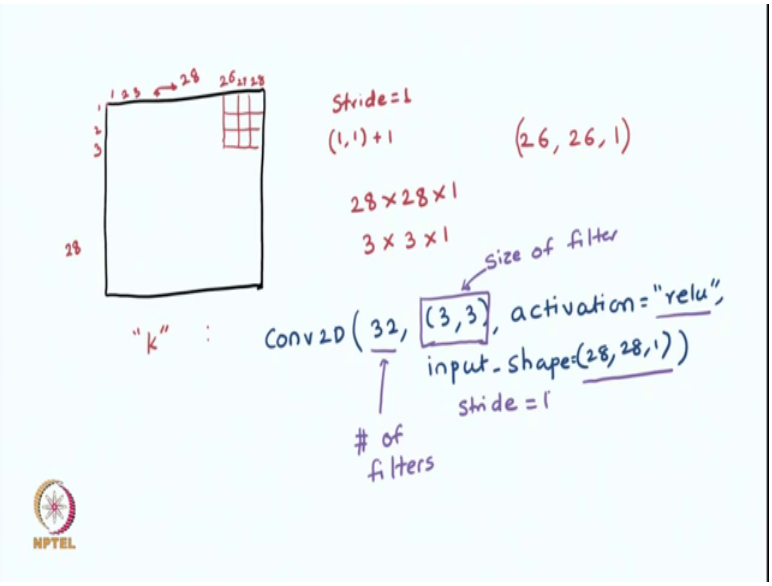
Practical Machine Learning
Dr. Ashish Tendulkar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 21
Convolutional Neural Network: Part 2

[FL] In the previous session, we studied convolution operations. In convolution operation we defined a bunch of filters and we use each filter to calculate activation at a particular point in the image. In this session, we will study how to slide each filter across image by setting strides. The output of convolution operation is usually smaller than the size of the input image. In order to keep the output of the same shape as the input we use padding. We will study padding in this session.

Apart from convolution there is second important operation in CNN which is called pooling. pooling aggressively downsamples the output of the convolution. After understanding pooling and convolution operation will use them in a practical setting to classify handwritten digits from MNIST dataset let us begin.

(Refer Slide Time: 01:31)



We said strides while sliding across the image, it provides a way to calculate the next position along each axis to position the filter starting from the current position. By

default we take stride = 1 which is the most common choice. We can specify different strides across different access, but we use that particular thing quite rarely. Such a strided convolution tends to down sample the input by the factor proportional to the stride. So, let us try to understand how the stride works.

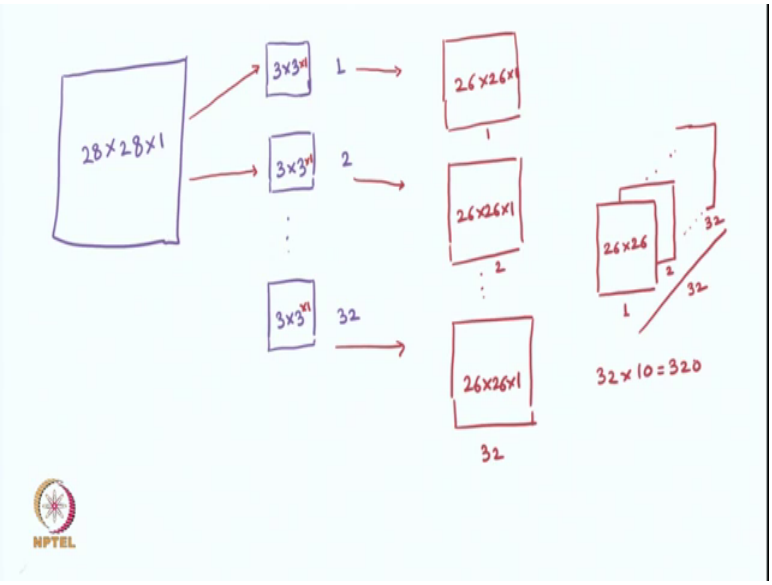
So, this is our 28 x 28 handwritten digit image and we have a filter positioned at the initial position of the image. So, we use a stride of 1. Ithelps us to calculate the next position of the filter. So, the current position is (1, 1). So, we will essentially take a stride of 1 in this particular direction. The next position of the filter will be. The filter got shifted by one column to the right. Once it exhaust all the columns we start shifting it downwards by rows. This is how we slide the filter across the image and try to match the pattern in the image.

So, let us say we have 28 x 28 x 1 image and we have filter of size 3 x 3 x 1. So, you can see that we will be able to position the filter at 26 possible positions along the width as well as on the height. So, the final position of the filter will be at position 26. So, this is how we get 26 x 26 x 1 output of the convolution.

