: 05:40)




Now once that part of my program is ready which has all of these divided into 2 different sets over there. Then we just keep down these file name stored into 2 different ones. Now trainlist and testlist and this is how the reason that in the subsequent programs; I am going to make use of these lists in order to fetch out those file names as well as the class labels over there.

And that would help me in creating a data loader kind of a mechanism in order for everything else to work out. So that sets just to make use of as much as intrinsics predefined and available for my programs to run down in the minimum hindrance possible ok.

(Refer Slide Time: 06:19)




So next is what you are going to do is once you have extracted out these frames. Now certainly in certain videos you have these issues of missing frames over there. So it might be an encoding error or some sort of an error that the decoder is not able to read from there. Now if there are missing frames then we typically try to delete them. So they might be blank frames or corrupted frames over there. So depict to be deleted out over there and if you have a substantial number of frames which are missing from a video then it makes it problematic because the video might start getting skewed.

Ah So you have some perfect part of the action being recorded and then in between some of these frames are missing. So technically what it means is that if you look into your time domain samples over there. Then there is a non uniform sampling which has taken place over there. Now that you have these missing frames, so it creates a lot of problems coming in. So we just remove those videos from our training set as such in order to keep everything Euclidian and uniformly sampled out.

(Refer Slide Time: 07:12)

jupyter lecture59_preProc1 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

```
16/492
17/492
18/492
19/492
```

```
In [7]:
1  # Filtering test set
2  count = 1
3  testDelList = []
4  for item in testList:
5      print(str(count)+'/'+str(len(testList)))
6      cl = item.split('_')[1]
7      srcPath = 'frames/'+cl+'/'+item
8      fNames = os.listdir(srcPath)
9      fNums = [int(x[:-4].split('_')[-1]) for x in fNames]
10     fNums.sort()
11     if fNums[-1]-len(fNames)>1:
12         testDelList.append('frames/'+cl+'/'+item)
13     count += 1
```

So we repeat the same thing for our test set as well.

(Refer Slide Time: 07:14)

jupyter lecture59_preProc1 (autosaved)

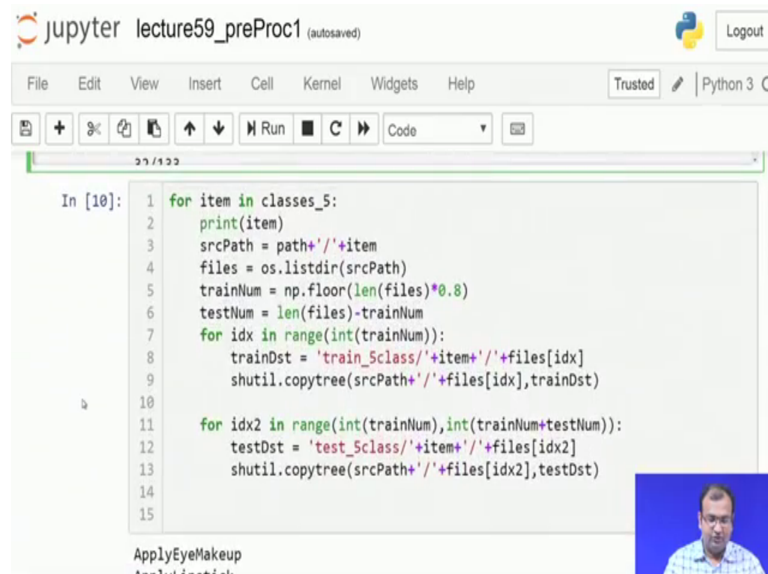
File Edit View Insert Cell Kernel Widgets Help

```
3  testDelList = []
4  for item in testList:
5      print(str(count)+'/'+str(len(testList)))
6      cl = item.split('_')[1]
7      srcPath = 'frames/'+cl+'/'+item
8      fNames = os.listdir(srcPath)
9      fNums = [int(x[:-4].split('_')[-1]) for x in fNames]
10     fNums.sort()
11     if fNums[-1]-len(fNames)>1:
12         testDelList.append('frames/'+cl+'/'+item)
13     count += 1
14 for item in testDelList:
15     shutil.rmtree(item)
```

```
1/133
2/133
3/133
4/133
5/133
```

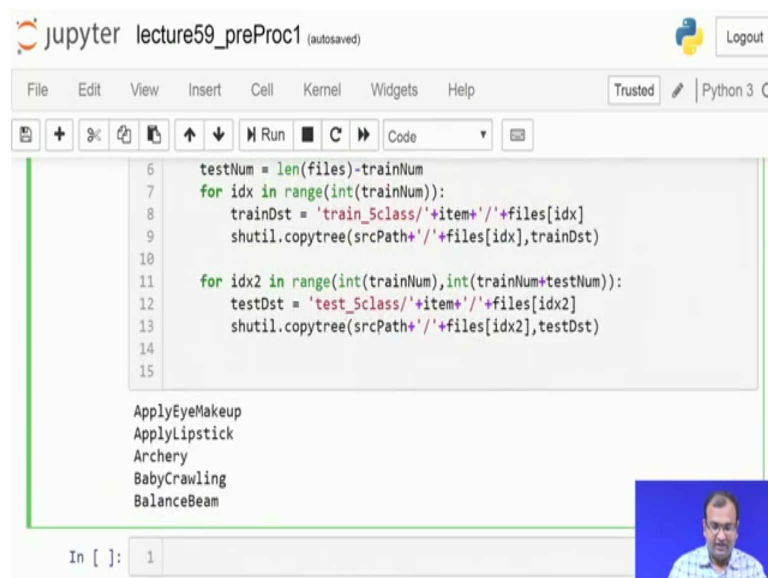
Over there also we are removing out videos which have some of these missing frames, so that we do not make a problem around while trying to infer from this one.

