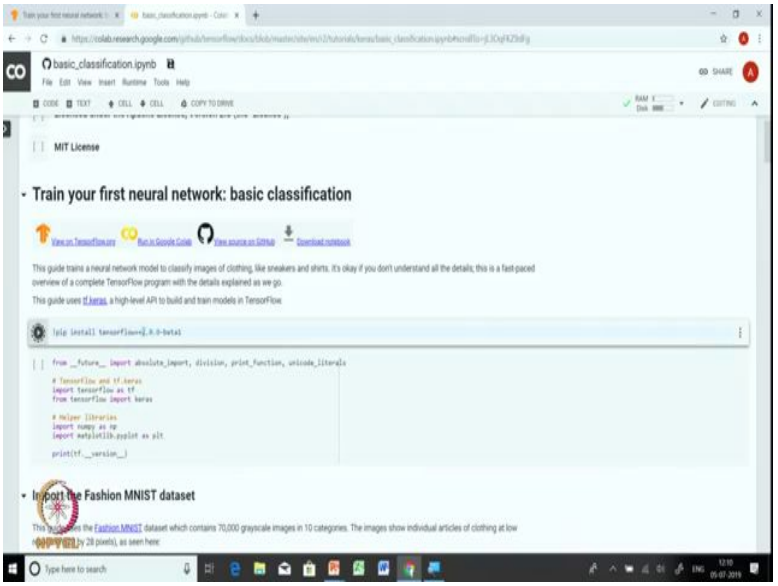
Practical Machine Learning with TensorFlow
Dr. Ashish Tendulkar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 14
Classify Images

Welcome to the next module of the course. In this module, we will build our first neural network model for an image classification task. In this exercise, we will be using tf.Keras API.

(Refer Slide Time: 00:32)

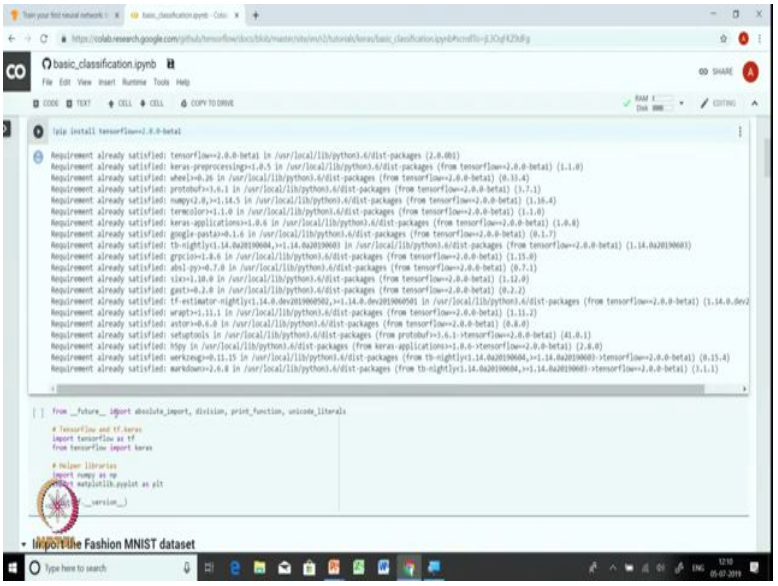


We are going to build a model to classify images, and for this exercise we use the fashion MNIST dataset. Fashion MNIST data set is very very similar to MNIST data set that we used earlier for building a hello world model for TensorFlow. The fashion MNIST data set has 10 classes of different fashion accessories.

There are 60000 images in the training set of fashion MNIST and 10000 images in the test set. Each image is 28 by 28 pixel in size and is associated with exactly one label. So, I would like to tell you that when you are going through the particular colab for this lecture, I would urge you to stop and try coding the things that you see by yourselves. That will help you to understand the TensorFlow Keras API better.

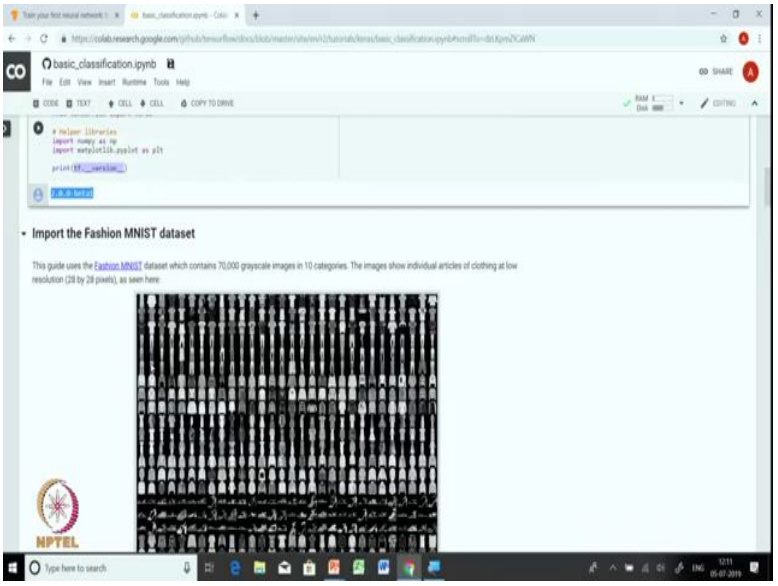
So, let us begin to begin by connecting to colab runtime and installing TensorFlow 2.0.

(Refer Slide Time: 02:26)



If the API is not available on the runtime, TensorFlow will be downloaded and then installed in the cloud run time. Once we download this TensorFlow 2.0, the next task is to import the libraries that are required for us in building the model.