So far, we looked at how to perform a convolution with a single filter. In a convolution layer we typically used *k* different filters. We define all these filters with Conv2D layer in a keras command. In a tf.keras API, we use Conv2D. We specify the number of filters, the size of the patch, the activation and the input shape. Here, we have 32 filters; each filter is of size 3 x 3. Then we specify the activation that we want to use after linear combination of weight of the filter with the values of the pixel in the image and finally, we specify the shape of the input.

The size of the filter and the stride (which is 1 by default) is applicable across all the k filters. After applying convolution of *k* filters we get 3D tensor with the same number of rows and columns for each filter.

(Refer Slide Time: 07:02)



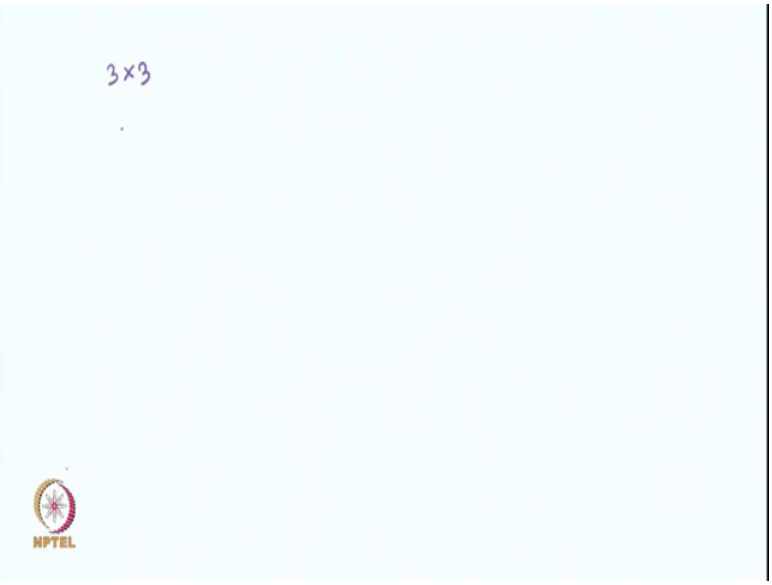
So, let us say this is our handwritten digit image which is 28 x 28 x 1. Let us say we apply 32 filters each of size 3 x 3. When we apply each of this filter to the images we essentially get 26 x 26 x 1 as an output. Since the filter has the same depth as the input so, here the depth is 1. The depth is normally not specified in the filter size.

All the outputs are combined. So, we get number of channels to be 32; each having 26 x 26 output.

So, concretely for our MNIST example we get 3D tensor as an output with 26 rows 26 columns and 32 channels, 1 for each filter. The total number of parameter for this filter will be 320 because we have 10 parameters per filter.

If you want convolution output to have the same shape as the input we use padding. So, after applying the convolution Conv2D the input gets shrunk by some amount. If you do not want our input to get shrunk we use padding where we add some dummy columns and rows to the input and then apply convolution on it.

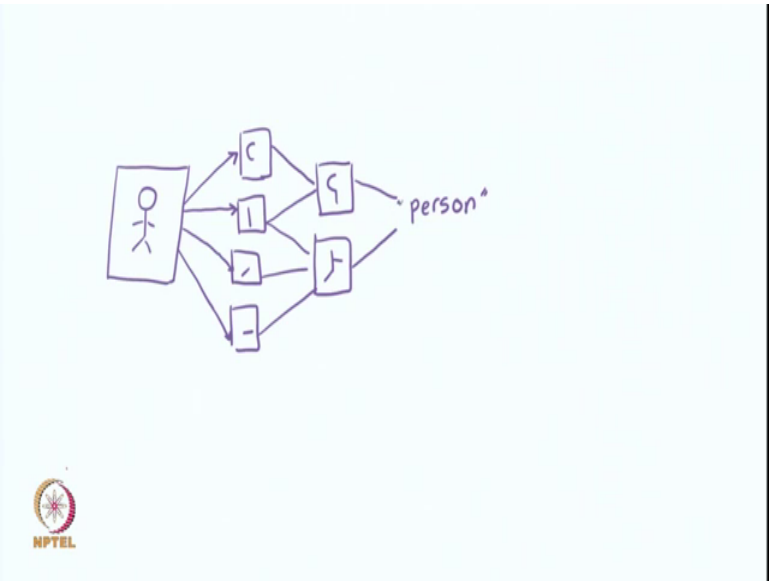
(Refer Slide Time: 10:57)



So, for a convolution with filter of size 3 x 3 we add one dummy row and one dummy column to the left and right and top and bottom. We add a dummy column to the left and to the right and a dummy row at the bottom and at the top of the image. This ensures that we have the shape of the output same as the shape of the input.