(Refer Slide Time: 07:24)



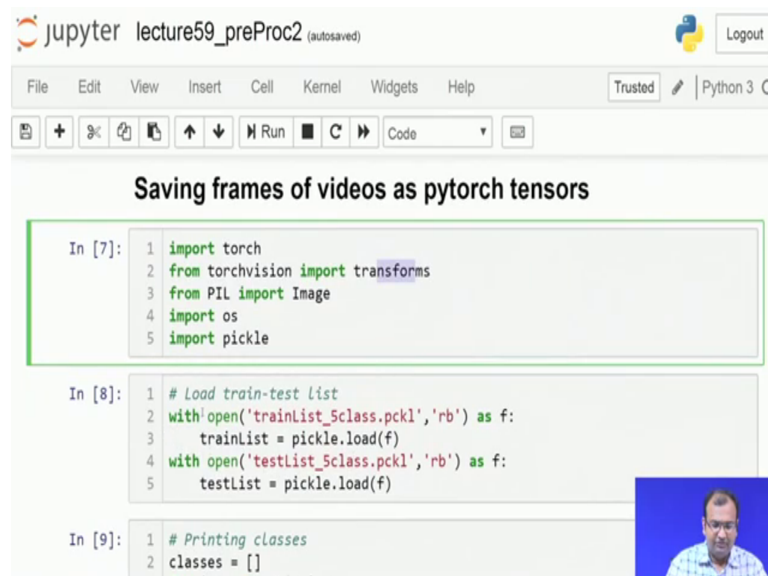

Now once that is done next what you do is; you have this stored into a separate directory structure in which you have your train dataset one folder and your test dataset within your train dataset you create 1 folder for each of these videos and you store all the frames corresponding to that video. So this is what we end up doing over here ok.

(Refer Slide Time: 07:47)




Now that creates my complete directory structure as well as that also creates my list available. So there is a file name list in terms of a small pickle file which I can use it on the next job over there.

(Refer Slide Time: 08:02)




Now with that we enter into the second one now what this second code is basically all about. In the first one we have stored everything in terms of images, and now we need to convert all of them to tensors. Because if you look into your 3D CNN; so what you have essentially is the first channel is your ah the first tensor dimension is basically your number of channels, the second tensor dimension is the time axis over there the third is x and fourth is y ok; now my frames are not stored in the same format.

Now one way I can do is within my training I can keep on loading everything and then do it, but that is going to make the whole process much more complicated. So instead of trying to do that and make it more complicated. We are simplifying in the whole process by storing each video in terms of one single tensor. If one single 4D tensor presentation such that you can just load this 4D tensor over there and then you can set your 3D CNN running in a perfect way.

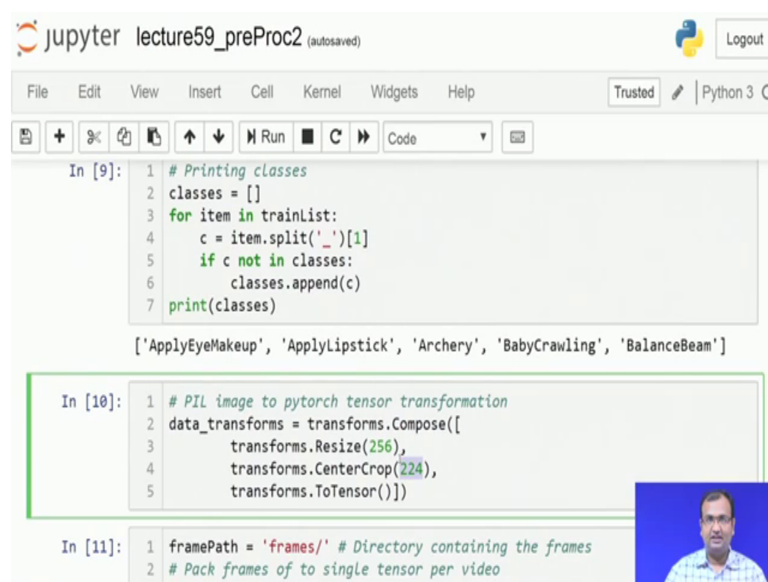
So let us get into what comes over there for this now says we need to make use of torch tensors. And store it as a torch tensor format for that reason we get the torch library and torch vision library over there. So torch vision is just for your transformation let us to make it down into a conformal representation for standard CNNS. Now what we do is essentially find out first the list of all the file names over there which I have in my train list and test list available.

So this is which frame belongs to train and which frame belongs to test. So that makes it easier for me to really scheme through the directory structure; instead of trying to rebuild

the directory every time. And also the other problem is that now that I have just a bunch of frames over there; the sequentiality between these frames is not stored, but that is a critical part.

Now this list over here in case of train list and test list is which has this sequentiality for a particular video stored. So like which frame is succeeding which frame and which precedes. So that when you are building up this 4D tensor; you can build it up in a much conformal way.

(Refer Slide Time: 10:02)




So what we do is first is read through it and find out whether my classes are present over there. So these are the five classes which we had stored initially and that is coming up over there, the next part is that when we are loading these ones we would like to apply a certain kind of a transformation. So first and foremost I would like to make it conformal, make all the image conformal to my original size for a image net kind of a problem, so whether your images were of size 224 plus 224.

So we are just going to do a center crop of 2 2 4 plus 2 2 4 and the other point is that all of these images which you are getting they are in some arbitrary size. Now the first point which we apply is resize them to 256 cross 256 and then crop out just the center 2 2 4 cross 2 2 4. And one of the reasons for doing this is quite simple that once you have this whole thing resized over there and you crop out the center. So you are going to reduce all the peripheral pixels over there.

The moment you are reducing and chunking out and throwing away all of these peripheral pixels. So an advantage which you get is that you do not have any of these boundary conditions coming into play. Because most of these videos have the activity which is more of centered over there and these extra side peripheral pixels are which are going to ah cascade down when you are having a deep convolutional neural network and that is a useless information. So most of your useful information might die out by the time you are still at the final depth over there.

So this helps in doing it and the other part is that you can because anyways you could have resize it to 2 2 4 cross to 2 2 4 at the first place, but the whole reason was that you wanted to keep as much as possible the most fruitful information which for these kind of activity videos is present in the center of the image. So we just keep it over here that makes it easier for us ok.

(Refer Slide Time: 11:42)

```
framePath = 'frames/' # Directory containing the frames
# Pack frames of to single tensor per video
for item in trainList:
    cName = item.split('_')[1]
    srcPath = framePath+cName+'/'+item
    fNames = os.listdir(srcPath)
    fTemplate = fNames[0].split('_')
    fCount = len(fNames)
    for fNum in range(fCount):
        fileName = fTemplate[0]+'_'+fTemplate[1]+'_'+fTemplate[2]+'_'+fTempl
        if os.path.exists(srcPath+'/'+fileName):
            # Load image
            img = Image.open(srcPath+'/'+fileName)
            # Transform to tensor
            imgTensor = data_transforms(img).unsqueeze(1)
            # Stack tensors per video. trainTensor dimension: 3xframe_countx
            if fNum == 0:
                trainTensor = imgTensor
            else:
```

The next part over here is to start loading each of these frames and then apply the transform over there. So once you have this transformation applied then you convert it on to a tensor.

