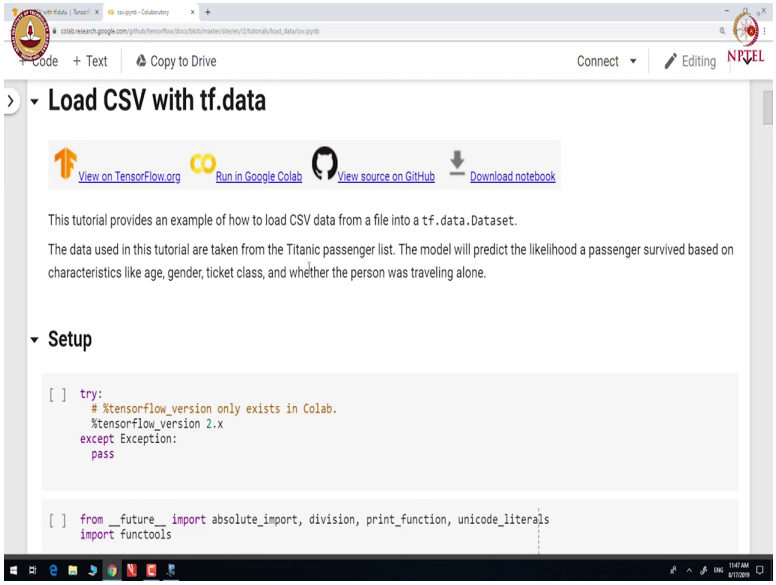
**Practical Machine Learning**
**Dr. Ashish Tendulkar**
**Department of Computer Science Engineering**
**Indian Institute of Technology, Bombay**

**Lecture – 13**
**Building Data Pipelines for Tensor flow-Part 2**

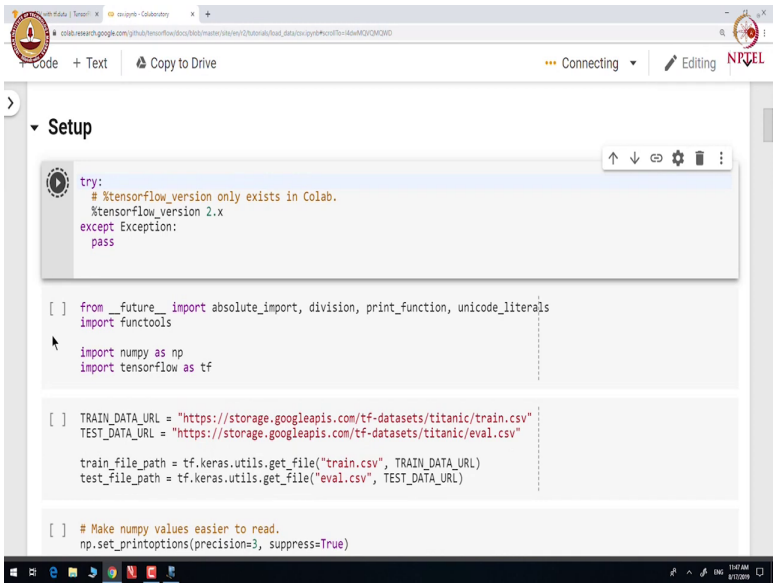
[FL] In this session, we will study how to construct dataset from different formats. We will cover formats like CSV or in memory structures from NumPy and Pandas we will also study how to construct dataset objects for images and text.

(Refer Slide Time: 00:35)



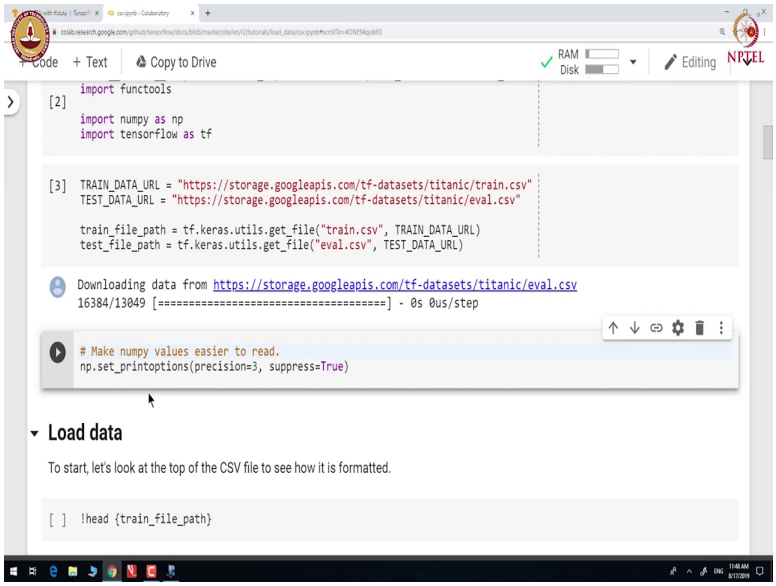

So, let us start with CSV data. So, the data that we use in this session is taken from Titanic passenger list where you want to build a model to predict the likelihood of a passenger survival based on their characteristics like age, gender, the ticket class, whether the passenger was travelling alone etc.

(Refer Slide Time: 01:17)



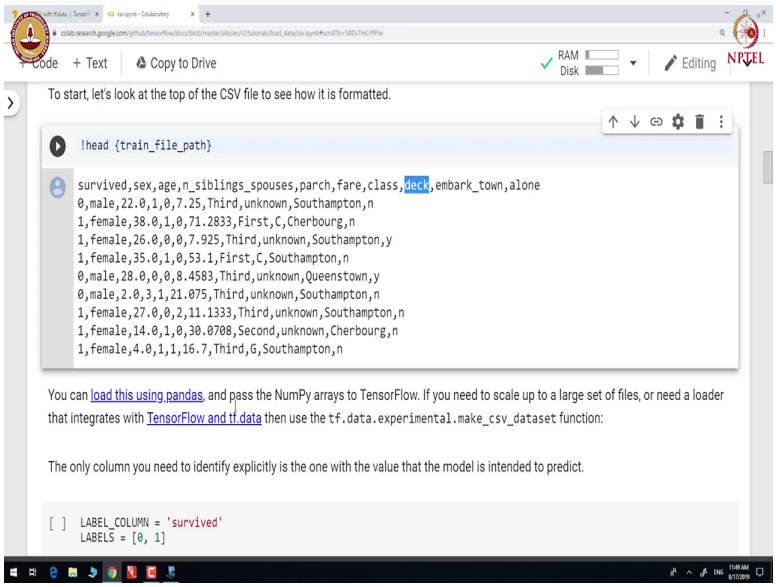

Install Tensor Flow 2.0 and import essential libraries.

(Refer Slide Time: 01:43)




Then, download the train and test data from titanic dataset. The titanic dataset is available in *tf.dataset*. We set the precision to 3.

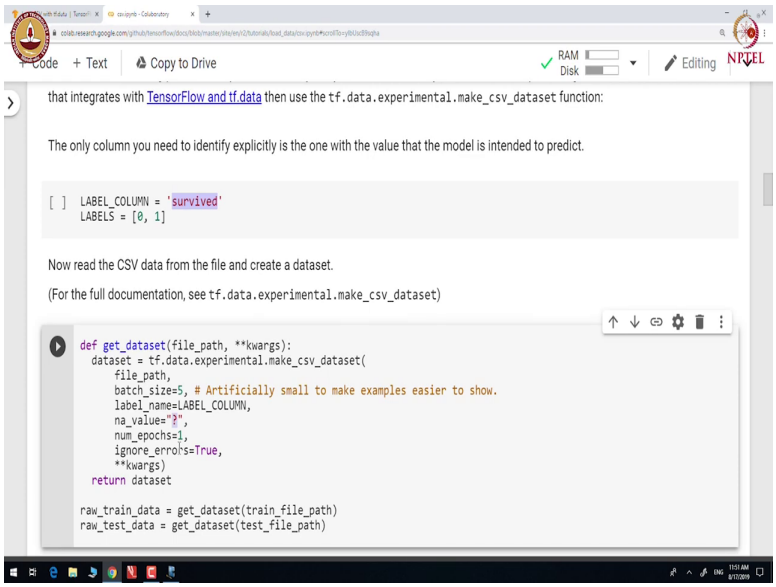
(Refer Slide Time: 01:59)




Let us look at first few examples from CSV file. So, you can see that the label survived is the first column it is either 0 or 1. Then, we have features or characteristics like gender or sex, age, number of siblings or spouses that were travelling together, parch, fare, class, deck, embark_town and whether the traveler was alone.

We loaded the data using Pandas and pass the NumPy arrays to Tensor Flow. If you want to scale if you want to scale up to a large set of files then we will have to use make_csv_dataset() function which is currently in the experimental version of TensorFlow.

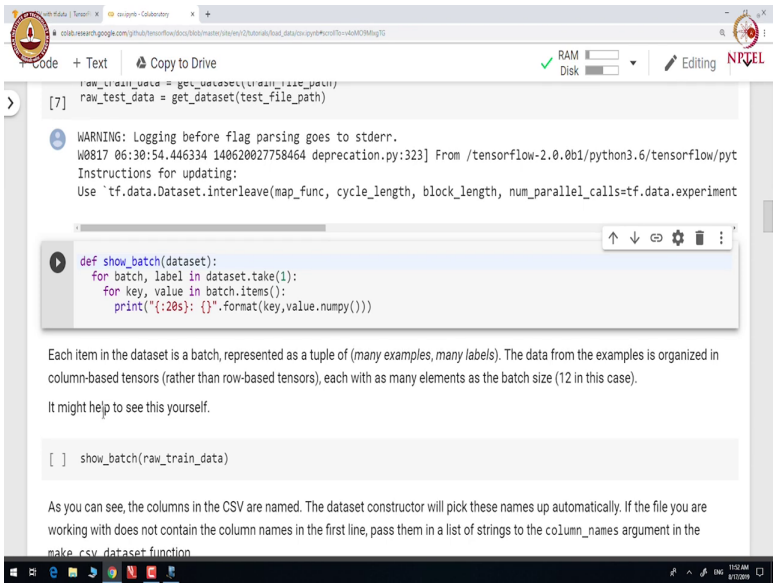
(Refer Slide Time: 03:13)




We separate the label column and as we said there are two labels 0 or 1. Let us read the file and create a dataset. Here we demonstrate the make_csv_dataset() function which can be used for large datasets the current dataset is pretty small, but still we decide to demonstrate make_csv_dataset() function with Titanic dataset because this will be applicable for large CSV files as well. Here we are taking a small batch size internationally, so that we can examine the examples in each batch. We specify the label name, specify how null values are predicted how null values are specified with question mark here in this case and number of epochs we set it to 1.

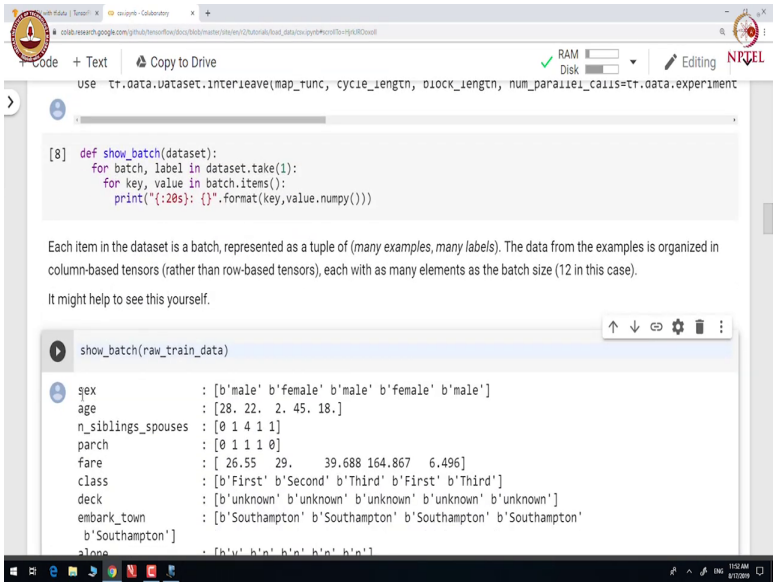
We construct the training data by giving the training file to get_dataset() function and test data by providing the path of the test file to the get_dataset()function. The get_dataset() function makes a dataset from CSV file and returns the dataset.