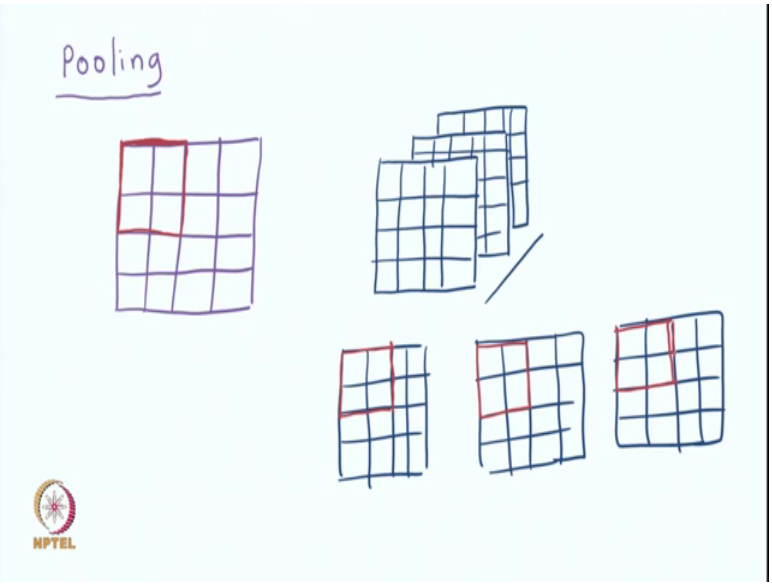
Convolution have a couple of key characteristics. The patterns that they learn are transition are translation invariant. After learning a certain pattern convolution neural network can recognize it anywhere. They can learn spatial hierarchies of the pattern. A first convolution layer we learn small local patterns such as edges, a second inclusion layer we learn larger patterns made from features from the first layer and so on. This allows us to efficiently learn increasingly complex and abstract visual concepts.

(Refer Slide Time: 12:15)



Let us say we have a convolution neural network that is trying to recognize a human from the image. The first convolution might capture some simple patterns like edges. In the second convolution layer we combined output of the first layer and we learn even more complex patterns and as you progress we learn increasingly complex patterns based on the output of the previous layers. Finally, these patterns will help us to detect a person in the image.

(Refer Slide Time: 14:00)

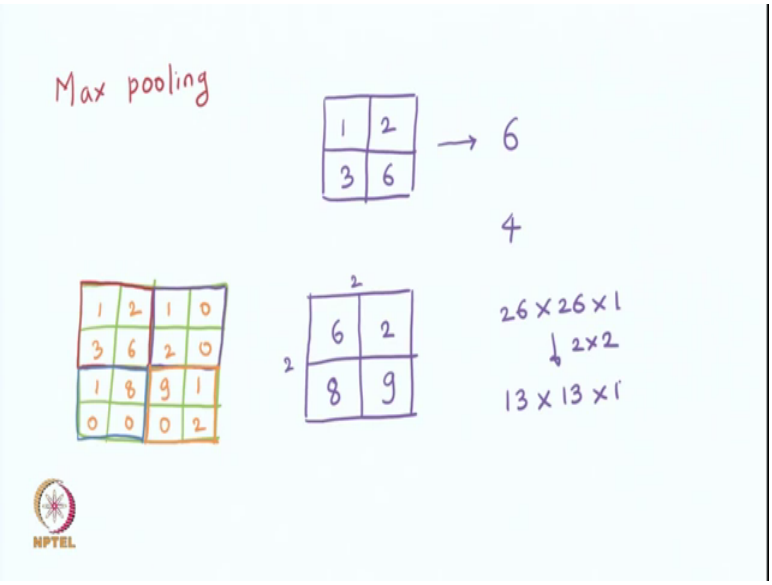


The second important concept in convolutional neural network is pooling. Pooling tends to downsample the output of the convolution aggressively. It is conceptually similar to the strided convolution. It consists of extracting a specific window from the input feature and compute the output based on the pooling policy. Pooling is usually done with a window of size 2 x 2 with stride of 2. Let us look at pooling with a concrete example.

We apply the pooling policy on the first 2x2 square box and select a number based on that policy. We use either max pooling or average pooling as the pooling policies. Second important point is we apply pooling operation on each channel separately.

For example, if we have the output of convolution. So, let us say the output of convolution is in multiple channels. While pooling what we do is we apply pooling on each channel separately. So, here we will separate the channels and we select one of we select one number from this particular patch based on the pooling policy.