(Refer Slide Time: 11:53)

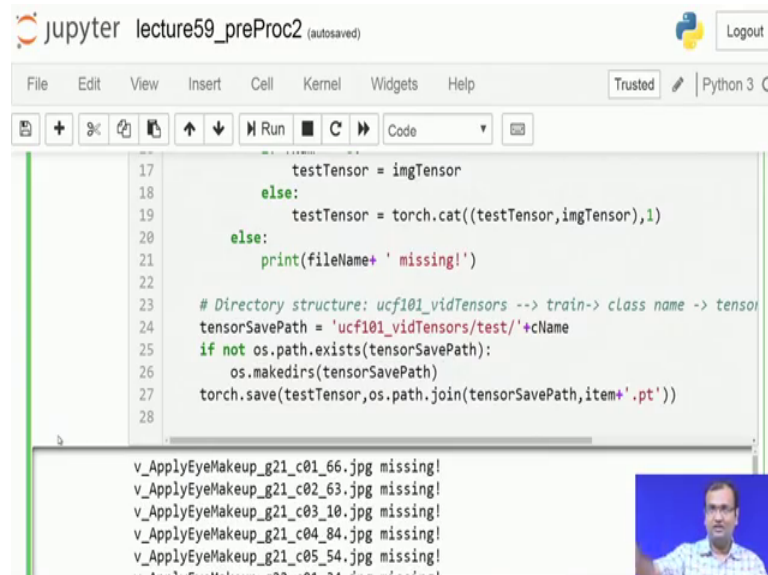
So after applying this transformation what you have basically is these axis changed over there. So your color becomes your first channel instead of your frame number when you are reading down over there. And the second channel is your frame number and then you try to concatenate each of them along the dimension of the frame number.

And so that you have your temporal domain concatenation coming down over there, now once that is done what we additionally do is that some of these frames might be missing over there and ah. We just basically it is it is a printing of it that which all frames are missing that the transformation has not been applied on to those particular frames over there ok.

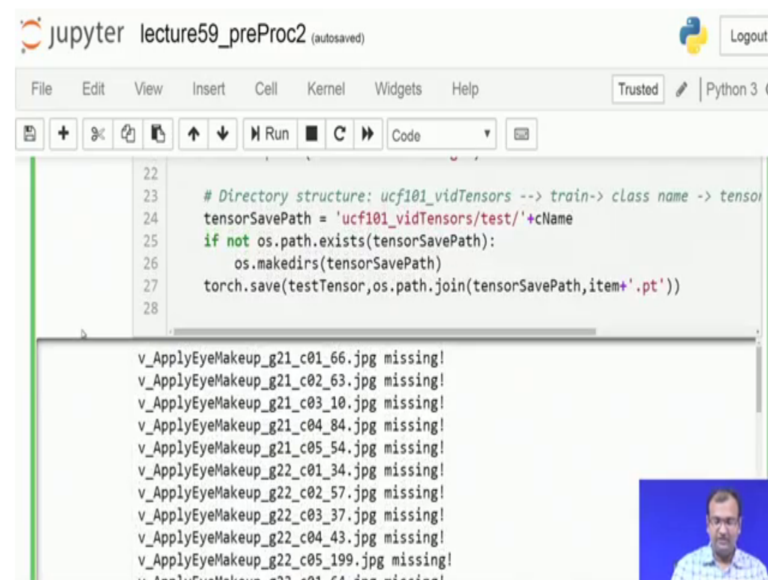
(Refer Slide Time: 12:35)

Now the same thing is applied onto your test list as well in order to create a tensor for video in your test 1

(Refer Slide Time: 12:39)

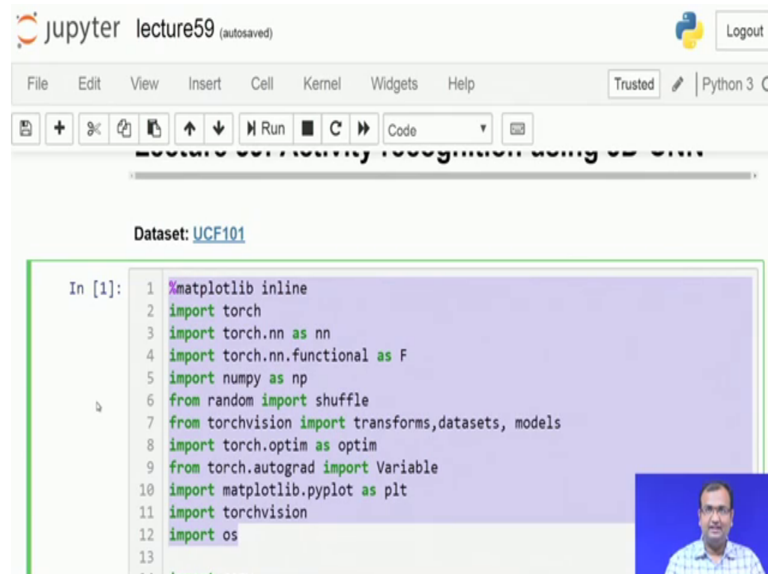



(Refer Slide Time: 12:44)



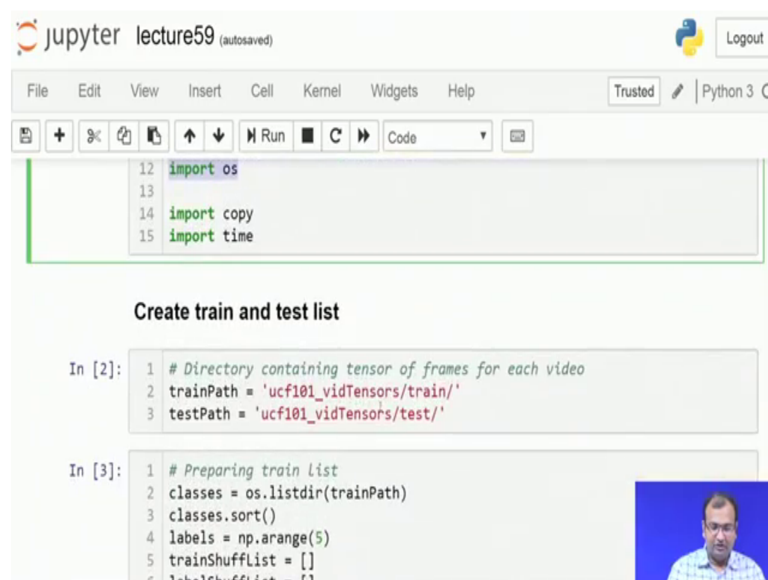

So this is where we finish off with creating a 3D tensor out of your standard frames which have been extracted out ok.