(Refer Slide Time: 05:29)




Each item in the dataset is a batch represented as a tuple of examples and labels. The data from the examples is organized in column based tensors each with as many elements as the batch size.

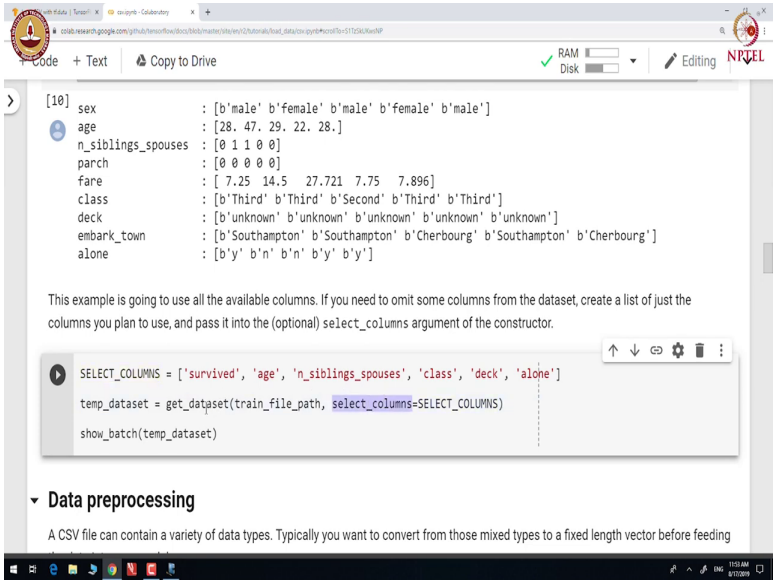
(Refer Slide Time: 06:07)




Let us look at one of the batch. You can see that since you set out the batch size to 5 for each of the feature you have exactly 5 values. Each feature is organized as a column

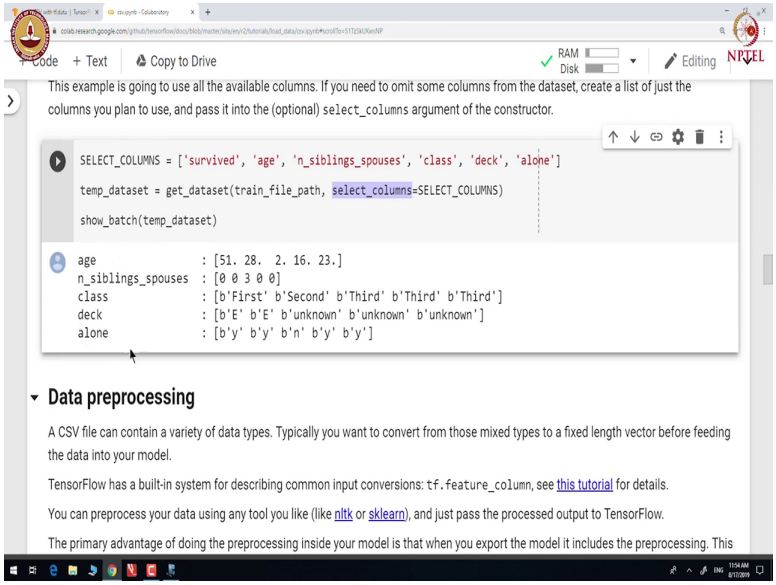
based tensor. So, you can see that there is one tenser for feature, one tenser for sex, another for age and so on, and in each of the tensor you will find exactly five values.

(Refer Slide Time: 06:43)



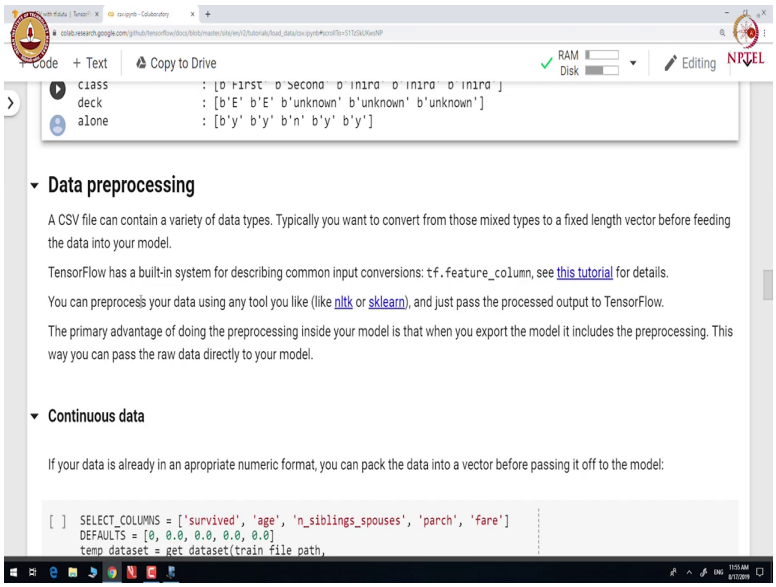

We can specify the columns that we want to use to construct a dataset and pass that in the *select_columns* argument of the get_dataset() function that we wrote earlier.

(Refer Slide Time: 07:09)



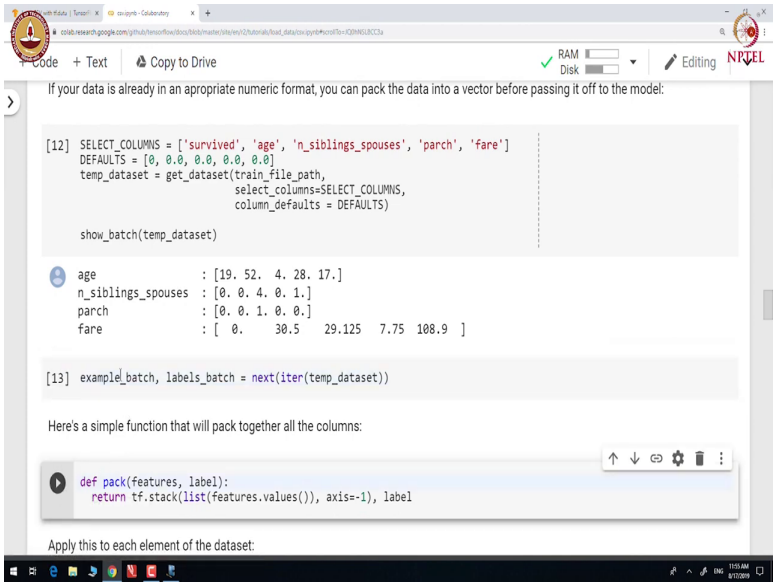

And, since we choose only hand full of the features we can see that there are only those many features that were selected are shown here.

(Refer Slide Time: 07:31)




A CSV file can contain a variety of data types and typically you want to convert the mix type to a fix length vector before feeding the data into the model. So, TensorFlow uses a construct called *tf.feature_column* for such kind of convergence.

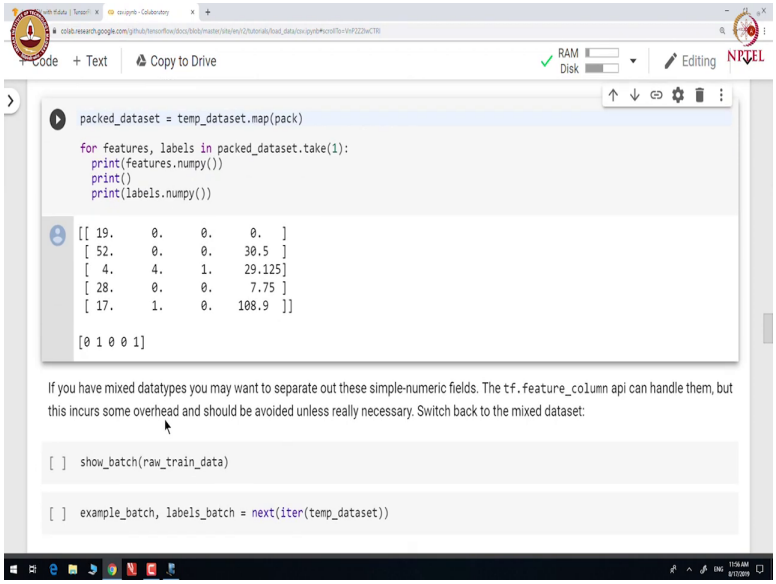
(Refer Slide Time: 08:01)




If a data is already in a numeric format we can pack the data into a vector before passing it out of the model. So, we select columns which are numeric is specify by the default values and we batch the dataset. We can look at example batch and the label batch. We

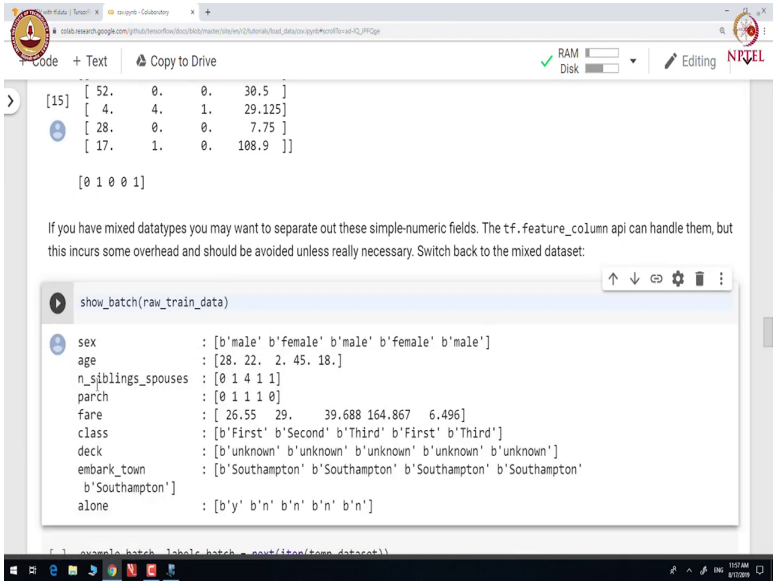
will pack together all the columns and we apply this to each element of the dataset. We apply the pack() function to each element in the dataset.

(Refer Slide Time: 08:59)

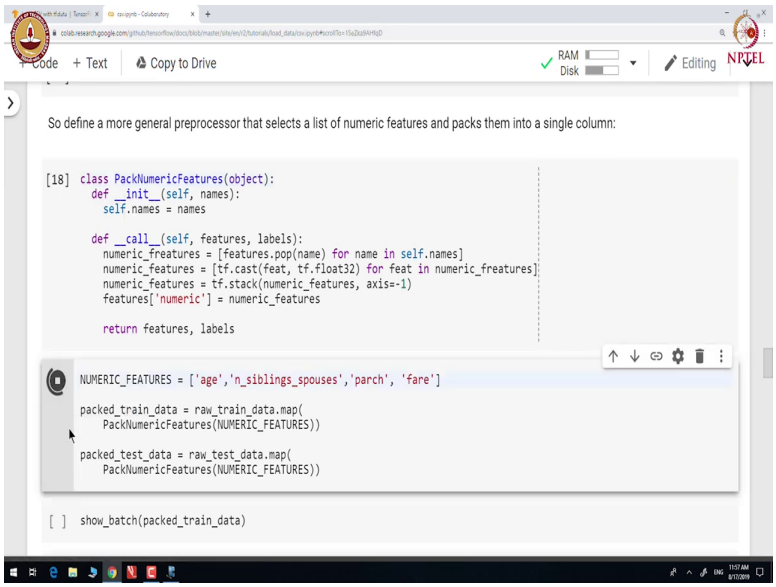



We can see that we have a feature metric or a 2D tensor of features and a 1D tensor of labels. If we have mixed data types we may want to separate out this simple numeric features. The *tf.feature_column* API can handle them, but this incurs some overhead and should be avoided unless really necessary.