(Refer Slide Time: 17:26)



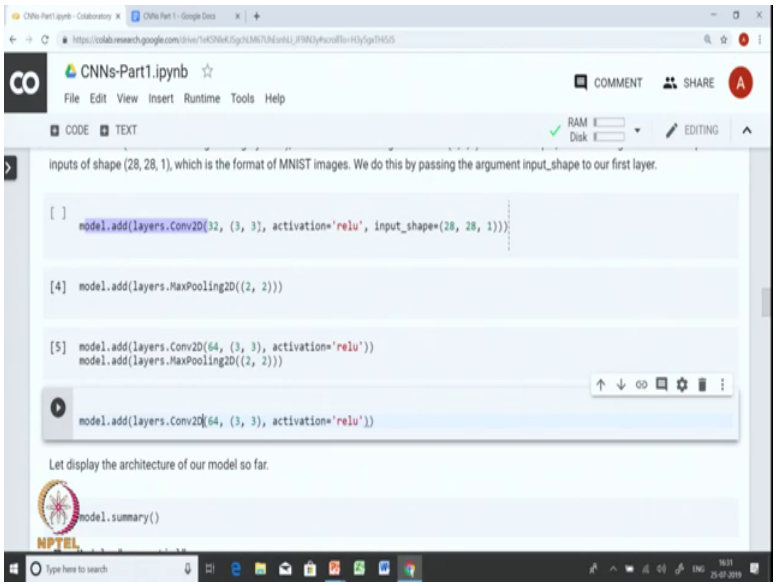
We use one of the following two pooling policies the first policy is called max-pooling and the second is called average-pooling. So, let us say these are four numbers (in the above image) in the max pooling window. The max pooling will return 6 which is maximum of these four numbers whereas, average pooling will return the average of these four numbers which is 4.

So, if we try the max pooling of in action on the output of the convolution layer. Each of these windows will return one number based on the pooling policy.

So, we get 2 x 2 output on the 4 x 4 input using 2 x 2 window of max pooling. We get $\begin{bmatrix} 6 & 2 \\ 8 & 9 \end{bmatrix}$. Similarly, if you are using average pooling as a policy we will compute average of the numbers in the patch. Let us say output is 26 x 26 x 1 and if you apply a max pooling of 2 x 2 we get the output of 13 x 13 x 1.

So, you can see that there is a down sampling happening from the output of the convolution when we apply max pooling on it. Note that max pooling does not have any parameters. In practice we set up a series of convolution and pooling layers in CNNs. The number of convolution and pooling layers is a configurable parameter and is set by the designer of the network. In current example we use two convolution and one more convolution layer.

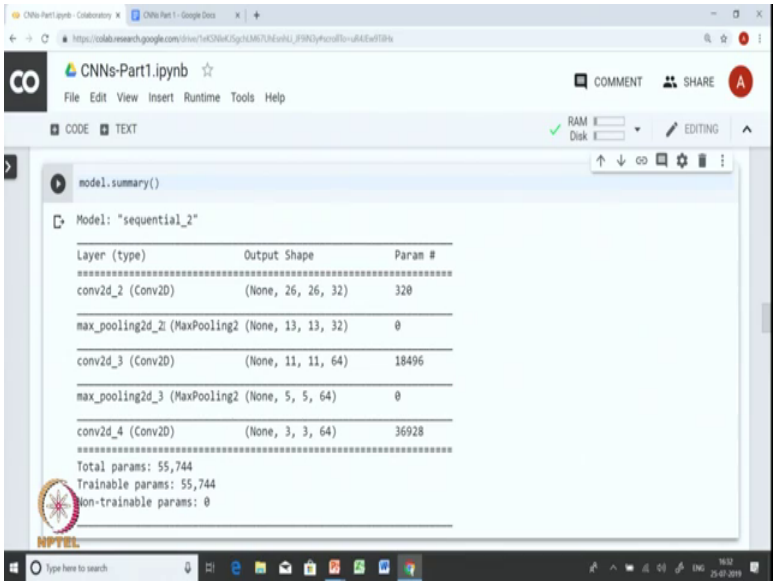
(Refer Slide Time: 22:06)



In the current example we use two convolution pooling layers and one additional convolution layer at the end. We use 32 filters in the first convolution layer and 64 filters each in the second and the third layer. Each filter is 3 x 3 in size and we use a stride of 1. We have not used any padding in any of the convolution layers.