(Refer Slide Time: 12:54)



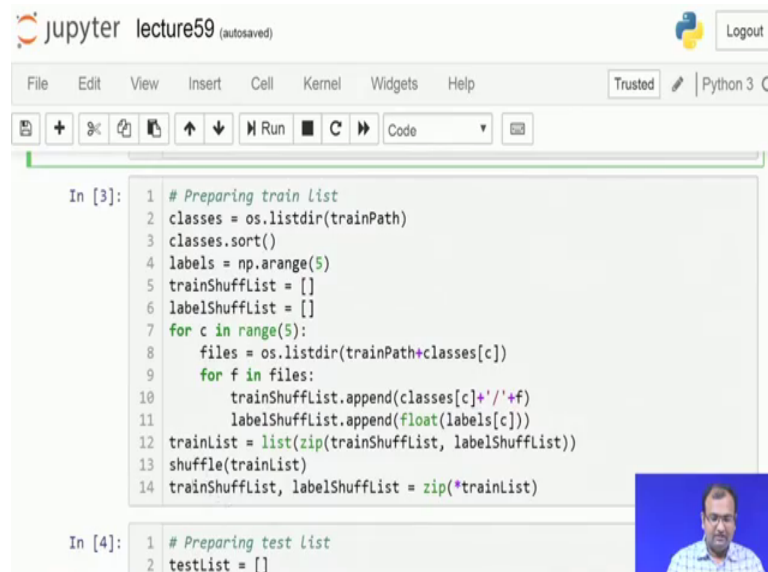

Now since the data handling part is over here is where we get into our actual learning mechanism. So the first part of the headers is pretty straightforward and it said that there is no change as such which comes down from a 2D CNN to a 3D CNN most of the math and the linear algebra over there being the same the headers also which we are taking down are all those in ok.

(Refer Slide Time: 13:14)




Now your videos are now stored in terms of a tensor and they are stored inside these 2 different directories one is for your train and another is for your test ok.

(Refer Slide Time: 13:23)




Next what I am going to do is we need to have some sort of a shuffling written down over here. Now keep in mind think that every epoch we are supposed to have a shuffled out variant coming out over here. And every say every time I do a run it is not supposed to be in the same order. So if I have my first video, second video, third video, fourth video any of these then there is supposed to be some sort of a shuffling between their orders in which it comes out.

So that is what we take care over here using this shuffling function; one of the problems is that you could not use a data loader in order to do this shuffle is because you have this not in terms of images anymore where data loader technically works, so you have a separate tensor which you are pulling it down. And the next part is that whatever you need to shuffle; you will have to shuffle down your list as well as the training video. And for that reason the simple trick over here which works out is that zip both these folders together. So you have a tensor and you have another say up torch tensor which has just the label over there.

So you have a scalar and a tensor something over there zip it together. So you have a zips of these available and then you can shuffle up this zip. And then when you unzip it out you basically get this as a tuple. So your class label and the corresponding video are now together in and shuffled out in their order which comes in.

(Refer Slide Time: 14:42)

jupyter lecture59 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
14 trainShuffList, labelShuffList = zip(*trainList)
```

```
In [4]:
1 # Preparing test list
2 testList = []
3 testLabelList = []
4 for c in range(5):
5     files = os.listdir(testPath+classes[c])
6     for f in files:
7         testList.append(classes[c]+'/'+f)
8         testLabelList.append(float(labels[c]))
```

**Define network architecture**

```
In [5]:
1 class net_3DCNN(nn.Module):
2     def __init__(self):
3         super(net_3DCNN, self).__init__()
```

So the next part is basically prepare your test list and what this test list is actually trying to do is that from your test. So this was what you applied on to your train data over there and you have to do a similar thing for your test data to have this as a tupled out variable, but you do not need any kind of a shuffling anymore over here; so that that is not any a major issue.

(Refer Slide Time: 15:04)

jupyter lecture59 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [5]:
1 class net_3DCNN(nn.Module):
2     def __init__(self):
3         super(net_3DCNN, self).__init__()
4         self.conv1 = nn.Conv3d(3, 16, kernel_size=5)
5         self.pool1 = nn.MaxPool3d(kernel_size=2,stride=2)
6         self.conv2 = nn.Conv3d(16, 32, kernel_size=3)
7         self.pool2 = nn.MaxPool3d(kernel_size=2,stride=2)
8         self.conv3 = nn.Conv3d(32, 32, kernel_size=3)
9         self.pool3 = nn.AvgPool3d(kernel_size=4)
10        self.fc = nn.Linear(32*13*13,5)
11
12    def forward(self, x):
13        x = F.relu(self.conv1(x),inplace=True)
14        x = self.pool1(x)
15        x = F.relu(self.conv2(x),inplace=True)
16        x = self.pool2(x)
17        x = F.relu(self.conv3(x),inplace=True)
```

Now we come down to the definition of the 3D CNN. Now the changes which you would see is quite evident, so instead of conv 2D which we were using in case of a 2D convolution, now we replace it down with 3D conv over here ok. Now your input number of channel still stays the same as the number of channels over here. So you have

a 3 color image over there. So it is it is 3 the number of convolution kernels over here has been made 16. So a pretty straightforward way of doing it out and the convolution kernels are of size 5 cross 5 ok.

Now instead of a 2D max pool now you will have a 3D max pool; this 3D max pool does on a kernel size of 2 cross 2 with a stride of 2. So it is going to do this striding and max pooling along the x axis y axis as well as along the time axis on all the 3 axis. So now your max pooling volume or the earlier you had a on a 2D you had a max pooling kernel of 2 cross 2 here you are going to have a volume over there. And this is of 2 cross 2 cross 2 and that is going to be with a stride of 2 on each of these dimensions on which it will be doing a max pool.