(Refer Slide Time: 09:35)

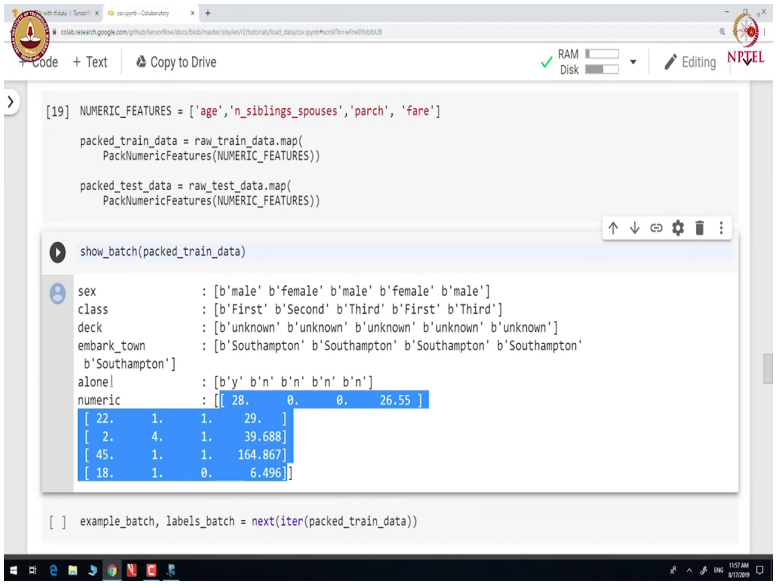


(Refer Slide Time: 09:43)



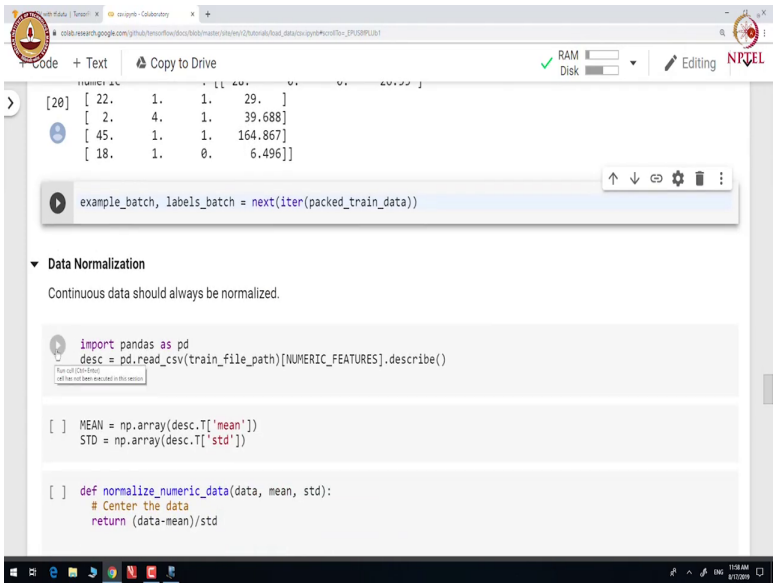

Let us define a more general preprocessor that selects a list of numeric features and packs them into a single column. We take the numeric features and pack them.

(Refer Slide Time: 10:11)



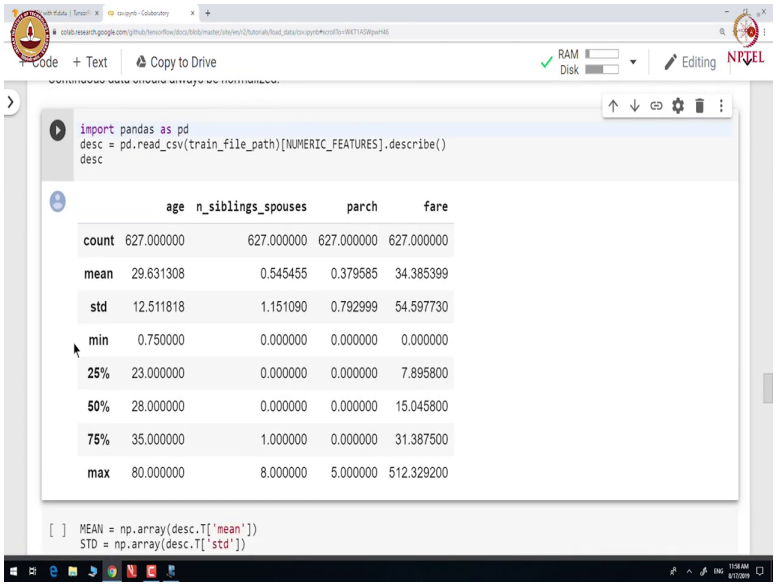

You can see that all the numeric features are packed into a tensor, whereas the other features are kept separate.

(Refer Slide Time: 10:37)



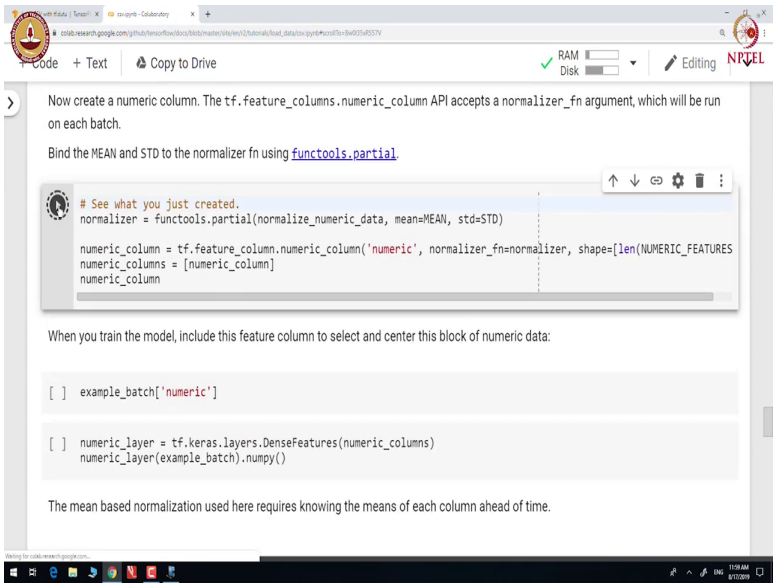

We need to normalize the continuous data.

(Refer Slide Time: 10:43)



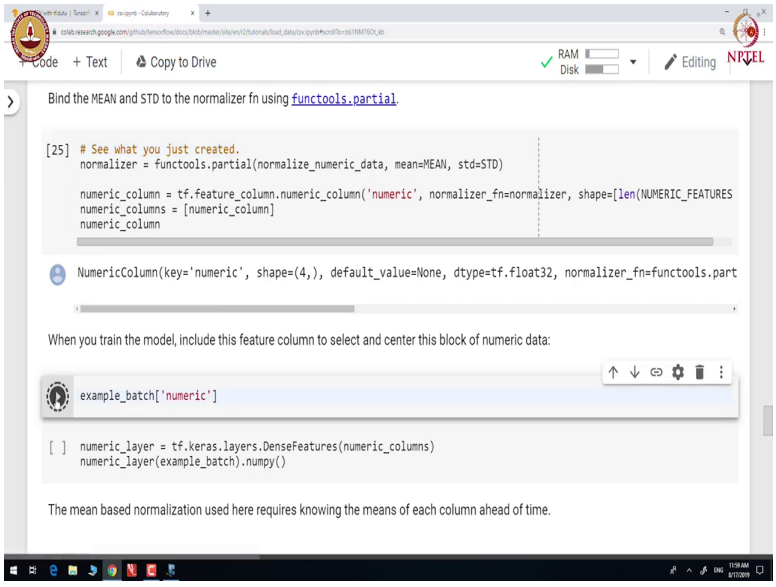

One of the ways of normalizing is by using z-score normalization that we studied earlier. Let us see how to do the z-score normalization on the numerical data or on continuous data. So, if first give the mean and standard deviation of each continuous column and then define a normalized function that centers the data.

(Refer Slide Time: 11:23)

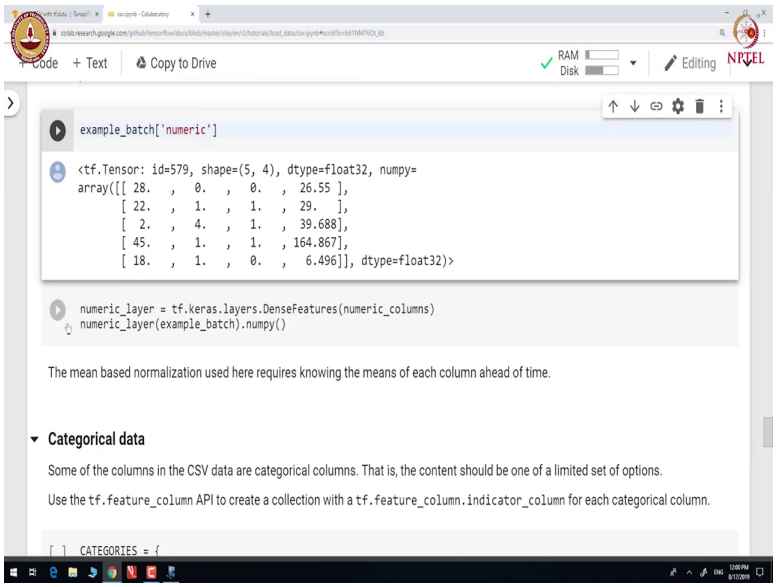



We create a numeric column with *tf.feature_columns.numeric_column* API and pass the normalization function as an argument. The normalization will be run on each batch.

(Refer Slide Time: 11:57)

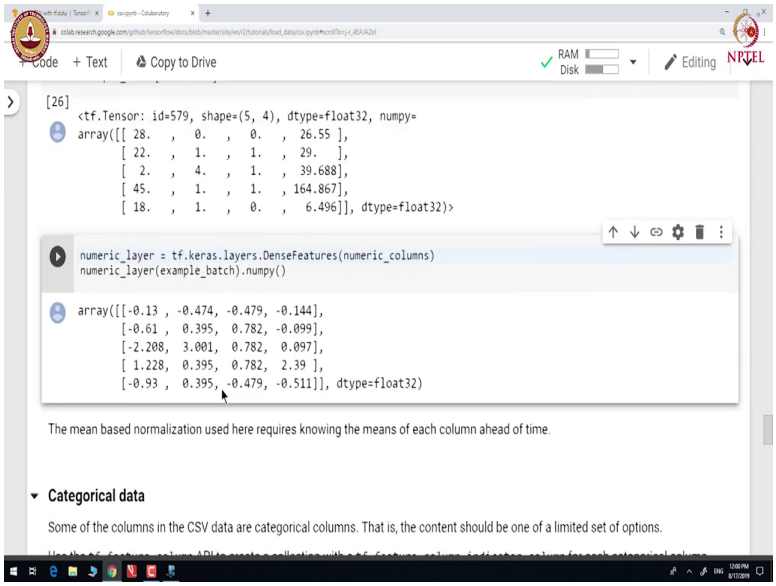


(Refer Slide Time: 11:59)



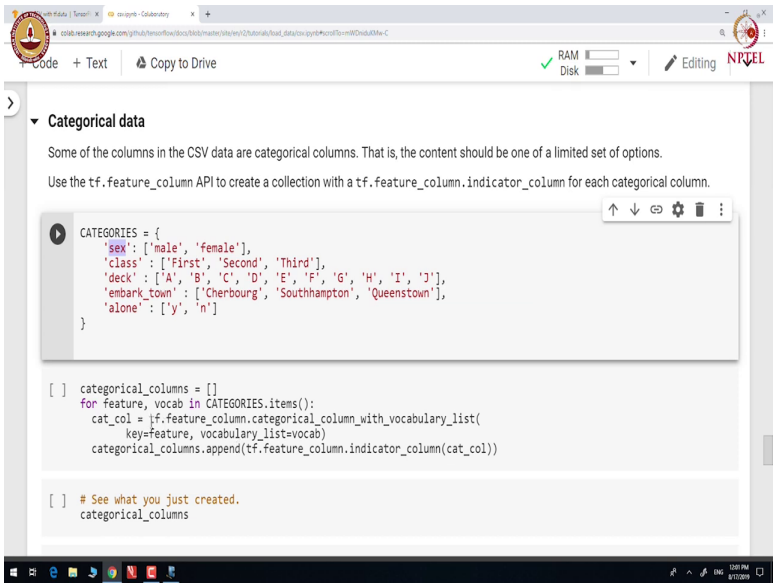

Let us put all the numeric columns in the dense features and pass it to the model.