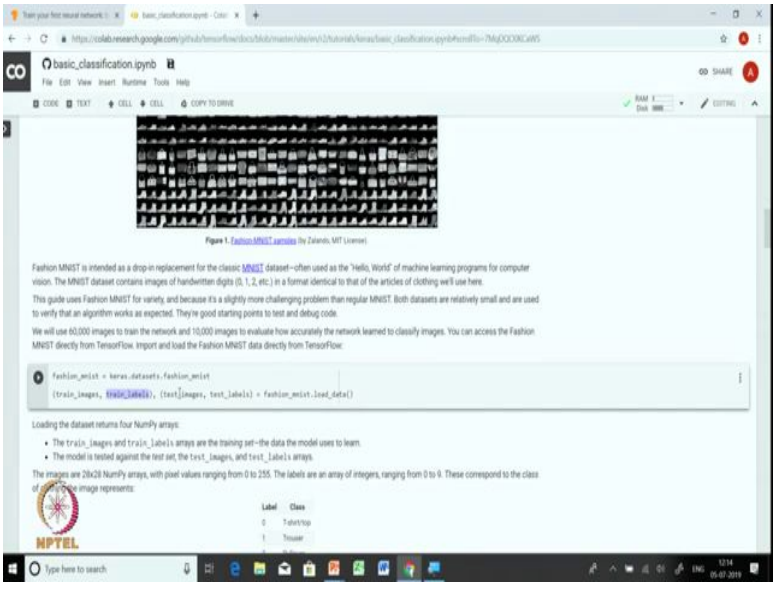
(Refer Slide Time: 02:52)



We will be using Keras API. So, we will we are going to import the Keras library and we are going to import NumPy for manipulating and storing the data. We will also import matplotlib.pyplot for plotting various images of objects in fashion MNIST dataset. Finally, we will make sure that we have the right TensorFlow version loaded on the colab run time. We ensure that by printing tf.__version__.

Data is the first prerequisite for machine learning model and in this exercise fashion MNIST is the data set that we are going to use. We are going to use our training data in fashion MNIST for training our model. The training data in this particular dataset has pairs of images and their associated label, and the image is presented to us in the form of a 28x28 pixels, and there are 10 possible labels. The labels are given IDs ranging from 0 to 9. All the images in MNIST dataset are all gray scale images.

(Refer Slide Time: 04:55)



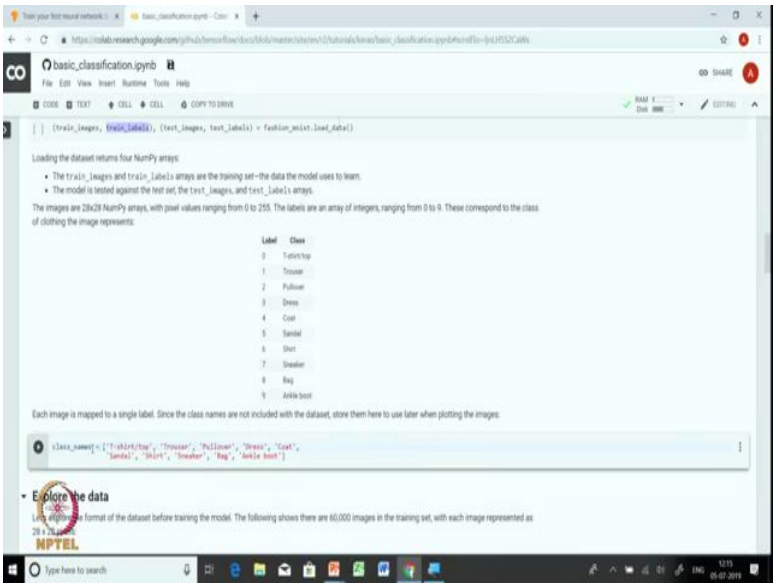
So, you can see some of the images are printed on the screenshot above. And these are images of objects from different classes.

Fortunately, the fashion MNIST data set is already present in TensorFlow, and since this data set is already present in TensorFlow, we can directly import and load the data from TensorFlow. If your dataset is not present in the TensorFlow we will have to write; we will have to write or we have to make provisions for making sure that our data set is available in

TensorFlow and we have covered this in one of our previous modules. So, you can go back and refer to that if you want to bring in your own data in TensorFlow.

But for the for the purpose of this exercise, the dataset that we are using here which is fashion MNIST data set is already present, so we will use fashion_mnist.load_data command() to load the data in colab, and this command returns us 4 NumPy arrays. So, two arrays corresponds to training data and the other two correspond to the test data. Within training data we have one array of train images and one array for training labels. Similarly, in test we have one array for test images and one array for, one array for test labels.

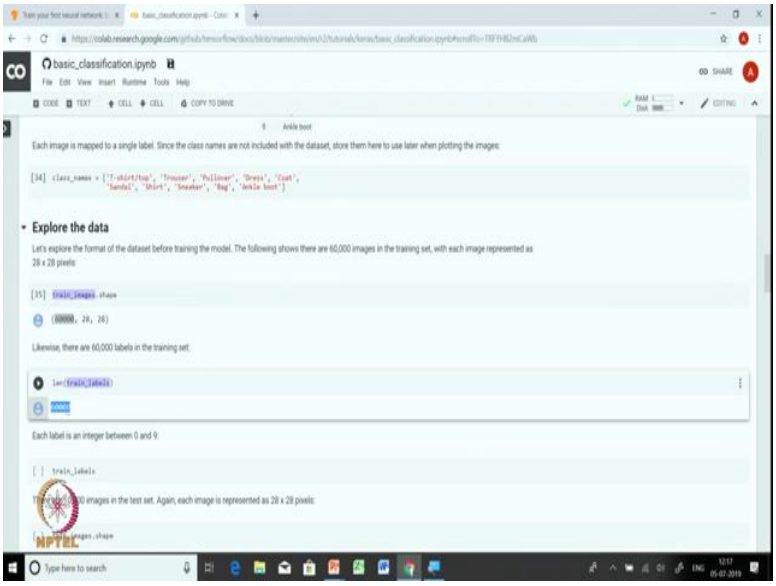
(Refer Slide Time: 06:51)



And as we you know said earlier each image is 28 by 28 NumPy array. The pixel value in each of the cells of this array ranges from 0 to 255 and the labels are an array of integers from 0 to 9. You can you can see all the labels are in the corresponding class name displayed above.