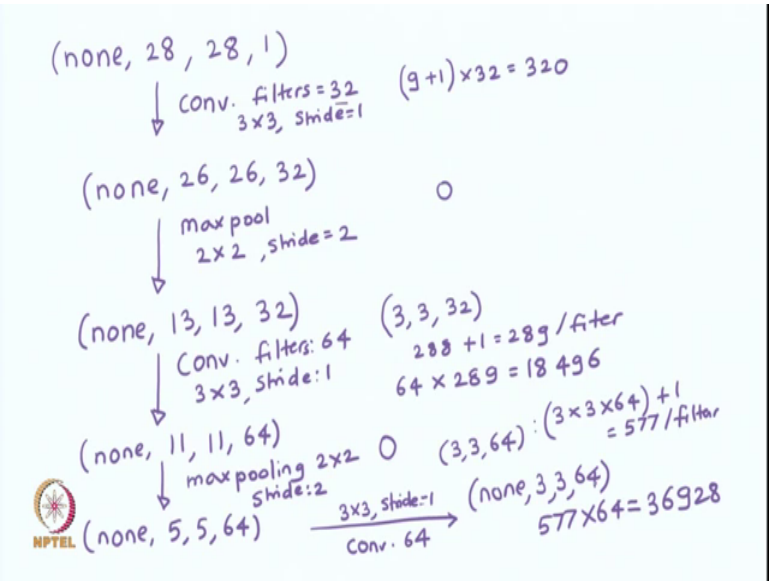
We used max pooling for down sampling with a window of 2 x 2 with a stride of 2. It is not necessary that every convolution layer should be followed by a pooling layer. Sometimes we can have pooling after a few convolution layers as well. Now that the model is built let us look at the model summary.

(Refer Slide Time: 23:02)



Let us work out a number of parameters for each convolution layer. We will write the shape of input and output tensors for each layer and then work out the number of parameters from that.

(Refer Slide Time: 23:25)



The input to CNN is a 4D tensor. In the 4D tensor the first axis corresponds to samples the second is number of rows or the height, then width and the number of channels. In the case of a MNIST we have a single channel because we have a grayscale image. So, this is the 4D tensor corresponding to the input image. So, we apply convolution operation on this where we use 32 filters. Each filter is of size 3 x 3 with stride of 1. Because you are applying 3 x 3 filter this input gets transformed into another 4D tensor having height and width of 26 and having 32 channels. Each channel corresponds to a filter that we used during the convolution operation. We apply max pooling on this output with max pooling window of 2 x 2 and stride of 2.

So, this gives us another 4D tensor with height of 13 width of 13 and having 32 channels. You can see that there is a downsampling that happened from the output of the convolution as we applied the max-pooling on it. Then you apply another convolution with 64 filters each filter is 3 x 3 filter with stride of 1.

We get a 4D tensor with shape none, with height and width of 11 and number of channels equal to the number of filters that we use in the convolution which is 64. Then we apply max pooling on this with 2 x 2 window size and stride of 2. So, we get a 4D tensor with height and width of 5 and number of channels as 64.

We apply another convolution with 64 filters each with size 3 x 3 and stride of 1. We get a 4D tensor with height and width of 3 and number of channels 64. So, this is output that we get after applying convolution pooling 2 times followed by a convolution operation. Let us work out the parameters for each of the layer.

So, you can see that in the first convolution we use 32 filters each with size 3 x 3. So, because we use 3 x 3 filters we have 9 parameters corresponding to the positions of the filter and 1 bias. So, there are 10 parameters per filter and there are 32 filters. So, there are in all 320 parameters. We learn weights of each of these parameters during training of the network. The max pooling layer does not have any parameters.

Let us come to the second convolution layer. The size of the filter here is 3 x 3 x 32. So, we are 3 x 3 filters with depth of 32 which is equal to the number of channels in the input that generates about 288 parameters + 1 bias. So, there are 289 parameters per filter. So, for 64 filters we get 18496 parameters. The max-pooling layer again does not have any parameters.

So, for the final convolution layer here we have the shape of the filter is 3 comma 3 comma 64, we have we have 3 x 3 filter with 64 channels.

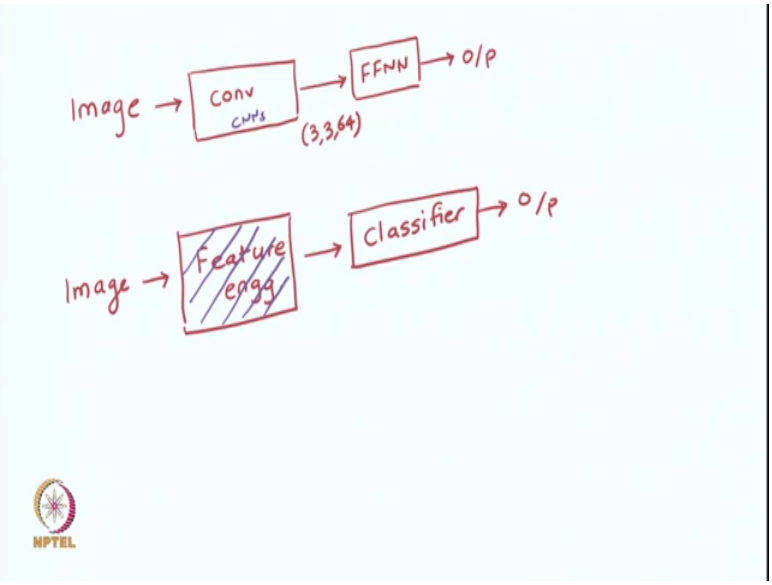
So, the number of parameters here will be 3 x 3 x 64 + 1 bias which makes it to 577 per filter and we have 64 such filter. So, we have in all 577 x 64 which comes down to 36928 parameters.