(Refer Slide Time: 12:15)



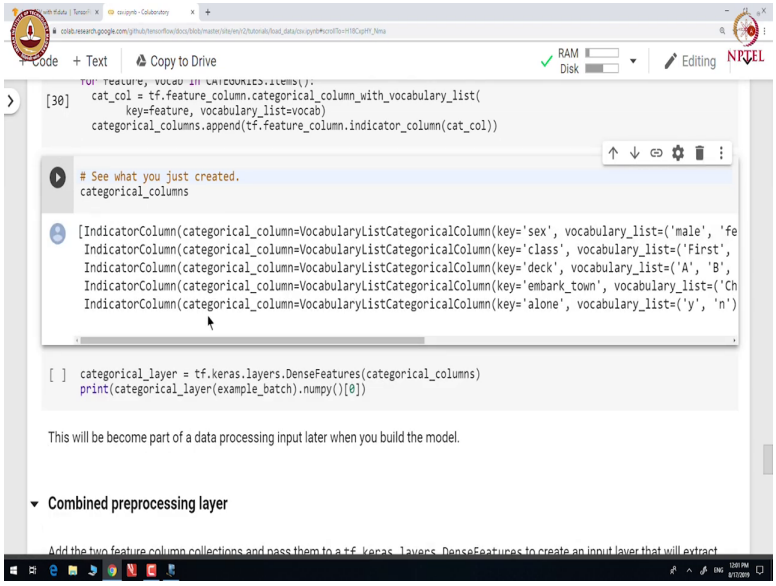

So, you can see that the numeric layer has normalized data for each of the numeric features. Let us see how to handle the categorical variable.

(Refer Slide Time: 12:35)



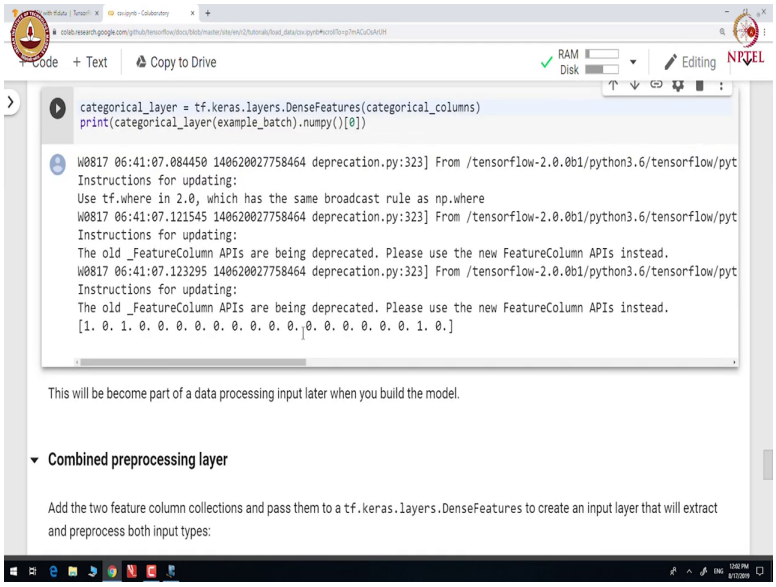

We use tf.feature_column.indicate_column to convert each categorical variable into a one hot encoding. We construct dictionaries of values in each of the categorical variables. For example, sex has male and female in it, class has first second and third in it. There are A to J decks then there are few embarkation towns, and alone has two values yes or no. Then we take each of these categorical variables along with their vocabularies and convert as categorical variables into indicator columns using the vocabulary list.

(Refer Slide Time: 13:39)



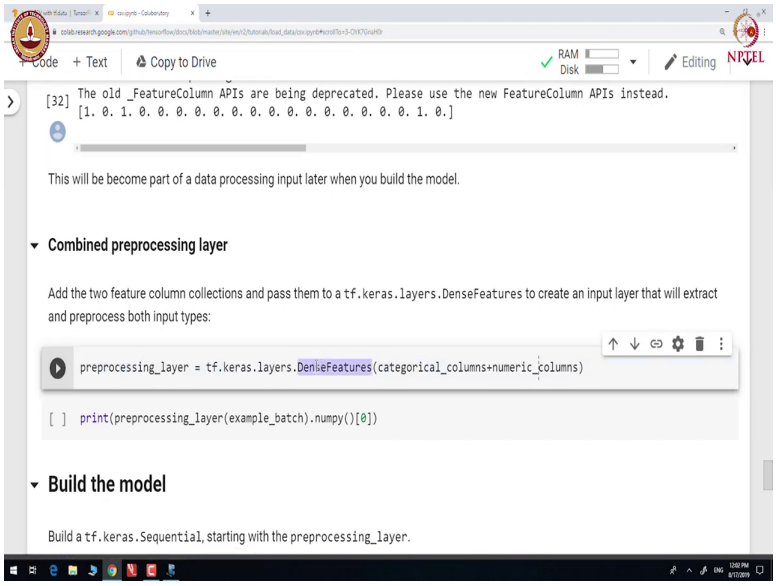
We can see that there are indicator columns that are created from this exercise.

(Refer Slide Time: 13:57)



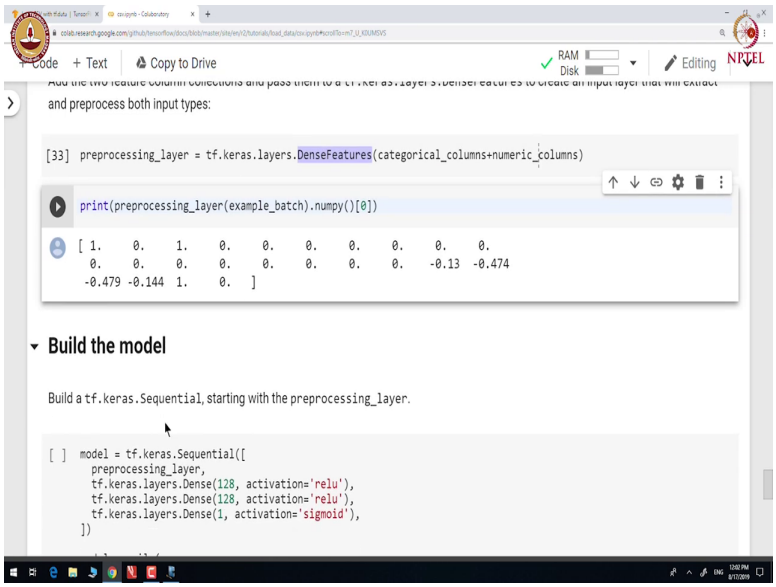

We also create a dense feature layer out of the categorical columns.

(Refer Slide Time: 14:05)

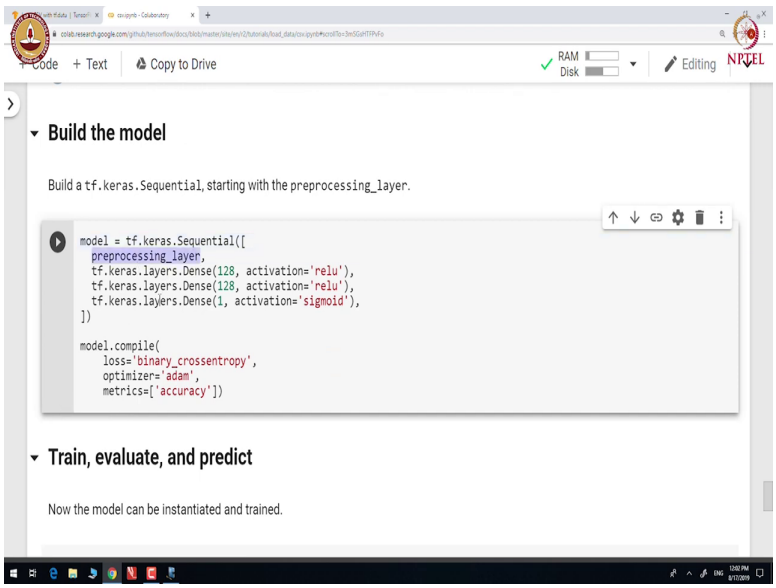



Since the categorical columns are converted into numbers you can combine categorical columns and numerical columns into dense feature layer and pass it to the model.

(Refer Slide Time: 14:27)

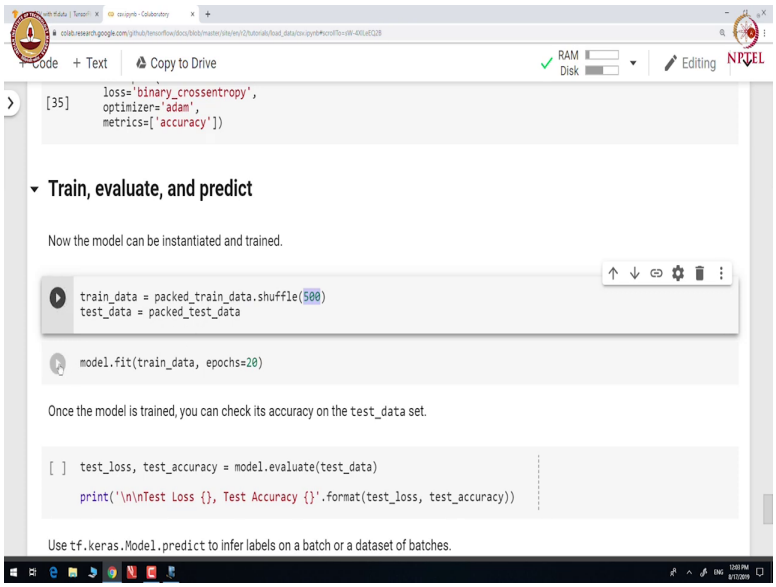



(Refer Slide Time: 14:37)



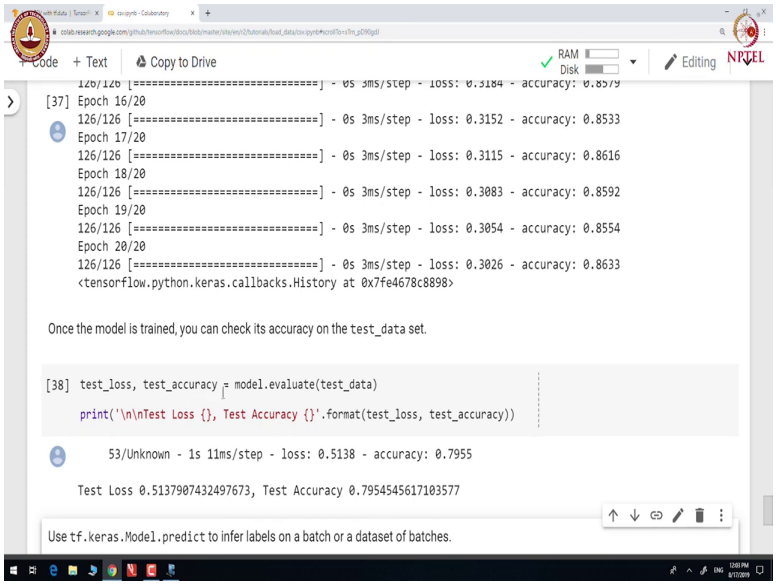

You can see that the preprocessing layer which is a dense feature layer is passed to a model. This is how we handle the dataset creation using CSV and preprocessing. Let us check how do we do shuffling and batching.