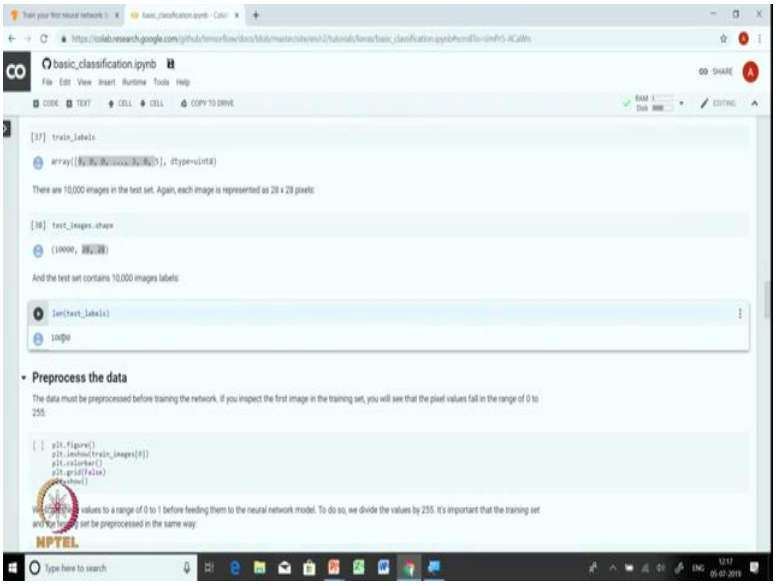
Since, this class names are not included in the dataset we will actually store them in an array, so that we can use it later to print the name of the class along with each image. This will be used later when we are exploring the data set or whenever we are trying to print the actual label and the predicted label.

(Refer Slide Time: 08:15)



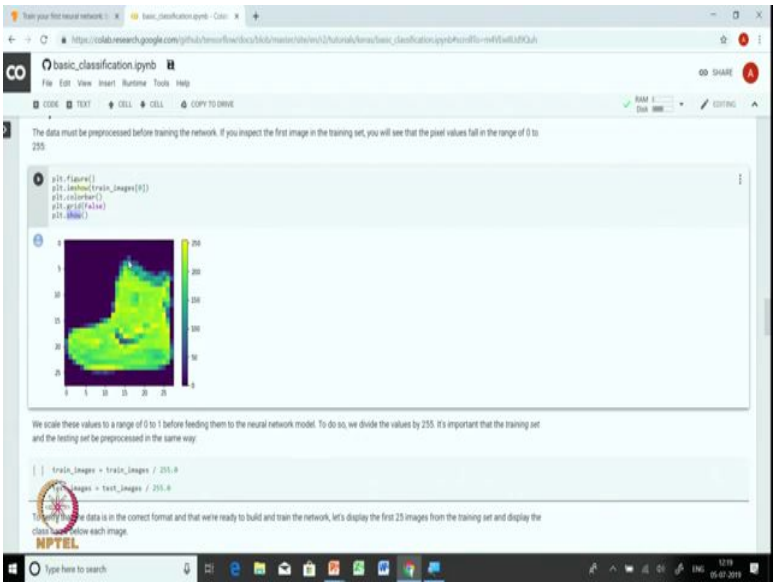
Before looking into the images, you can see that train images is of the shape (60000, 28, 28). This 28 by 28 here is the dimensions of each image and we have 60000 such images in train_images NumPy array.

(Refer Slide Time: 09:04)



The test set shape is (10000, 28, 28). There are 10000 test images each of size 28 by 28. So, you can you can observe that the test image has exactly the same pixel size as the training image.

(Refer Slide Time: 10:13)



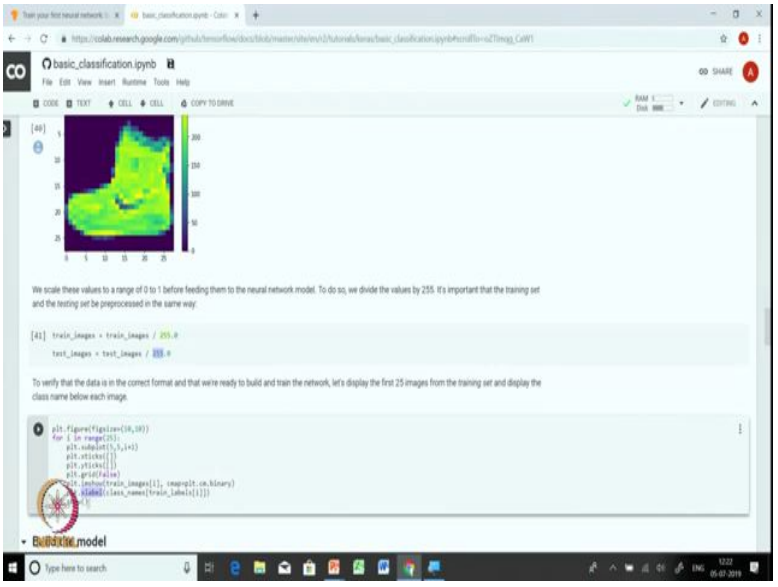
After exploring the data, it is important to preprocess the data. In the data pre processing step we are generally transform the data, we normalize the data. So, let us see how we do that in case of images.

But before normalizing the data, let us plot an image and see how it looks like. So, we will be using imshow method of matplotlib.pyplot to plot the first image of the training set. You should see an image of an ankle boot on the screen. So, you can see that there are exactly 28 rows and 28 columns, each cell in this array corresponds to 1 pixel value and pixel values range from 0 to 255. So, this is the color coding of the pixel value. So, this is a visual representation of the first image.

As part of image normalization, we will make sure that each pixel value ranges from 0 to 1 and we do that by dividing each pixel value by the range of pixel value which goes from 0 to 255. So, essentially we can simply divide each pixel value by 255 and this will make sure that each pixel value is between 0 to 1.