Ah Following that I have my second 3D convolution layer coming down now since the number of channels on the output of my first convolution is 16 my ah max pooling does not impact the number of channels over there. So now I take down 16 channels as my input, I generate thirty 2 channels on my output. And I have a kernel of size 3 cross 3 now after that I again do a max pooling with a kernel size of 2 cross 2 and as stride of a sorry kernel size of 2 cross 2 cross 2 and a stride of 2 comma 2 comma 2 then I again have a 3D convolution. And I convert thirty 2 channels to get me 3 2 channels with a kernel size of 3 cross 3.

And then after that we employ an average pooling instead of a max pooling and this average pooling is with the kernel size of 4. So this will put me a 4 cross 4 average pooling, but then I do not have a stride in place. So it means that there is a stride of 1 which is taking place over here to basically have an averaging done together, And finally, you would be getting down 32 cross 13 cross 13 number of pixel locations or number of volumes over here. And that has to be mapped onto 5 neurons. So that is my linear stage which applies over there.

So my final classification is just a 5 classification problem and for that reason you just have 5 neutrons over here.

(Refer Slide Time: 17:22)

jupyter lecture59 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
4           self.conv1 = nn.Conv3d(3, 16, kernel_size=5)
5           self.pool1 = nn.MaxPool3d(kernel_size=2,stride=2)
6           self.conv2 = nn.Conv3d(16, 32, kernel_size=3)
7           self.pool2 = nn.MaxPool3d(kernel_size=2,stride=2)
8           self.conv3 = nn.Conv3d(32, 32, kernel_size=3)
9           self.pool3 = nn.AvgPool3d(kernel_size=4)
10          self.fc = nn.Linear(32*13*13,5)
11
12      def forward(self, x):
13          x = F.relu(self.conv1(x),inplace=True)
14          x = self.pool1(x)
15          x = F.relu(self.conv2(x),inplace=True)
16          x = self.pool2(x)
17          x = F.relu(self.conv3(x),inplace=True)
18          x = self.pool3(x)
19          x = self.fc(x.view(x.size(0),-1))
20          return x
```

So the forward pass of it is defined in a similar way. So you have your first convolution in implied then you have your max pooling the first pool operation then you have and ah. So you have your first convolution applied then you have your relu as a non-linear transformation function; then you have your pooling. Now the output of that one is again convolved you have your relu in place you have another pooling and similarly it keeps on going till the last layer over there.

So on the last layer you just have your fully convolutional layer coming down.

(Refer Slide Time: 17:50)

jupyter lecture59 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

## Define train routine

```
In [6]:
1   def train(net, inputs, labels, optimizer, criterion):
2       net.train(True)
3       inputs, labels = Variable(inputs).float().cuda(), Variable(labels).cuda(
4       outputs = net(inputs)
5   #     print(outputs.size())
6       _, predicted = torch.max(outputs.data, 1)
7       # Initialize gradients to zero
8       optimizer.zero_grad()
9       # Compute loss/error
10      loss = criterion(F.log_softmax(outputs), labels)
11      # Backpropagate loss and compute gradients
12      loss.backward()
13      # Update the network parameters
14      optimizer.step()
15      correct = (predicted.cpu() == labels.data.cpu()).sum()
16      return net, loss.data[0], correct
```

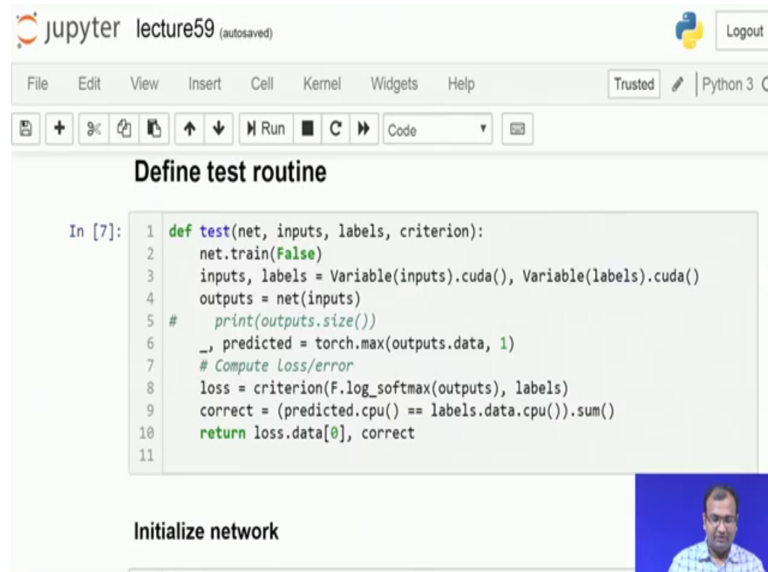
So having done that ah; we write down our training routine over here,so this is quite different from what we have been doing in the earlier cases. In most of our earlier cases what we have been doing is that we have been writing this training function over there directly inside my train routine over there, but here we choose to just make this as a separate function call separately which is outside over there ok.

Now what we do is we need to convert all of my inputs and labels on to variables. So that is this typecasting which we are doing over here. And if we have a GPU available then there is going to be a cuda type conversion over. There if I do not have a GPU available and that is something I can skip out totally. The next is that you put your inputs onto the network and then you are going to get your outputs and from there you can find out what is your predicted class ok.

Now once you have your predicted class found out and the forward pass already solved out. So you can zero down the gradients on your optimizer and get your criterion function or the loss function calculated. So here what we do is since like we are going to have a classification problem. And we would stick down to using the negative log likelihood criteria for that person purpose we are putting a log softmax on the output over there for it to be conformal.

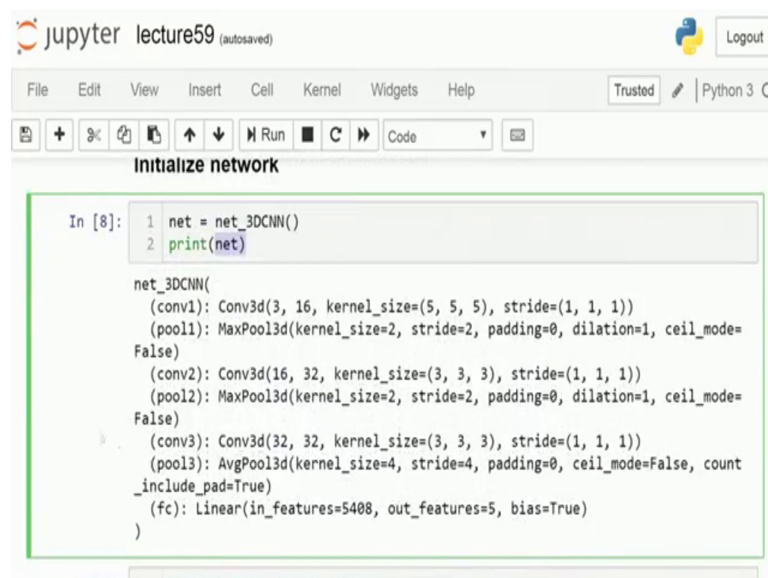
So I get my loss calculated and then I can do a nabla of loss or the gradient of my loss found out then my optimizer dot step which is an update rule over there and then find out the total number of corrected ones over there or the number of correct classifications coming out of this network ok.

(Refer Slide Time: 19:24)




So similarly I have my test routine define; the only different difference in my test function over here is that I do not have this back ward calculation in either I just do a forward pass over there and find out what is my loss and the total number of correct classifications which it was supposed to do.

(Refer Slide Time: 19:38)




Next I initialize my network and then this is just a printed version of the network which comes over here.

(Refer Slide Time: 19:45)

jupyter lecture59 (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3