(Refer Slide Time: 15:03)



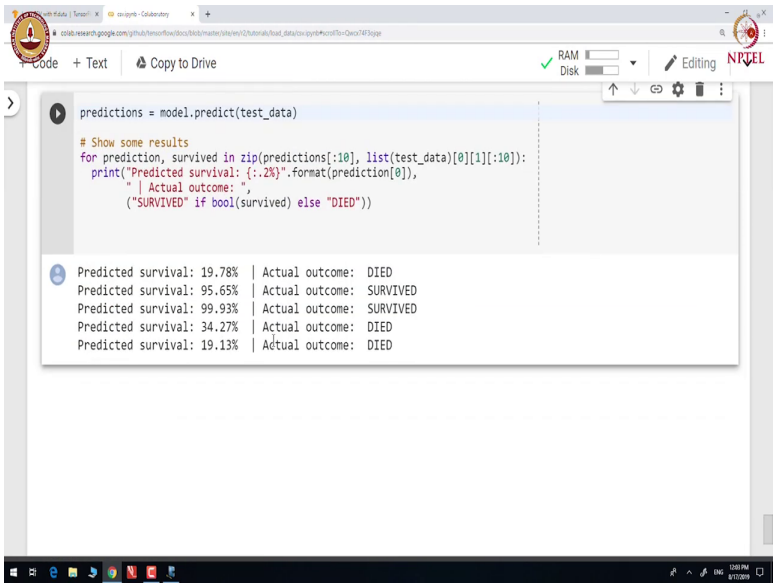

So, we can simply call a shuffle transformation on the training data by specifying the buffer size.

(Refer Slide Time: 15:29)



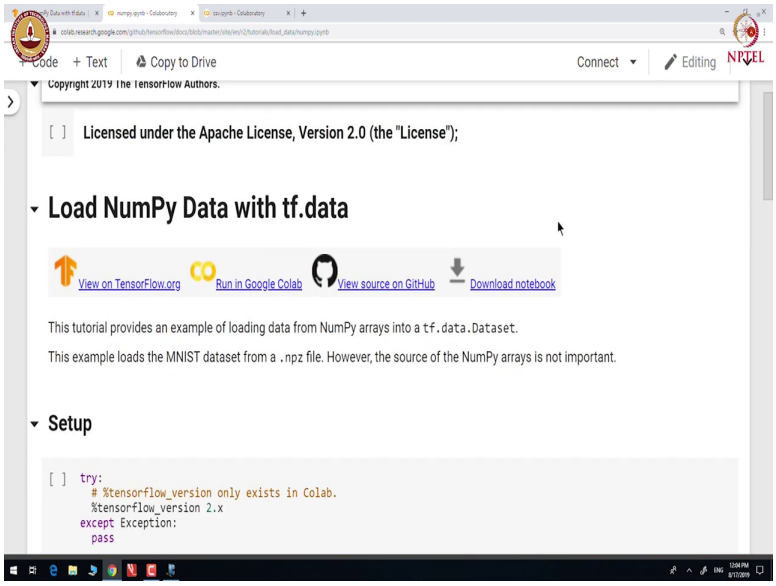

Then we can evaluate the model on the test data and see the accuracy.

(Refer Slide Time: 15:39)



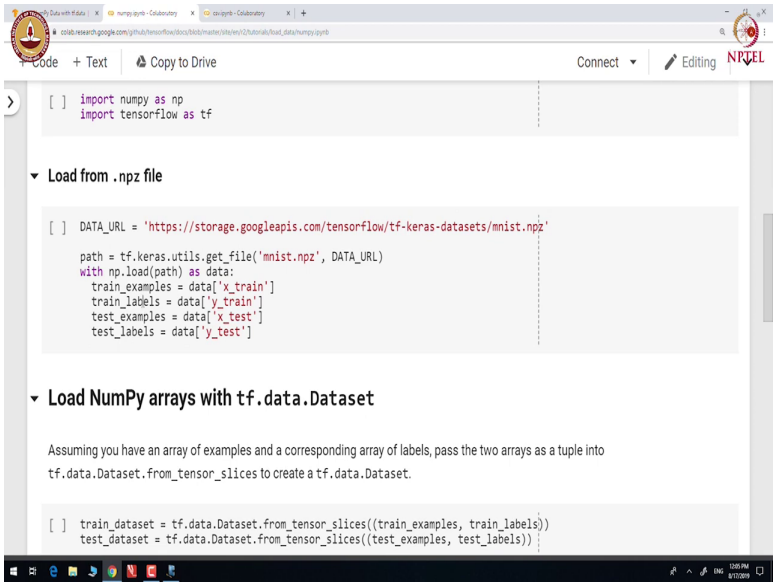

You can also check out the model predictions and compare that against the actual outcome. We will be doing more and more of model tanning and evaluation in the subsequence secessions. So, we are not giving much stress on that particular part. In the sessions more stress is given on creating input pipelines from multiple sources. Let us look at another source of creating input pipeline, this time with NumPy.

(Refer Slide Time: 16:19)



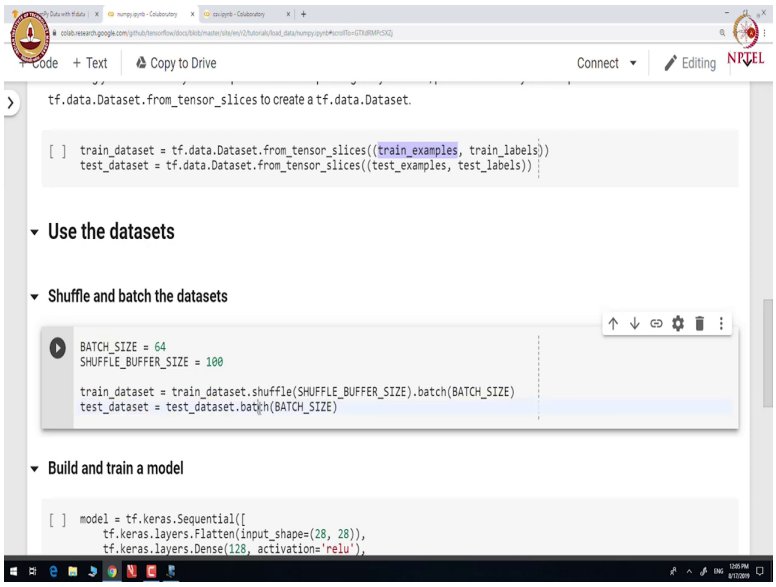
So, here the data is stored in NumPy array and you want to construct the dataset from this NumPy arrays. We read the NumPy array where we have Fashion-MNIST dataset which is stored as NumPy array.

(Refer Slide Time: 16:59)



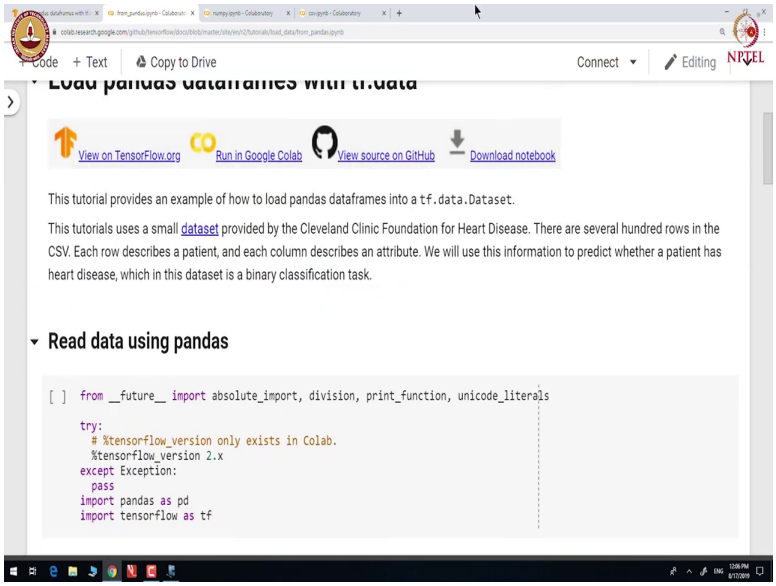

We read that we get train_examples, test_examples, train_labels and test_labels corresponding to train examples, test examples, training labels and test labels.

(Refer Slide Time: 17:13)



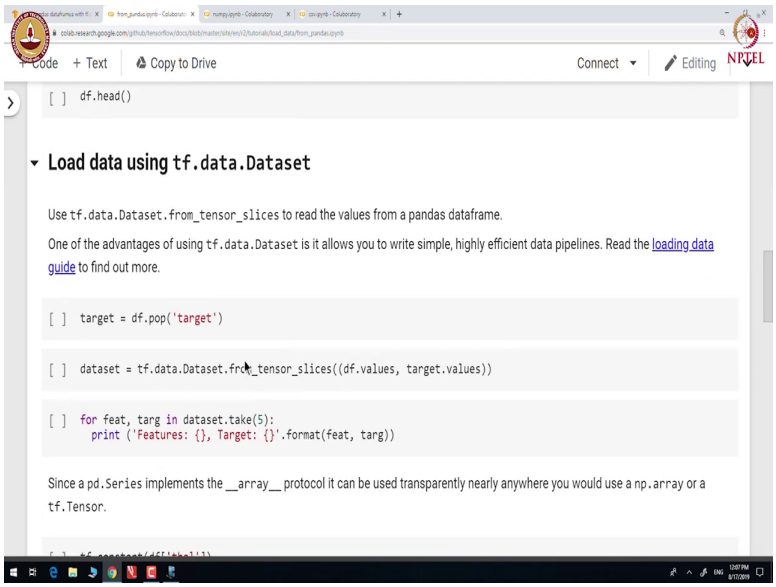
We use *from_tensor_slices* from the dataset API for constructing datasets from NumPy arrays. Finally, we can do shuffling and batching of the training dataset, whereas on the test dataset we can apply let say batch transformation and then you can build a model and evaluate its accuracy on the test dataset.

(Refer Slide Time: 18:03)



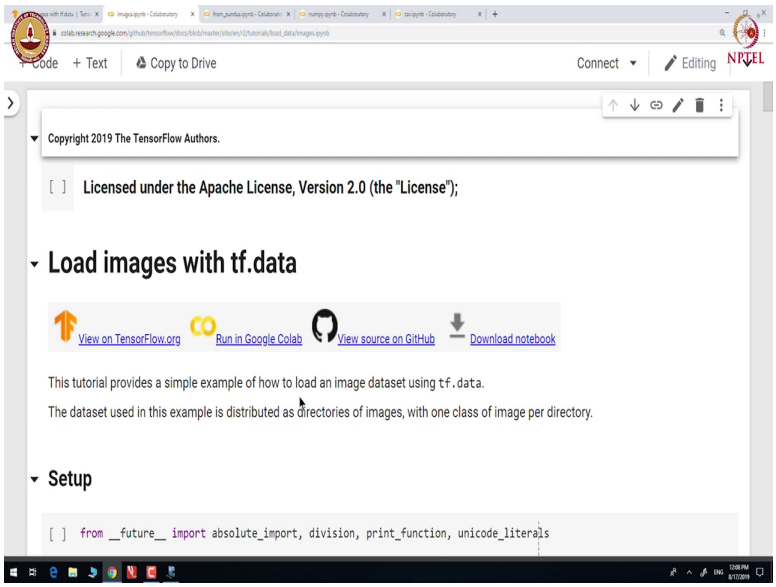

Let us look, how to construct datasets from panda's data frames. Here we have a dataset provided by Cleveland Clinic Foundation for Heart Disease. There are several hundred rows in the dataset. The dataset is in CSV format. Each row describes a patient and each column describe an attribute of a patient. We will use this information to predict whether a patient has heart disease or not.