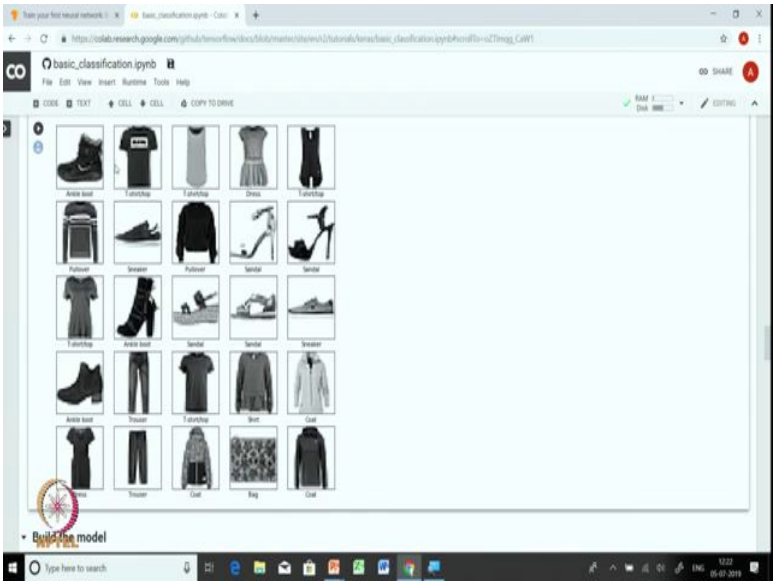
We do the same thing on the test data, and in order to make sure that training data and test data is pre processed or specifically normalized in this case in exactly the same way. It is very important to ensure this particular step, that we use exactly the same normalization steps across training and test data.

(Refer Slide Time: 13:05)



Let us confirm that the data is in the correct format and we are and we have normalized it correctly. So, we will again use imshow function to print each of the image. So, here what we do is we are going to plot first 25 images. And along with the image we are going to print the class name of the image that we are stored in the class name array, and class name will be stored as an as a label on the x axis and finally, we print all these images using matplotlib.pyplot.show method.

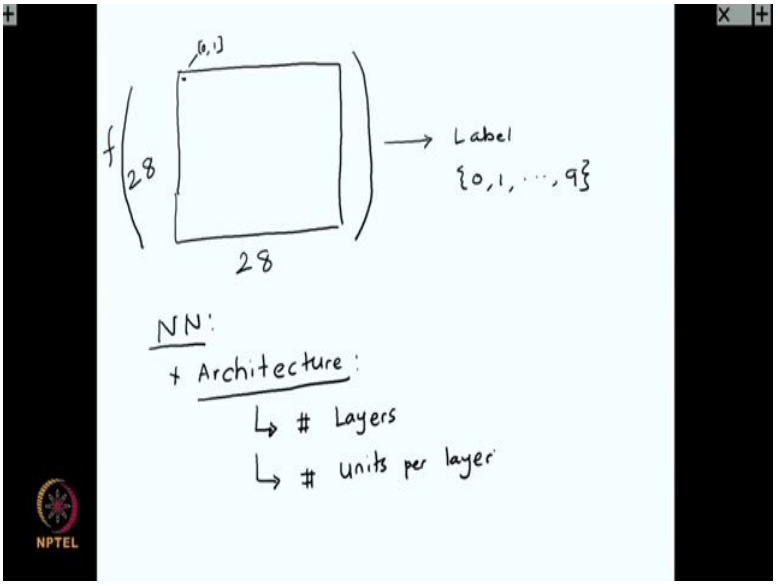
(Refer Slide Time: 14:11)



So, now you can see that each image is printed here along with the names below the object. So, we have plotted 25 images in 5 by 5 grid with each row has 5 images and there are 5 such rows.

Now, that we have explored, normalized and visualized the data, the next task is the core task of model building. In this case we are going to use a neural network model for classifying the images. Let us look at the architecture of the neural network model before getting into the code.

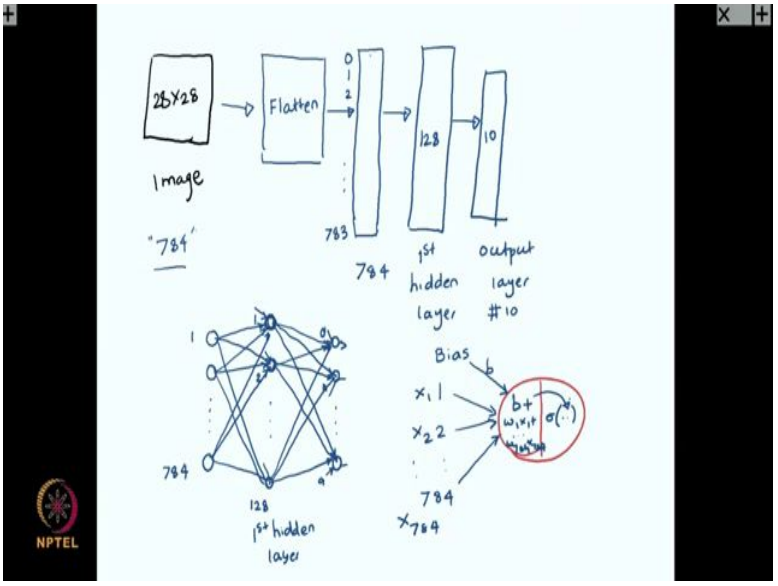
(Refer Slide Time: 14:59)



There are 28 rows and 28 columns in each image and each cell as a value between 0 to 1. We are going to use this pixel information to learn the label. So, essentially we are trying to design a function that takes in this particular image and it maps this particular image to possible labels between 0 to 9.

So, the first component is the architecture of the neural network model. In architecture we specify number of layers and number of units per layer. In this particular exercise, we are going to build a very simple neural network model. We have 28x28 image as an input. We first introduce a layer called flatten layer.