The number of parameters in the convolution layer depends only on the filter size. It does not depend on the height and width of the input. It can be observed that the width and height dimension tend to shrink as we go deeper in the network. We started with height and width of 28 each and after a couple of convolutional pooling followed by a single convolution operation we got height and width of 3.

The number of output channel for each convolution 2D layer is controlled by the first argument. The number of filters used in convolution a normally 32 or 64. Typically, as the width and height shrink we can afford to add more output channels in each convolution 2D layer. Now, end of this particular operation we got an output of 4D

tensor which has height and width of 3 and number of channels 64. We feed this output into a dense layer to perform the classification.

(Refer Slide Time: 31:14)



So, what is really happening here is, we take an image, we apply a bunch of convolution pooling operations that gave us representation that will feed into feed forward neural network which will give us the label corresponding to the digit written in the image.

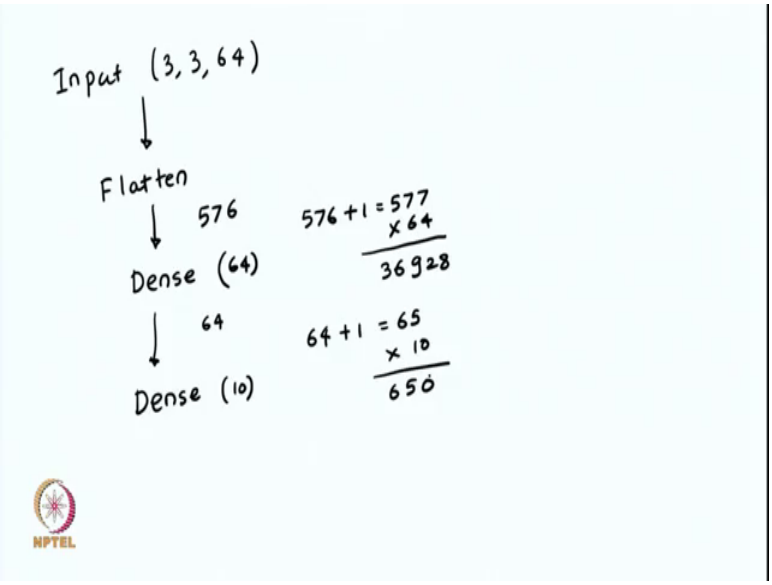
In traditional machine learning flow given an image we use to first perform feature engineering using computer vision libraries. A feature is used to get fed into any machine learning classifier which after training will give us output.

Now, you can see that the feature engineering part in the traditional machine learning is getting replaced by CNNs. So, we can think of CNNs as a way of generating features automatically for a given image. The beauty of this approach is that the right representation is learned during the model training freeing us from expensive and tedious feature engineering task. We will use feed forward neural network for classification task. Let us see how to set this up. So, we will add a Flatten layer that will take the output of the convolution layer and flatten it into a 1D tensor.

You feeding the output of the flattened into a dense layer containing 64 units, we will use Relu as an activation. MNIST as 10 output classes. So, we use a final dense layer with 10

outputs and a Softmax activation. Let us understand it with an illustration and work out the parameters and work out the parameter calculations ourselves and we will compare that with the parameters shown in the model.summary.

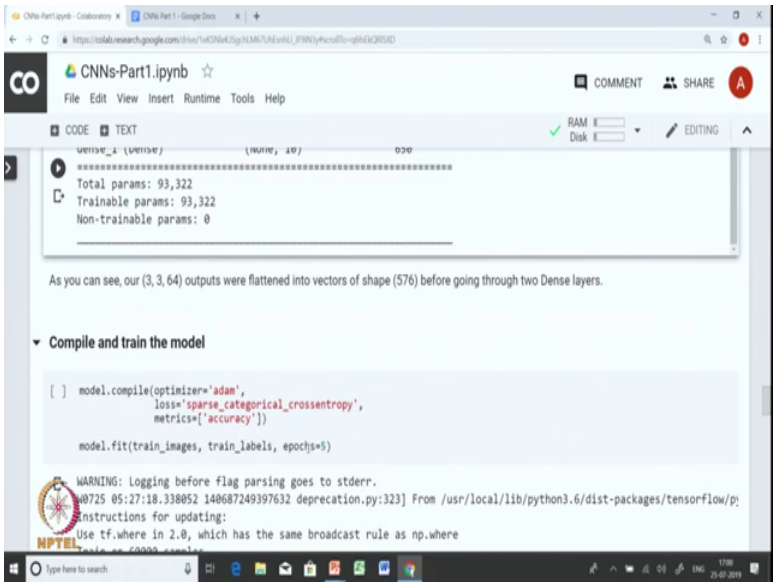
(Refer Slide Time: 34:14)