```
In [9]:
1 # Check availability of GPU
2 use_gpu = torch.cuda.is_available()
3 if use_gpu:
4     print('GPU is available!')
5     net = net.cuda()

GPU is available!
```

**Define loss function and optimizer**

```
In [10]:
1 criterion = nn.NLLLoss() # Negative Log-likelihood
2 optimizer = optim.Adam(net.parameters(), lr=1e-4) # Adam
```

**Train the network**

So after doing all of this we can now get started with our actual job. So given I if I have a GPU available then just check out if it is there.

(Refer Slide Time: 19:55)

jupyter lecture59 (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3

```
4     print('GPU is available!')
5     net = net.cuda()

GPU is available!
```

**Define loss function and optimizer**

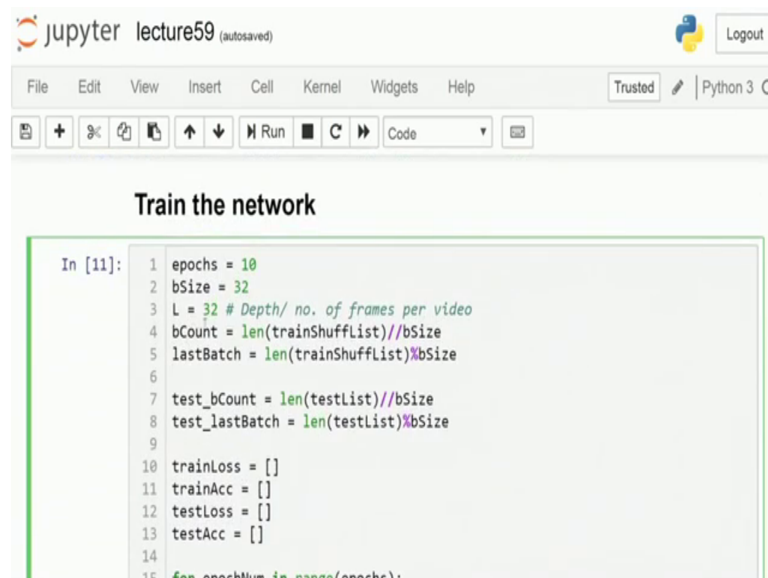
```
In [10]:
1 criterion = nn.NLLLoss() # Negative Log-likelihood
2 optimizer = optim.Adam(net.parameters(), lr=1e-4) # Adam
```

**Train the network**