(Refer Slide Time: 17:15)



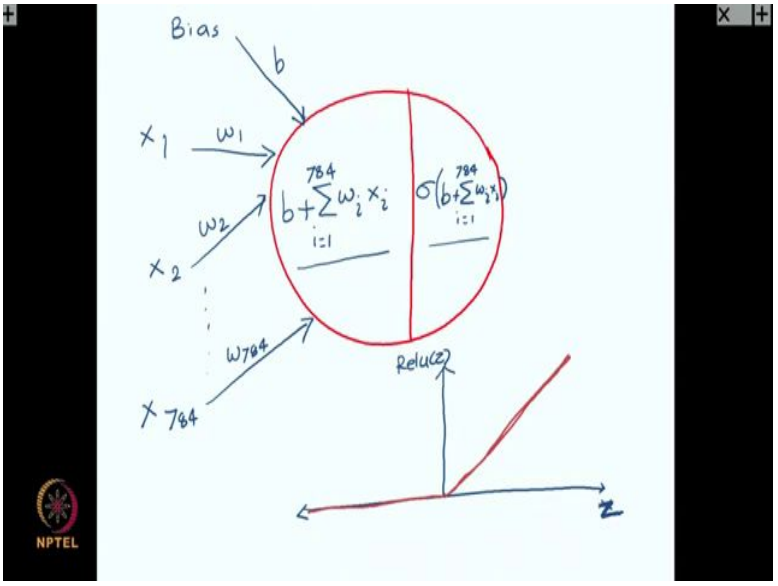
What is flatten layer does is it takes us to 28 by 28 pixel values and convert that into a list of 784 numbers because its 28 by 28 there are totally 784 values or 784 cells. So, we essentially open the cells up and append them one after the other. So, we have this pixel 784 values indexed from 0, 1, 2, up to 783. The image which was in the matrix form is opened up in the form of a list or in the form of an array.

Then we will use one hidden layer having 128 units. So, this is the first hidden layer and then we have an output layer with 10 units, each unit corresponding to 1 class. So, this is the model. The block representation of a neural network model is shown above. So, we have pixel values 1 to 784 as input, then we have 128 units is the first hidden layer and then we have 10 units in the output layer.

So, we are going to use dense layers that make sure that each node in the previous layer is connected to the node in the next layer. We will again use a dense layer for the output layer. Let us see what each individual node in the hidden layer is doing.

It has all 784 inputs and one additional input called bias. This particular node does two things. First, it does a linear combination. We multiply each input x_i with a corresponding weight w_i and sum across all the inputs . Secondly, we apply a non-linear activation to the result of the first step. Let me present this, let us expand this in the next slide.

(Refer Slide Time: 23:03)

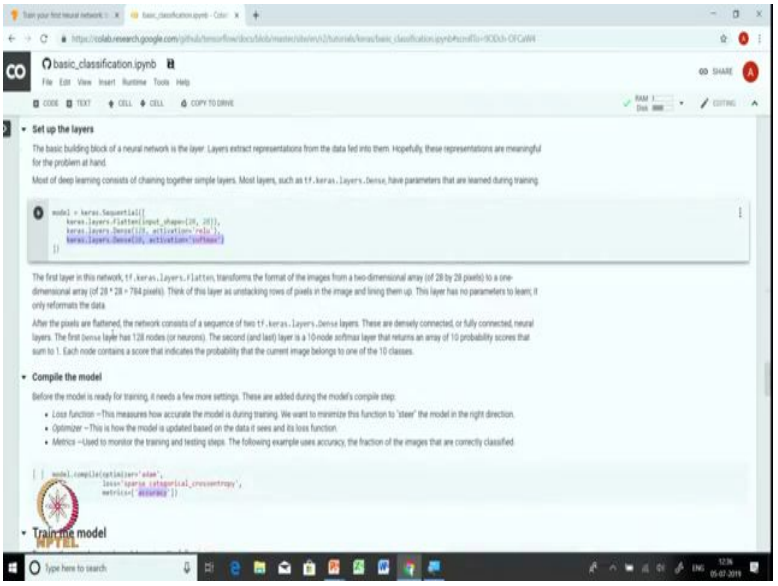


First step: $b + \sum_{i=1}^{784} w_i x_i$

Second step: $\sigma(b + \sum_{i=1}^{784} w_i x_i)$

In this case, we are going to use ReLU which is a popular choice for activation function. ReLU returns the value if the value is positive, and returns 0 if the value is negative. The output of the ReLU is shown above. So, the Relu helps us bring in the nonlinearity in the equation. So, each of the, each of the hidden unit has this particular computation going within them.

(Refer Slide Time: 25:38)



Coming back to colab, we first use a keras.layers.Flatten layer which helps us to convert the matrix representation of the input to an array representation. So, this layer we will convert this 28 by 28 representation to an array of size 784.

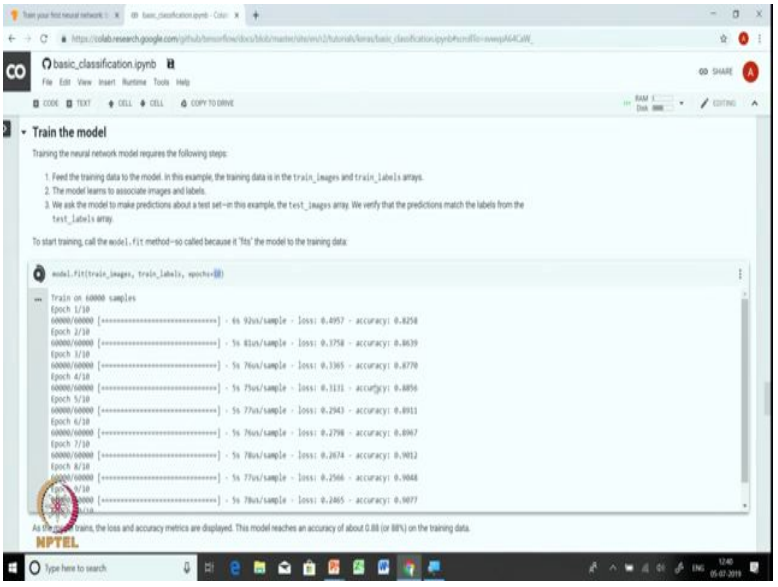
Then we have one hidden layer which we call as dense. I just explained how the dense layer works and dense layer has 128 units and we are going to use us ReLU as an activation function. Finally, we have a dense layer of 10 units and this uses softmax as an activation function.