So, here the input is (3, 3, 64). We pass it to the Flatten layer which gives us 576 numbers which are fed into a dense layer whose output is fed into another dense layer; flatten has no parameters it outputs 576 numbers which are input to each of the 64 units in the dense layer over here. So, each of the unit in the dense layer has 576 parameters + 1 bias which makes it to 577 parameters per unit and we have 64 such units making it, 36928 parameters. So, this produces 64 values one corresponding to each unit.

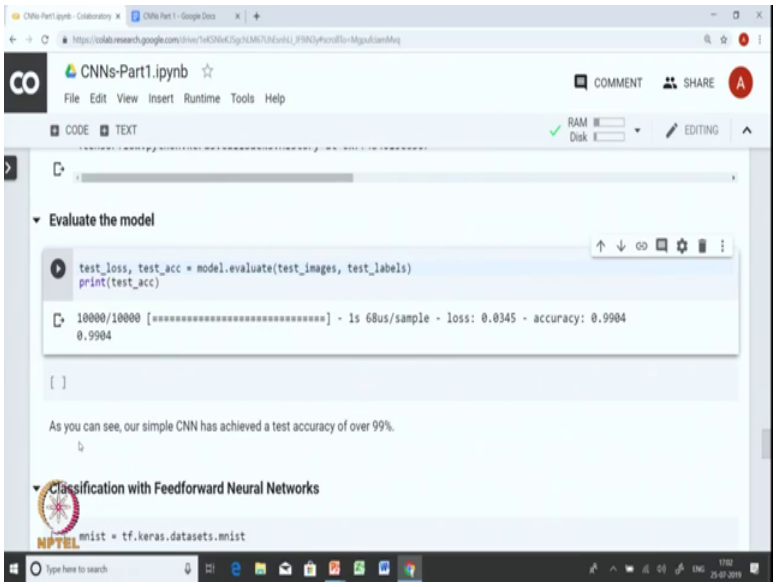
So, the final dense layer has 10 units; each unit receives 64 values from the previous layer adding 1 bias parameter to it makes it 65 values per unit. So, in total we have 650 parameters for the final layer. So, if we sum across the CNNs and fully connected top layer we are total of 93,322 parameters. So, let us compare our calculation with model.summary().

(Refer Slide Time: 36:53)



So, we can see that we arrived at 93322 model parameters which is matching with the output of model.summary(). Let us set up the training of the model. We use *sparse_categoricalcrossentropy* loss with *Adam* optimizer. Let us train the model for 5 epochs with training images and training labels.

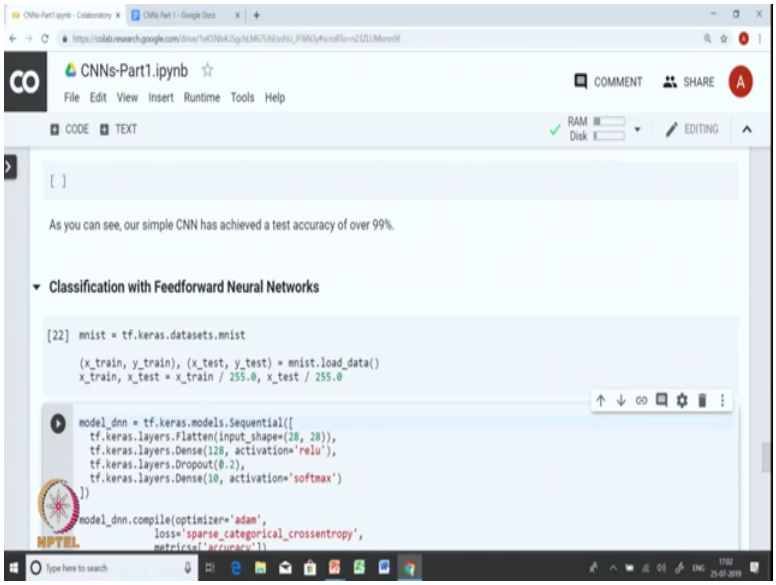
(Refer Slide Time: 37:28)



After 5 epochs the model has achieved more than 99% accuracy in recognizing handwritten digits on the trading data. Let us check the performance of the model on the