(Refer Slide Time: 18:55)



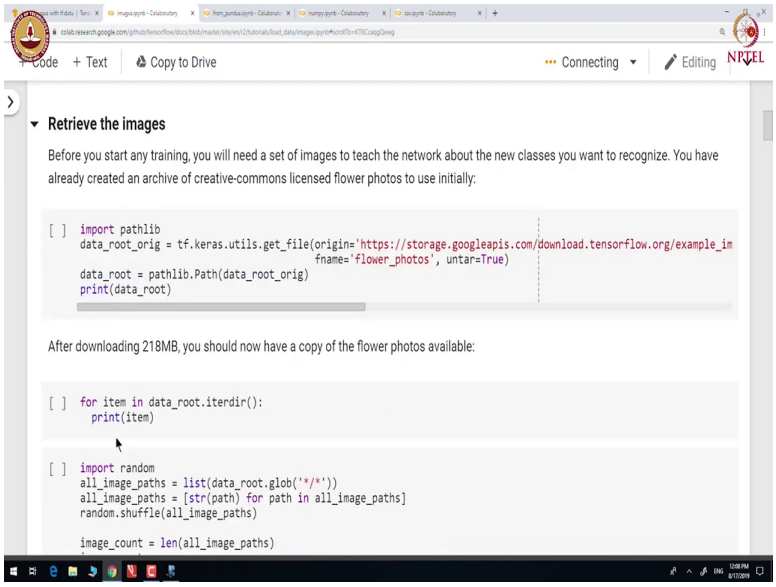

So, let us look at the steps to create a dataset. We use *dataset.from_tensor_slices* to create the dataset from the Pandas DataFrame. So, in NamPy and Pandas whether data is sitting in memory we use from_tensor_slices() function of dataset api to create data sources.

(Refer Slide Time: 19:21)



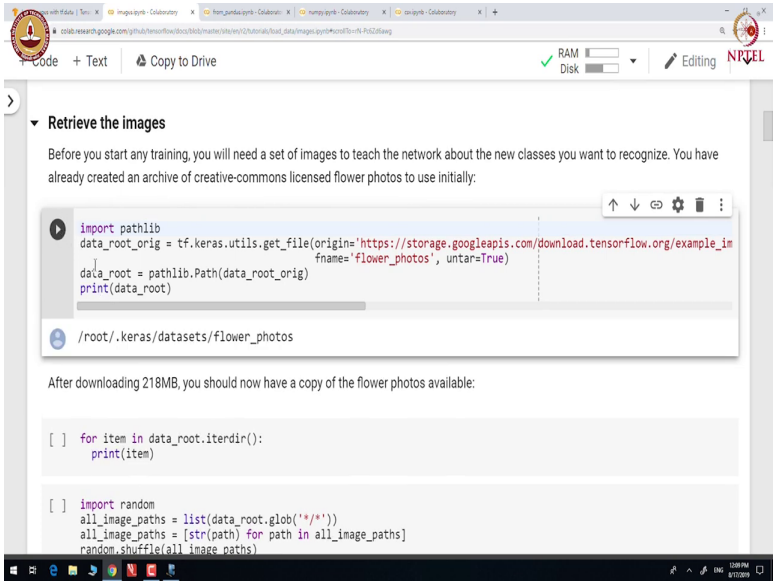
Let us try to understand how to create a dataset from an image dataset. The image class is provided in the directory name and we have multiple directories of the images. So, let us see how to create input data pipeline for such a data through this example.

(Refer Slide Time: 19:53)

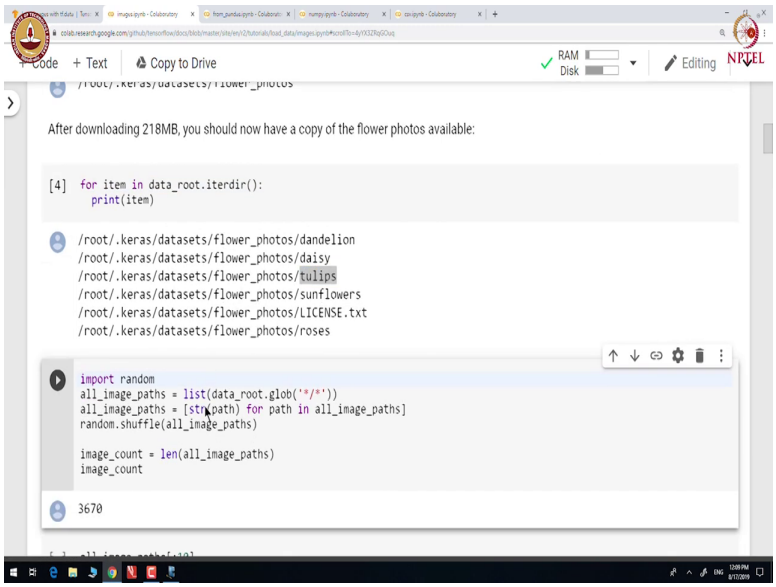



First download the file.

(Refer Slide Time: 20:07)

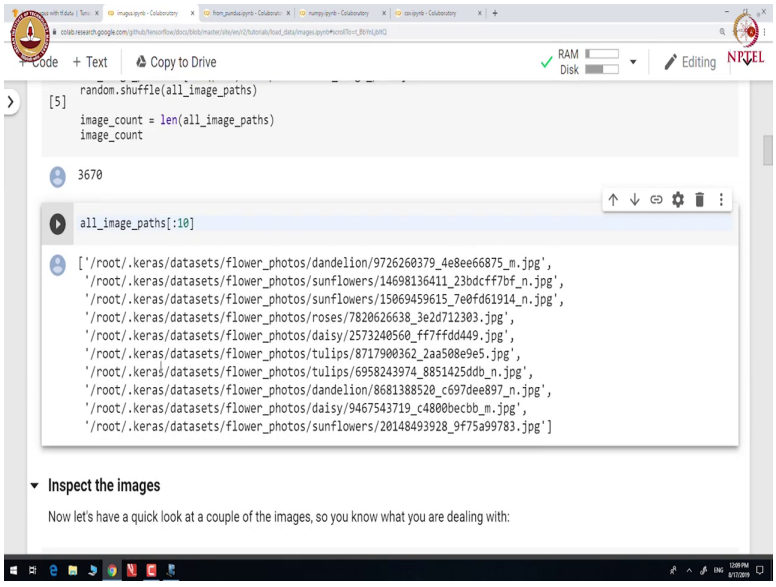


(Refer Slide Time: 20:15)



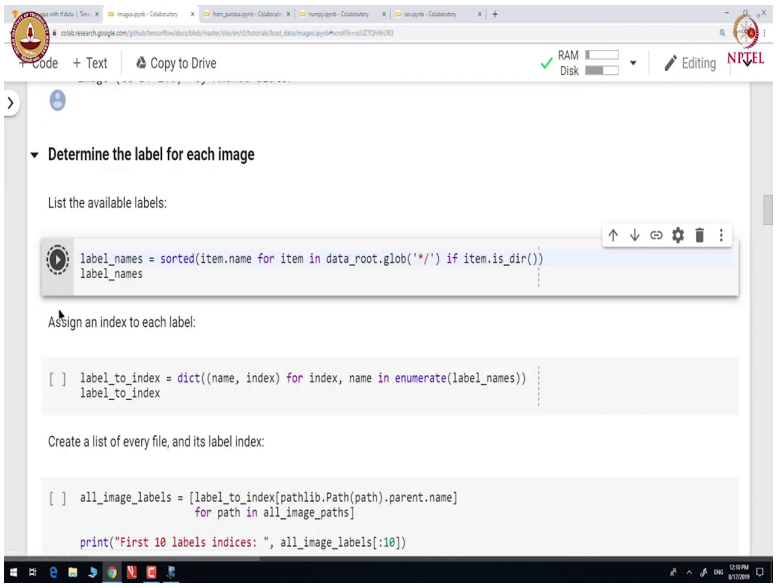

You can see that there are directories like dandelion, daisy, tulips which contain images of the respective flowers. There are 3670 images in all.

(Refer Slide Time: 20:45)



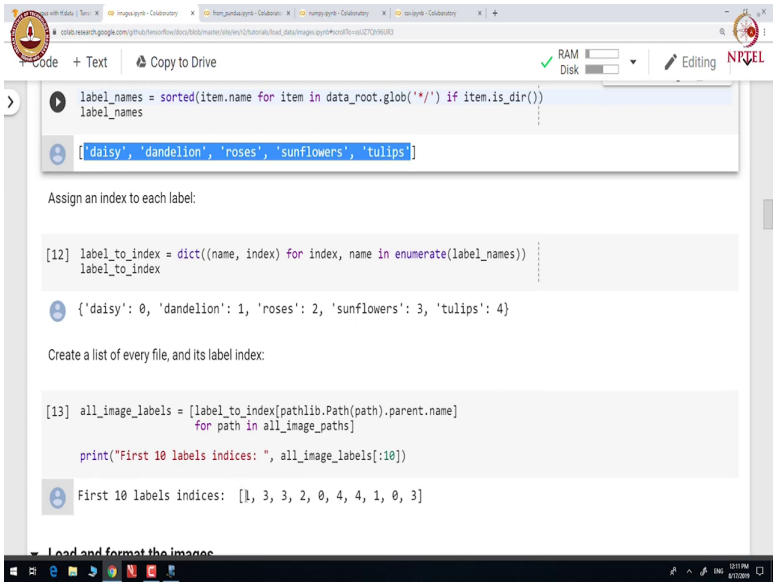

And, these are top ten image paths. You can see that there are few images of dandelion flower, some images of sun flowers, some of roses, tulips, daisy and so on. Let us look at couple of images.

(Refer Slide Time: 21:07)



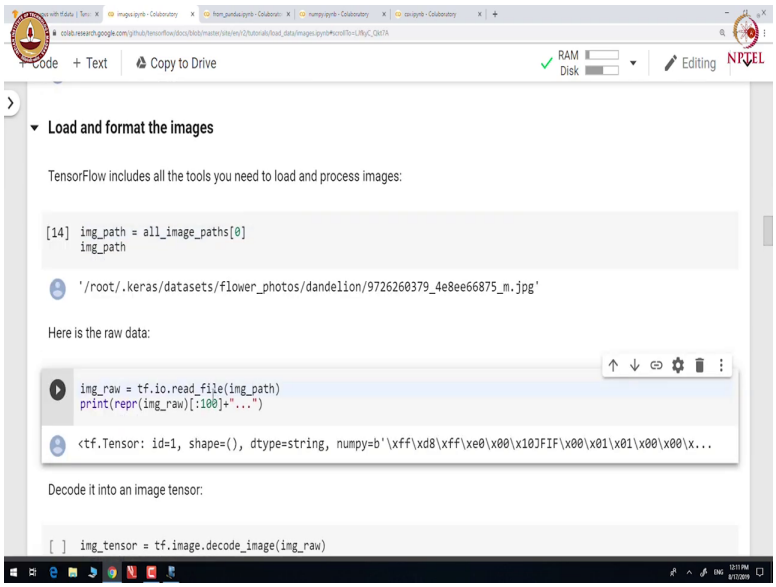

We will first assign the label to each of the images.

(Refer Slide Time: 21:13)



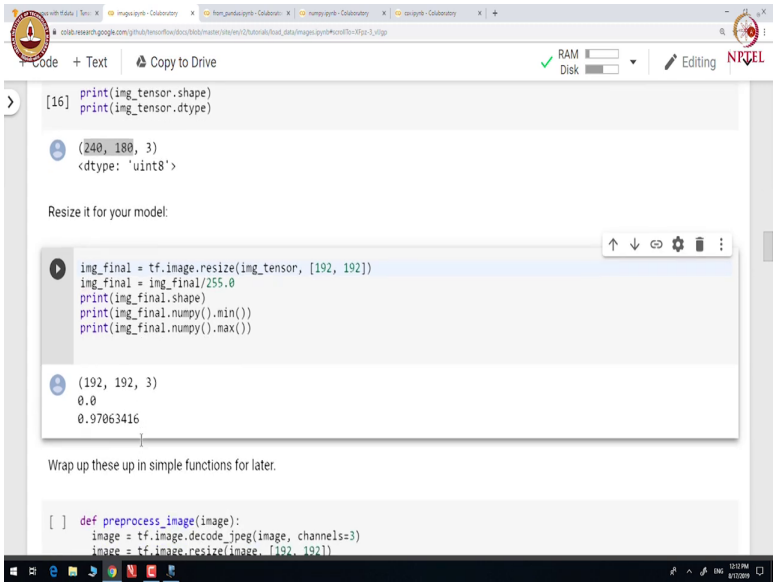

For this what we did is we took the string labels and assign an index to each of the labels. So, we have converted those strings into numbers.