The softmax layer returns an array of 10 probability scores that sum up to 1, with each score indicating the probability that the current image belong to 1 of the 10 classes. Having set up the model, we will compile the model. We are required to specify what kind of optimizer we will use for training the model, the kind of loss functions we will be using, and also the metric to track during the training process.

In this case, we use Adam which is proven to be one of the better optimizers for training deep neural network models. We are using sparse categorical cross entropy loss because we have 10 different classes, in the output and we are using accuracy as a metric as a measure that we will be monitoring or a metric that we will be monitoring. Let us execute this particular step, now set up the model and now we will compile the model.

After compiling the model the next step is to train the model, and we said that for training the model we need to specify the training data and for how many iterations we want to train the model. It is advisable to train the neural network model in a batch setting. So, we sometimes also specify the batch size and the regularization parameter for training.

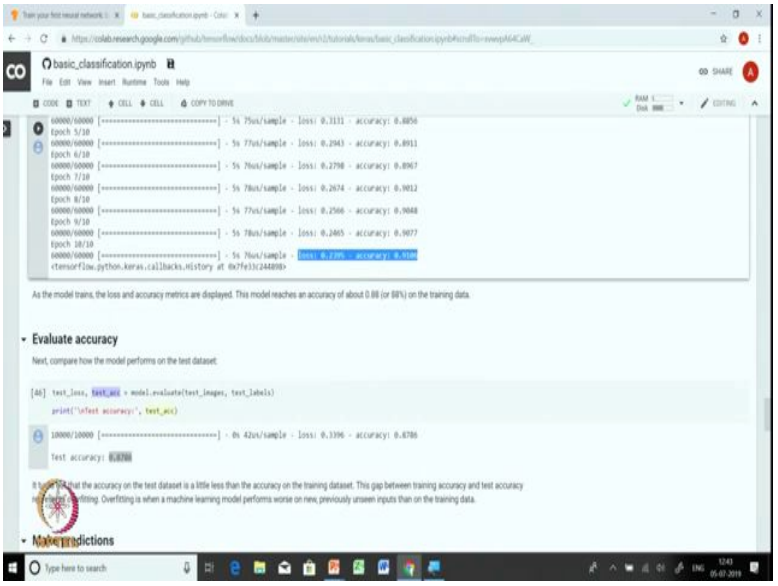
(Refer Slide Time: 29:18)



We will train this model for 10 epochs. You have to be careful with number of epochs, if you if you know train for longer epoch there is a chance that the model will over fit. So, you have to watch out for over fitting. And we are not specifying the batch size, so default batch size of 32 will be used for this particular fit function.

So, next let us train the model and see where we reach. So, you can see that there is a progress bar that shows us progress. You can see the time taken per sample which is 92 microsecond in this case, and you can also see the loss and the accuracy numbers. You can observe that the loss is going down as we train the model further and further and accuracy goes up. So, we started with loss of somewhere like 0.49 and accuracy of 0.82 and after 10th iteration let us see where we reach.

(Refer Slide Time: 30:57)



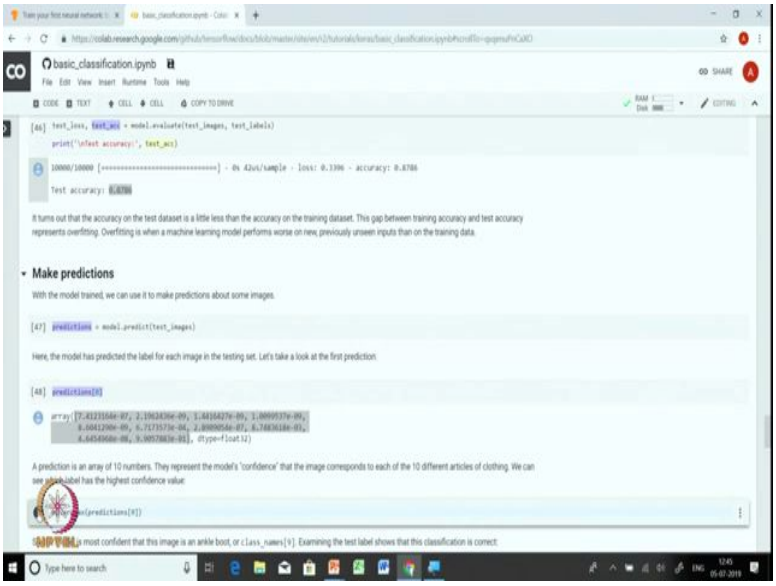
So, model has completed 10 epochs and the loss has come down to 0.23 and accuracy has gone up to 91 percent. So, we started with 82 percent accuracy with the initial parameters set and after 10 epochs we got accuracy of 91 percent.

Let us see how this model performs on the unseen data set. And as we have been talking in this class or you must be aware having basic background in machine learning that we want machine learning algorithms to work well on the future data. And how do we really test its performance in the future data? We use some data as a surrogate for the future data.

We will evaluate the accuracy of the model on the test data. We use model.evaluate function which will take test images as an input and we will also supply the actual labels as an output. So, actual labels help us to compare the actual label with the predicted label and that helps us to get the test accuracy.

The evaluate function returns the test loss and the test accuracy. We got a test accuracy of around very close to 88 percent which is slightly lower than the training accuracy the numbers that you see here after 10th epoch is the accuracy on the training set and since we train the model on the training set training set is training set always receives higher accuracy than the test set.