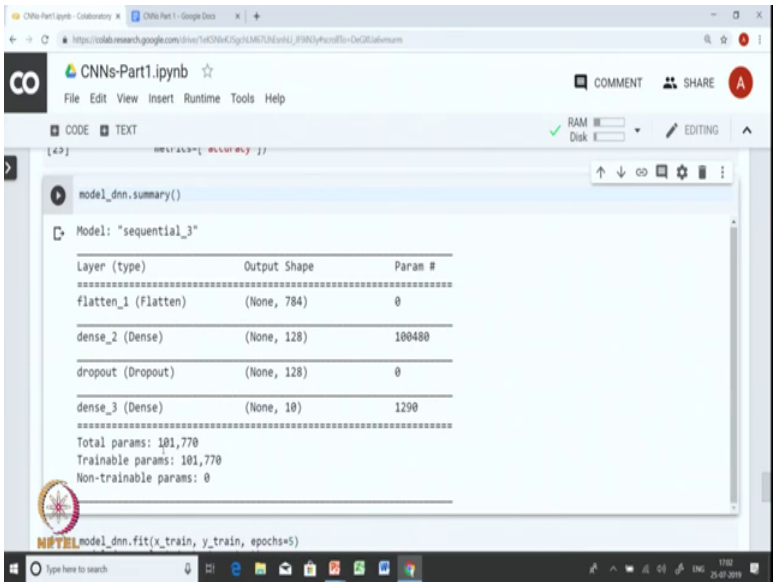
test data. We used evaluate method of model class for calculating accuracy of the model on the test data. We passed test images and their labels as input, so that the statistics can be calculated. Here we observed more than 99% accuracy on the test set. It is interesting to compare this architecture with fully connected feed forward neural network that we used in earlier exercise.

(Refer Slide Time: 38:20)



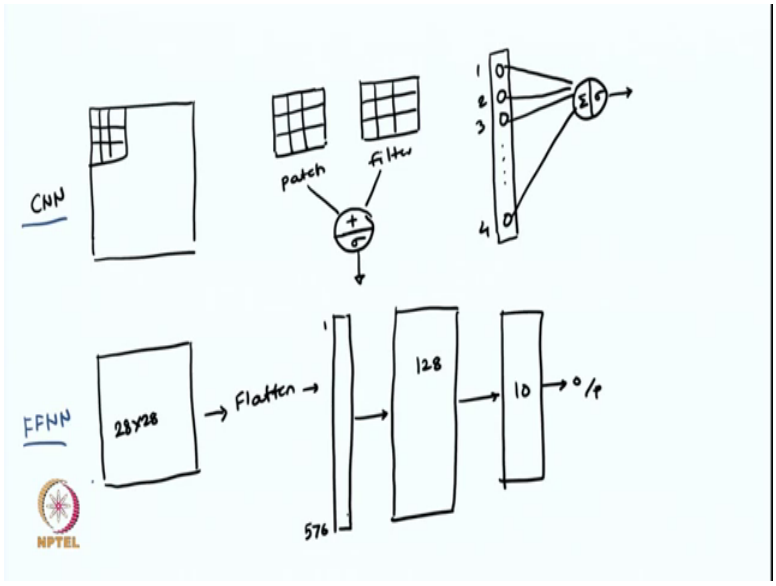
Let us set up the feed forward neural network and compare it against the CNNs. So, we input the MNIST data set and we set up a feed forward neural network with a hidden layer with 128 units.

(Refer Slide Time: 38:40)



You can see that this feed forward neural network has more than 100k parameters as compared against 93k parameters that our CNN has.

(Refer Slide Time: 38:54)



So, we can see that in case of CNN we are defining patches and we are taking a patch and performing a convolution operation with the filter. We perform a linear combination of each position the image with each parameter in the filter. We perform linear combination followed by non-linear activation; while in case of feed forward neural

network we take the entire image, you flatten it so that we get a single array. In this case since you have 20 x 20 image we get an array of 576 numbers which we are passing to a hidden layer with 128 units followed by a dense layer of 10 units to get the output.

If we come up with the equivalent flattened representation, we have these 9 values and we have a node. These 9 values are connected to this particular node which is a neuron or a unit in neural network which performs linear combination followed by activation. So, we are basically capturing local patterns in case of CNN. So, this is CNN and this is feed forward neural network. So, CNN procedures by capturing local patterns; whereas, in fully forward neural network, we learned global pattern involving all the pixels.

Today, we learned about CNNs which are extensively used in computer vision applications. We discussed key operations in CNNs and demonstrated their usage in recognizing handwritten digit task. We compared CNNs with feed forward neural network. We learned about CNNs ability to automatically generate features from the given image.