```
In [11]:
1 epochs = 10
2 bSize = 32
3 L = 32 # Depth/ no. of frames per video
```

And then you can initialize your criterion or the loss function and your optimizer both of them.

(Refer Slide Time: 20:01)

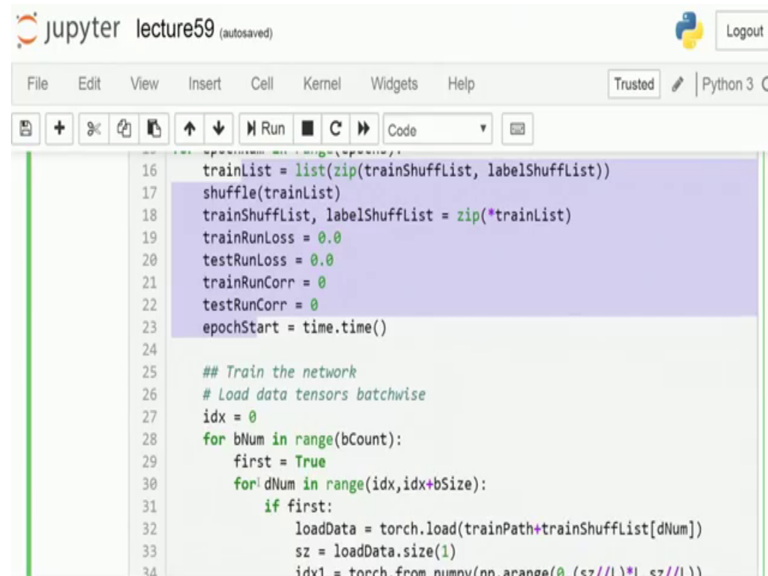



Now with.all of these initialize you can start with your training of the network. Now one extra parameter which we add over here is basically the number of frames per videowhich you are going to take down then that is supposed to be fixed because you have your x and y dimension fixed which is at 2 2 4 cross 2 2 4 your number of time access dimensions is no more fixed it was not given down for me.

Now we are going to take down just 32 time frames over there and not more than that we are not taking 224 time frames in any way ok. Now once we have that we can actually find out what is the total number of batches which it will fit down because I have my batch size defined over here. And then find out what is the size of my last batch and the reason for doing this is that it might my total number of frames are available to me in order divide into batch may not be in some way integral multiple of 32 or my batch size.

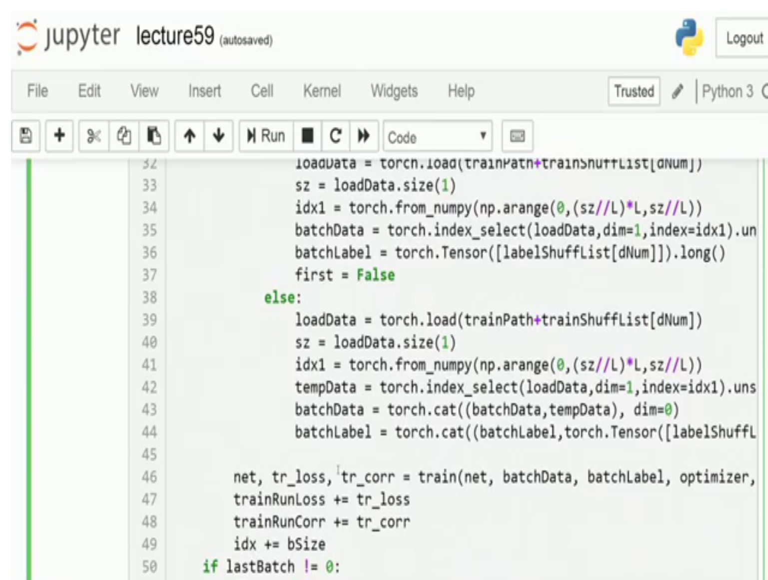
So some frames will be left out I do not want to leave them out from my whole training over there. So this last batch is just an exception added to take care of all of those remaining residual frames over there. Now I do the same thing for my test dataset as well; now once I have it then I can start my running of the training part of the network.

(Refer Slide Time: 21:11)




So we will start within one batch I would load my data I will convert it and apply whatever my transformations are supposed to be applied and then I will have that one as a forward pass through my network.

(Refer Slide Time: 21:24)




Now this forward pass of through the network is something which is implemented on the train routine which we had defined earlier. Now once that is done I get down my losses and my total actual correct version over there and the updated network also coming out of it.

(Refer Slide Time: 21:42)

jupyter lecture59 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
51            first = True
52            for dNum in range(idx,idx+lastBatch):
53                if first:
54                    loadData = torch.load(trainPath+trainShuffList[dNum])
55                    sz = loadData.size(1)
56                    idx1 = torch.from_numpy(np.arange(0,(sz//L)*L,sz//L))
57                    batchData = torch.index_select(loadData,dim=1,index=idx1).un
58                    batchLabel = torch.Tensor([labelShuffList[dNum]]).long()
59                    first = False
60                else:
61                    loadData = torch.load(trainPath+trainShuffList[dNum])
62                    sz = loadData.size(1)
63                    idx1 = torch.from_numpy(np.arange(0,(sz//L)*L,sz//L))
64                    tempData = torch.index_select(loadData,dim=1,index=idx1).uns
65                    batchData = torch.cat((batchData,tempData), dim=0)
66                    batchLabel = torch.cat((batchLabel,torch.Tensor([labelShuffL
67            net, tr_loss, tr_corr = train(net, batchData, batchLabel, optimizer,
68            trainRunLoss += tr_loss
69            trainRunCorr += tr_corr
```

Now with that one now I can so this is just for the last batch over there then I can within each epoch my testing part as well.

run down.

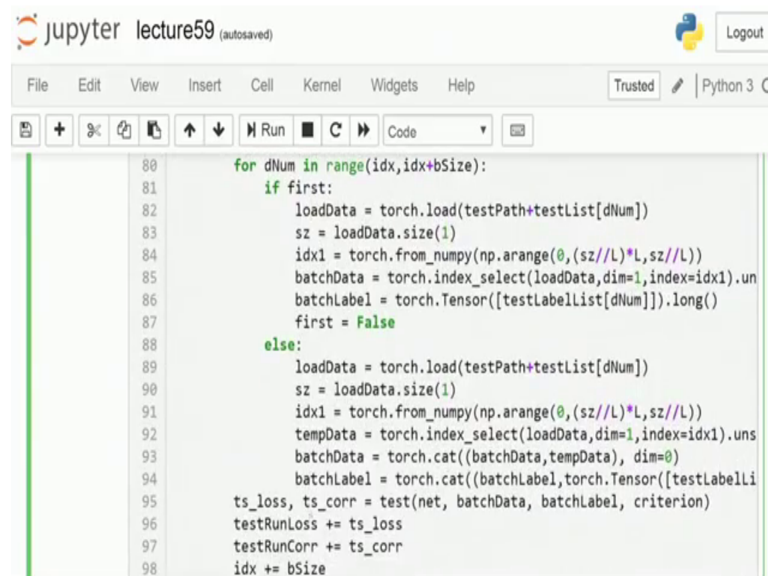
(Refer Slide Time: 21:45)

jupyter lecture59 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

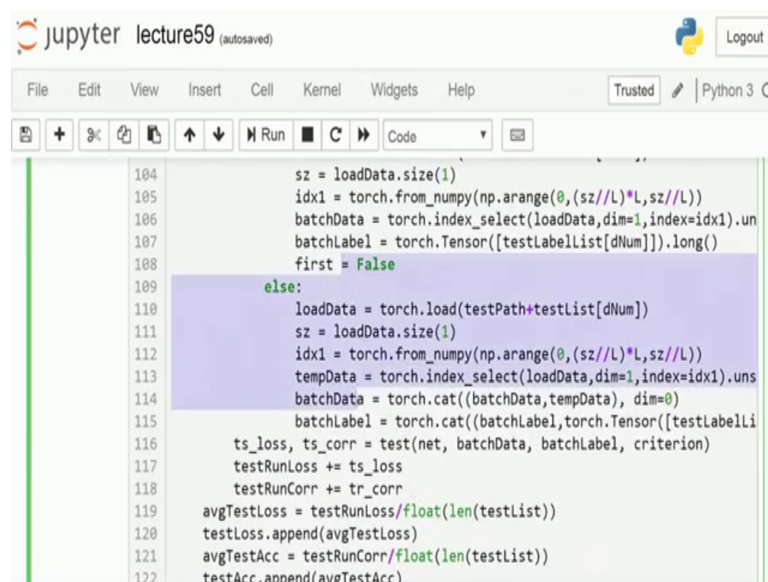
```
68            trainRunLoss += tr_loss
69            trainRunCorr += tr_corr
70        avgTrainLoss = trainRunLoss/float(len(trainShuffList))
71        trainLoss.append(avgTrainLoss)
72        avgTrainAcc = trainRunCorr/float(len(trainShuffList))
73        trainAcc.append(avgTrainAcc)
74
75        # Test the network
76    #     Load data tensors batchwise
77        idx = 0
78        for bNum in range(test_bCount):
79            first = True
80            for dNum in range(idx,idx+bSize):
81                if first:
82                    loadData = torch.load(testPath+testList[dNum])
83                    sz = loadData.size(1)
84                    idx1 = torch.from_numpy(np.arange(0,(sz//L)*L,sz//L))
85                    batchData = torch.index_select(loadData,dim=1,index=idx1).un
86                    batchLabel = torch.Tensor([testLabelList[dNum]]).long()
```

(Refer Slide Time: 21:48)




Now in the test part it is straightforward that you are going to do a feed forward over the whole network. And that gives you what is the testing performance coming out over there.