(Refer Slide Time: 33:14)

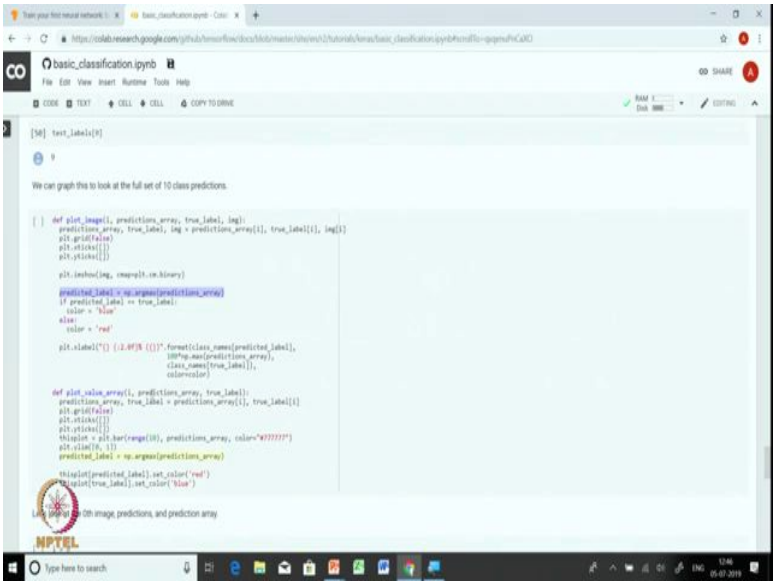


Let us use this model to make predictions because that is that is going to be our objective, we have training the model. We will be using this model to predict the label of a new fashion item.

We use model.predict as a function which takes the which takes a bunch of images as input and returns prediction for each of the image. So, let us run this on all the test images and look at the prediction for the first image.

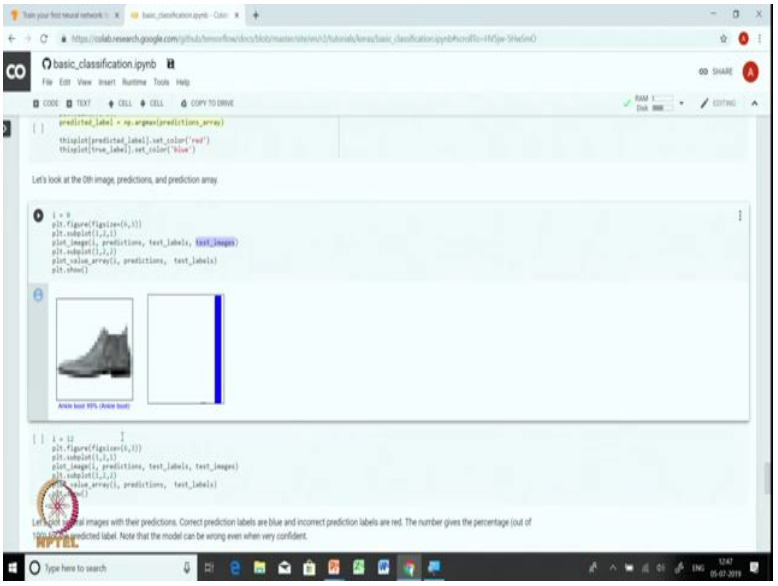
If you take the first value, it represents the probability of this image having the 0th. So, it appears that you can see that this particular image has the highest probability mass or highest probability at label 9. So, we essentially will what we will do is we will use np.argmax function to assign the label correspond corresponding to the position having the highest probability mass.

(Refer Slide Time: 34:59)



Let us plot the images and the probabilities in the form of a graph. So, we are going to print actual value as well as predicted value, and if the actual value is matching with the predicted value we will use blue color and if there is a mismatch we will use red we will be using the red color. So, let us look at how does the first image looks like.

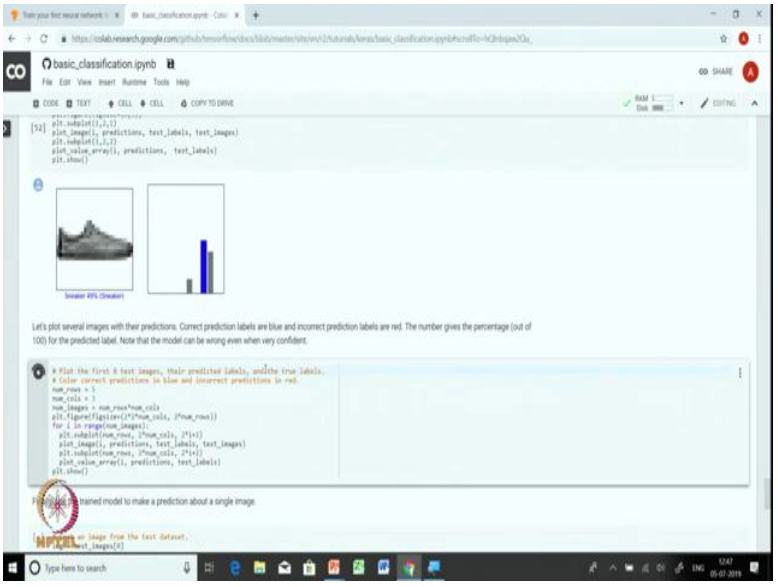
(Refer Slide Time: 36:12)



So, what we do is for the first image we supply the predictions, the predictions array and the test label and the test image. So, we essentially give all this information and we try to print