(Refer Slide Time: 21:41)



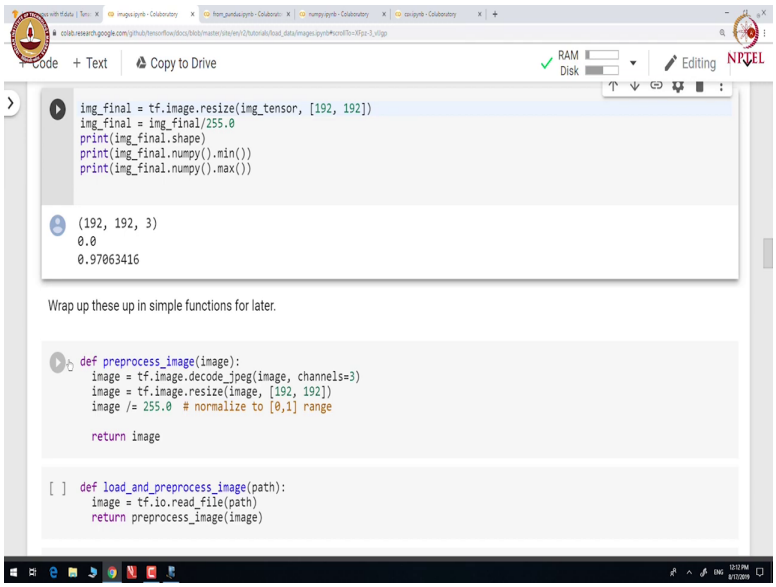

We can read the image using tf.io.read_file() function by providing the image path.

(Refer Slide Time: 21:59)



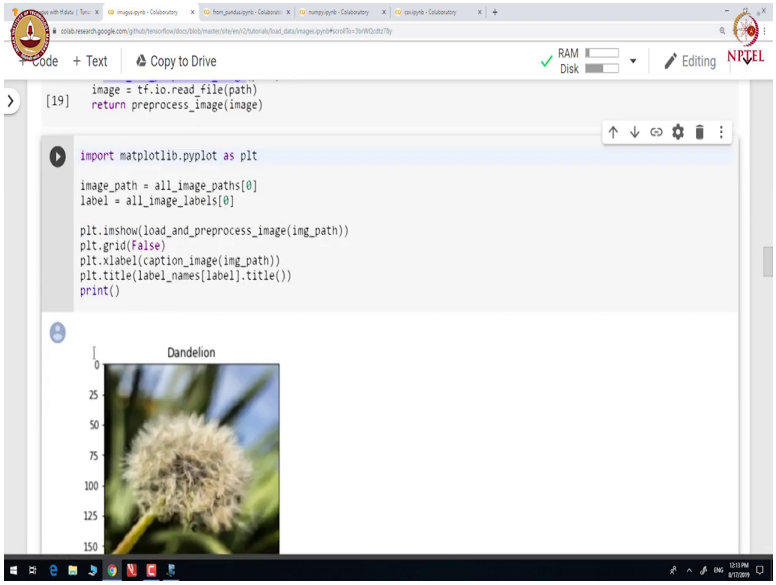

And, we can decode the image into an image tensor. So, we can see that an image tensor is a 3D tensor which shape 240 x 380 x 3; there are three colour channels and each images of the size 240 x 180. We will resize each of the image to 192 x 192 and we normalize the image by dividing each pixel value by 255.

(Refer Slide Time: 22:37)



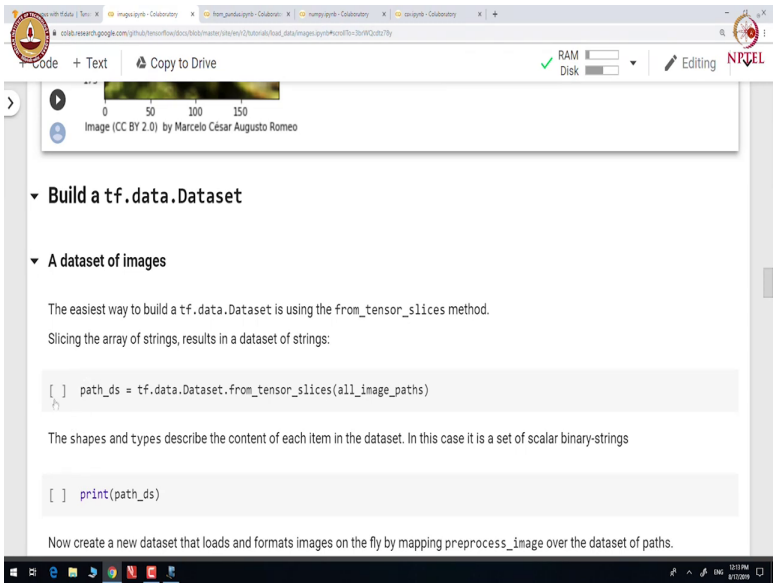

So, we wrap all these transformation into a simple function called preprocess_image where we first decode the jpeg file resize image and then normalize the image in the range between 0 to 1. We put load and pre process, we wrap it in load_and_preprocess_image() function.

(Refer Slide Time: 23:15)



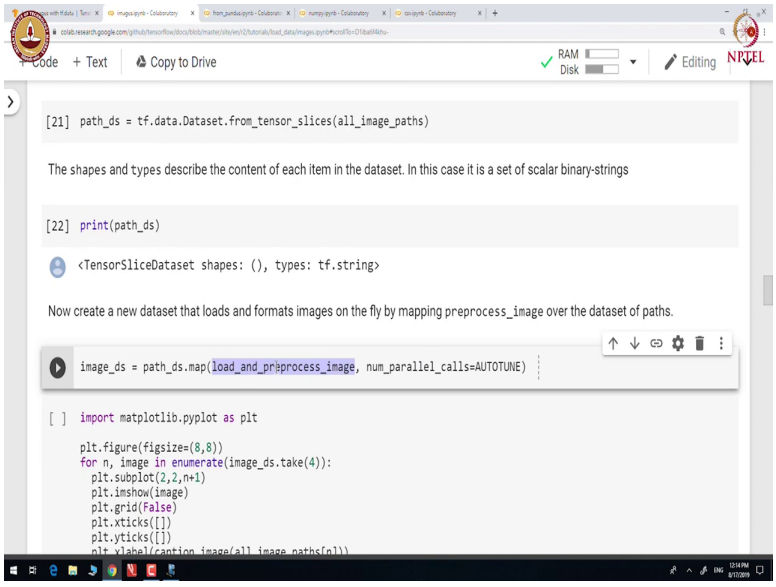

Let us look at some of these images after pre-processing.

(Refer Slide Time: 23:23)



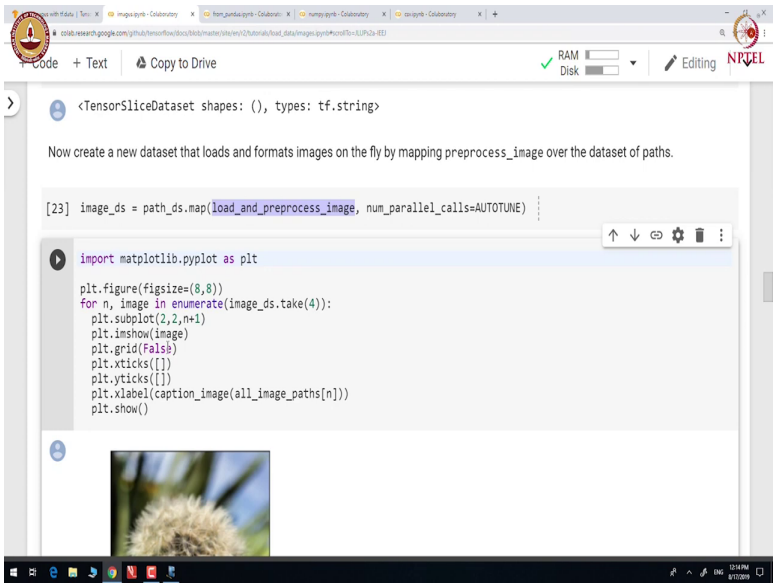

Let us see how to build a *tf.dataset* from this images. So, you first construct a dataset of paths. All the paths are in memory.

(Refer Slide Time: 23:37)



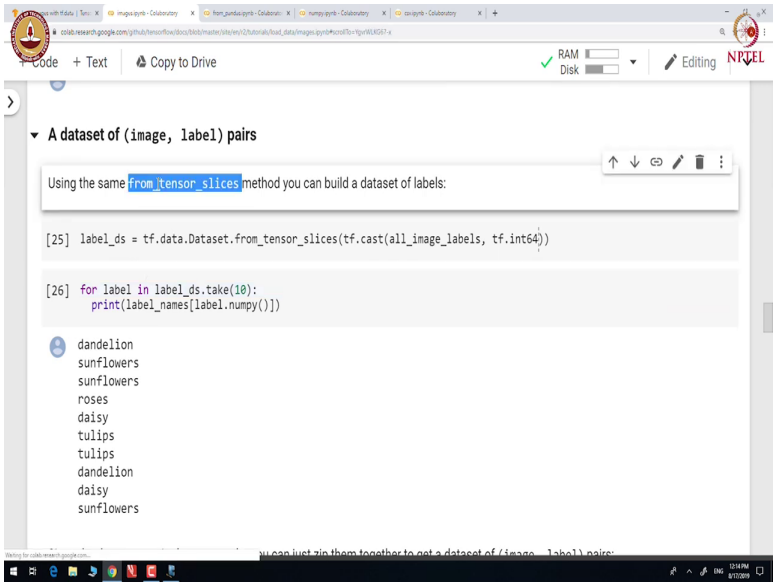

We use *from_tensor_slices*, as you are doing earlier in case of NumPy and Pandas datasets. Now, we create a new dataset that loads and formats image on the fly by mapping *preprocess_image* over the dataset of path.

(Refer Slide Time: 24:17)



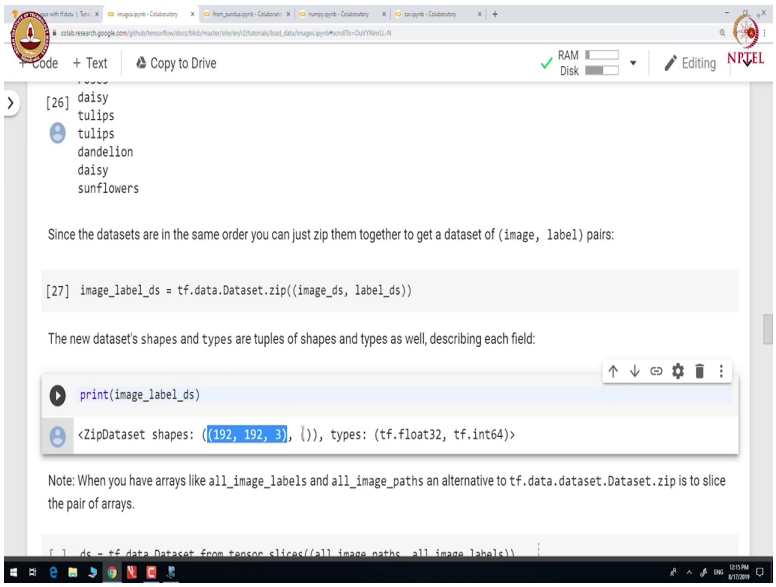

So, we use a map transformation and the and apply this function on each and every image in the dataset of paths. Let us look at some of the images from the dataset. We look at first four images. There are images along with their descriptions.