(Refer Slide Time: 21:57)




Similarly you have the exception added for the last batch in order to take care of the remaining frames. And then you have your routine for plotting all of these losses and then you can get it started.

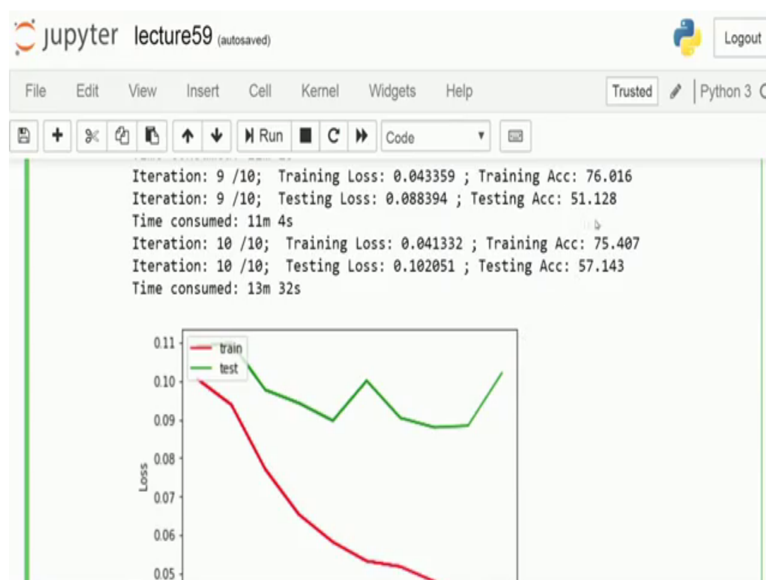
(Refer Slide Time: 22:08)

```
/home/deepsip/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:10: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to include dim=X as an argument.
  # Remove the CWD from sys.path while we load stuff.
/home/deepsip/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:8: UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the call to include dim=X as an argument.

Iteration: 1 /10;  Training Loss: 0.100434 ; Training Acc: 23.374
Iteration: 1 /10;  Testing Loss: 0.109071 ; Testing Acc: 24.812
Time consumed: 9m 17s
Iteration: 2 /10;  Training Loss: 0.093812 ; Training Acc: 38.211
Iteration: 2 /10;  Testing Loss: 0.109884 ; Testing Acc: 30.075
Time consumed: 10m 9s
Iteration: 3 /10;  Training Loss: 0.077095 ; Training Acc: 51.626
Iteration: 3 /10;  Testing Loss: 0.097697 ; Testing Acc: 42.105
Time consumed: 10m 21s
```

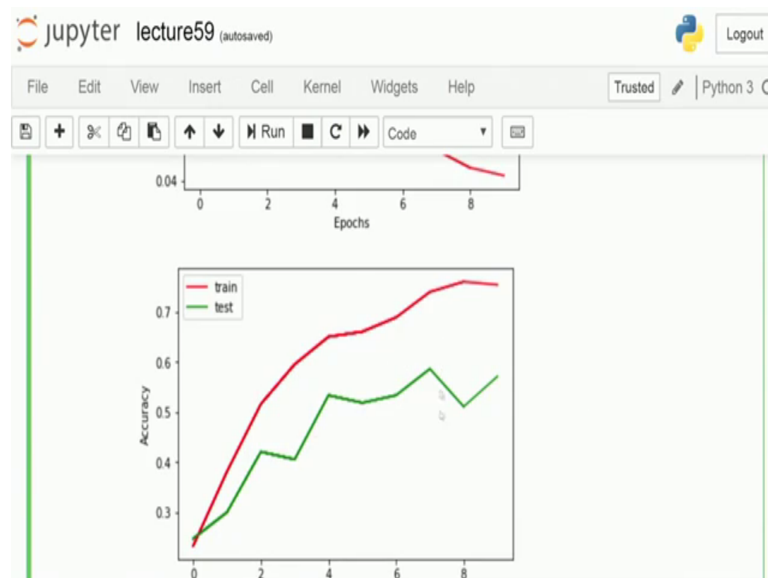
So we have trained this one for ten epochs it starts with an training accuracy of about 23 percent and a testing accuracy of 24 percent and then keeps on rising as it keeps on going.

(Refer Slide Time: 22:17)




Where at the end of a 10th epoch you have a training accuracy or 75 percent and the test accuracy of about 57 percent. So these are the 2 loss curves which comes up and these are the accuracy curves.

(Refer Slide Time: 22:17)




Now if you look into this training part as well as the testing you can pretty much see that it is not yet at the convergence part over there. So that is still dwindling if you keep on running this for a longer period of time that will definitely get over. And come down to a convergence the only downside is that every iteration is going to take you some more amount of time than any of the other networks.

So it takes roughly about the 11 minutes in order to train for frame. And per like per epoch over there and one of the reasons for this one is that that tensor volume which you are handling down and the dense amount of operations in terms of 3D convolutions going is much higher.

And for that very specific reason it consumes a lot more or time um. When you do these parameter calculations you would also find out the total number of parameters is much higher than in case of a 2D CNN. The total number of a mathematical operations, you would do over here is also much higher. So under these conditions is why it takes more amount of time over there; nonetheless this is video.

So you have to bear with the complexity and the challenges faced over here. So that is where we come to an end about handling videos for classification using a 3D CNN. And the next class we are going to do our part with trying to use a recurrent neural network in order to see if this can be brought down. If the complexity can be brought down even further lower. So till then stay tuned and.

Thanks.