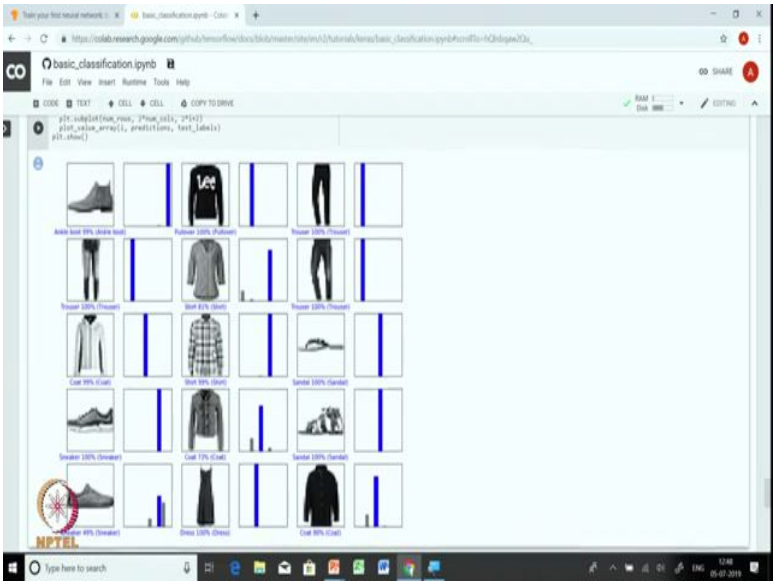
this. So, this is the actual image along with its label, this is the actual label which is there in the bracket which is ankle boot and this is the predicted label which is again ankle boot with 99 percent confidence. You can see that there is a very tall graph at the ankle boot at a position corresponding to ankle boot. Let us look at the label for the 12th image.

(Refer Slide Time: 37:05)



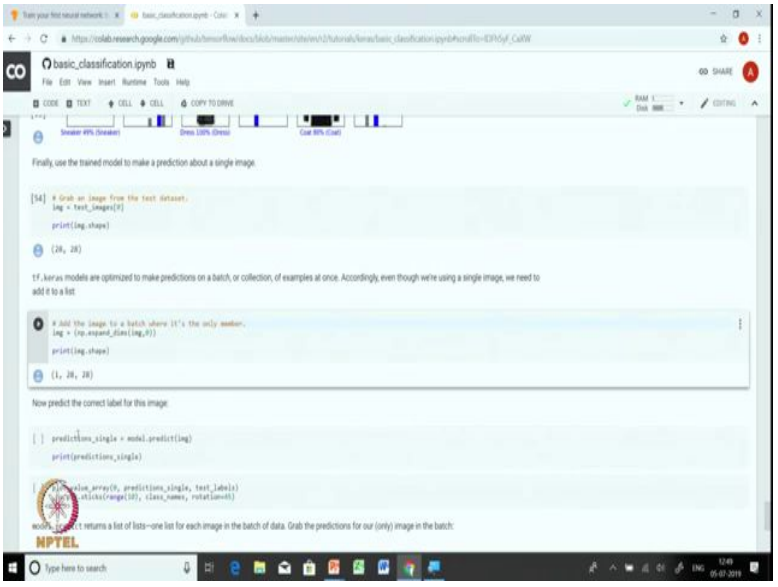
So, 12th image is a sneaker which is correctly predicted sneaker, but here the probability is much smaller compared to the earlier example.So, let us plot several images along with their prediction just to see you know how we do on some more images.

(Refer Slide Time: 37:46)



So, we are going to print 5 rows and 3 columns, each column, each row has essentially 3 images and we have he here on the prediction for first 15 objects. And you can see that in this case all the objects have been correctly classified. Some objects have been classified to the 100 percent confidence level and so this particular snicker example that we saw has the least confidence amongst all these labels that we, all the objects that are on the screen.

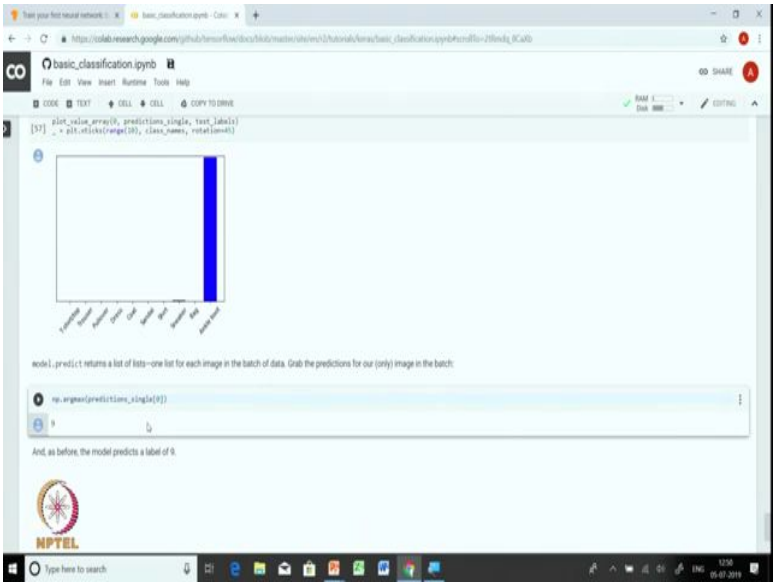
(Refer Slide Time: 38:27)



Finally, let us understand how to make a prediction for a single image. So, this is a single image which has a shape of 28 by 28, and Keras prediction function is optimized to make prediction on a batch or a collection. Hence, we also insert the single image into a collection and then send it for prediction. So, let us, add image to a batch where it is the only member and you can see that the size of the batch is 1 comma 28 comma 28 and we will pass this particular image tensor to the model.predict function and it will give us the prediction vector.

And prediction vector has 10 values as we saw earlier, and we will plot the array of predicted value and on the x axis we have you know the labels corresponding to the names of the label, and you can see that you know there is a very high probability mass on the ankle boot which is also displayed in blue. So, this is the correct prediction.

(Refer Slide Time: 39:26)



And let us see np.argmax and you can see the label of 9 just as before. So, in this particular exercise we build an image model with a feed forward neural network. So, this was our first model that we built with TensorFlow Keras API. In the next exercise, we will use the TensorFlow Keras API to build models for structured data as well as for regression problems.

Thank you.