(Refer Slide Time: 24:43)



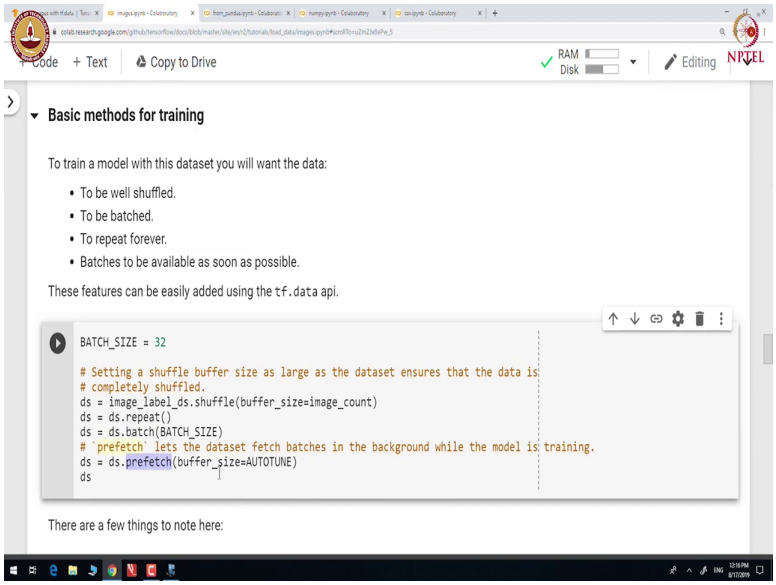

We build dataset of labels next using *from_tensor_slices* function.

(Refer Slide Time: 24:55)




We use dataset.zip to assign label to each of the image. And, you can see that each images of the size 192 x 192 x 3 and each label is a scalar.

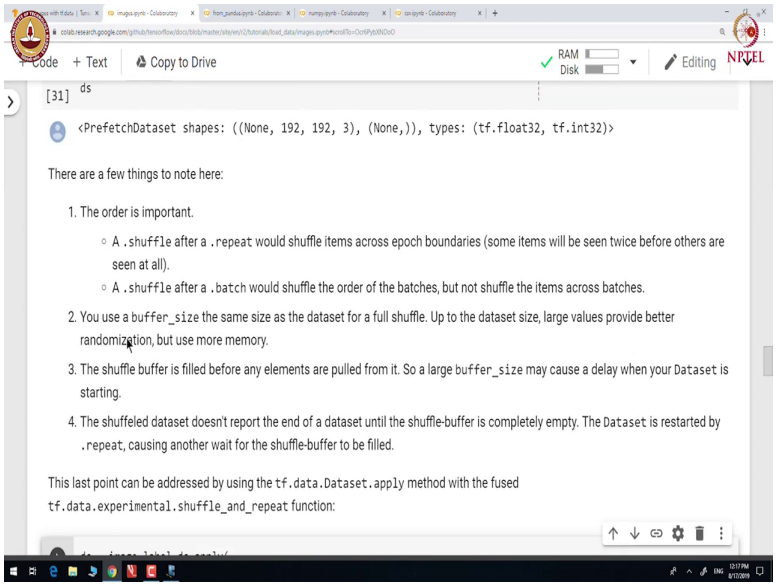
(Refer Slide Time: 25:27)




In order to train a model with this dataset, we want to first shuffle the data and then we want to batch it and repeat forever and we want to make sure that batches are available as soon as possible.

So, we use shuffle() transformation for shuffling the data by giving sufficient buffer size, then we use repeat() for repeating the dataset, epochs after epochs; specify the batch and we use a prefetch() function that clears the dataset page batches in the background while model is training.

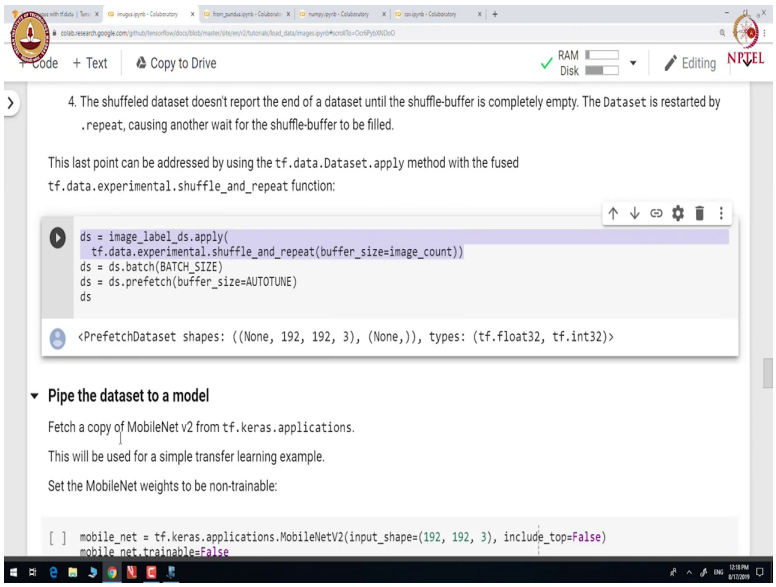
(Refer Slide Time: 26:15)




There are few points you note here; the order is important. A shuffle after repeat would shuffle items app across epoch boundaries. Some items will be seen twice before others are seen at all. A shuffle after batch would shuffle the order of the batches, but not shuffle the items across the batches. We use *buffer_size* of the same size as a dataset for a full shuffle up to the dataset size large values provide better randomization, but they need more memory. Shuffle buffer is filled before any elements they are pulled from it.

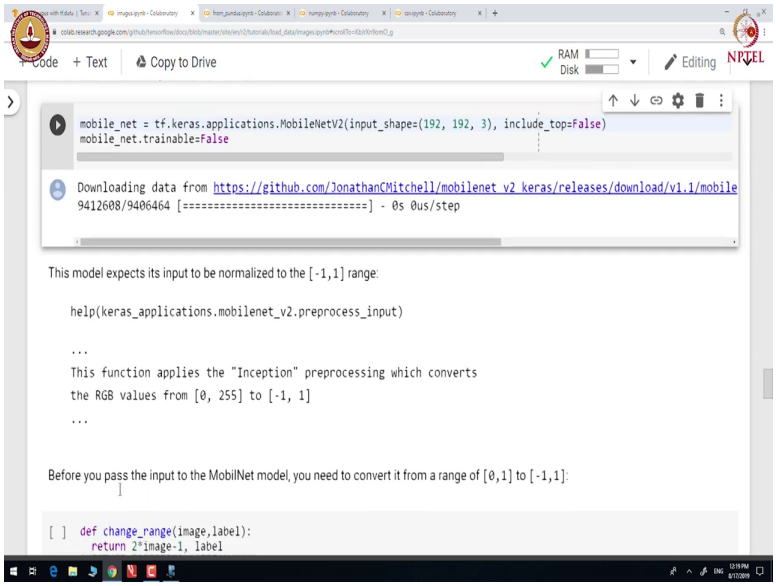
Large buffer size may cause a delay when your dataset is starting. The shuffle dataset does not report the end of a dataset until the shuffle buffer is completely empty a dataset is started by repeat causing another wait for shuffle buffer to be filled. We use the *dataset.apply()* method to address the last point.

(Refer Slide Time: 27:31)



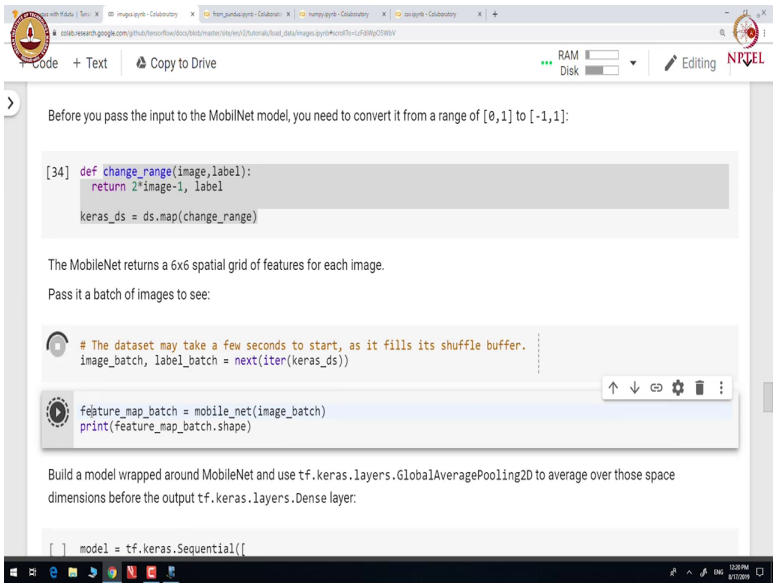

And, we fuse it with shuffle and repeat() function as seen here.

(Refer Slide Time: 27:43)



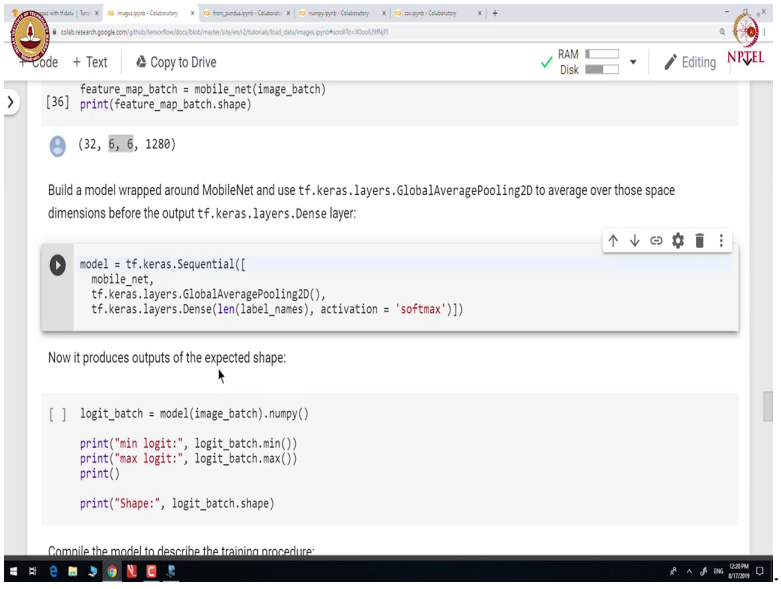

Finally, we pipe the dataset to the model and model and train the model. The MobileNet expects the input to be normalized to -1 to +1 range. So, we convert our data from 0 to 1 range to -1 to 1 range.

(Refer Slide Time: 28:19)



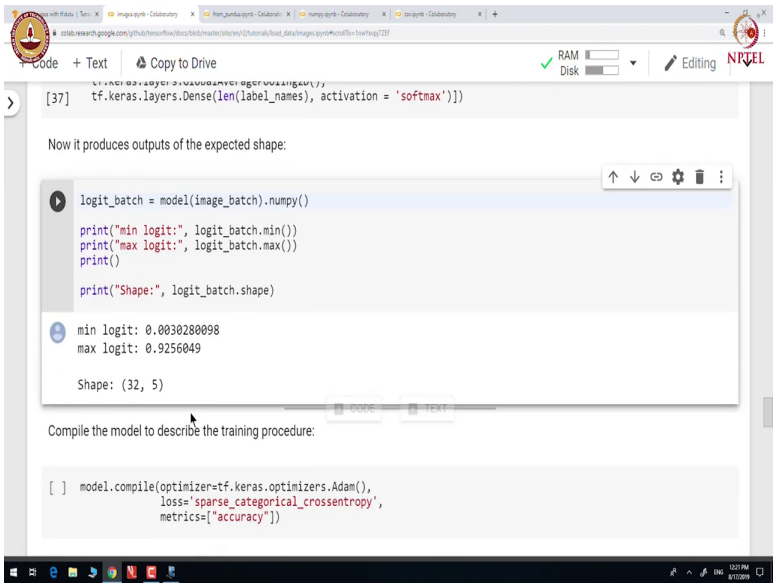

We define a change_range() function and apply the tern each element with map function. The MobileNet returns 6 x 6 special grade of features for each image. We pass it a batch of images to see the output of the MobileNet.

(Refer Slide Time: 28:43)



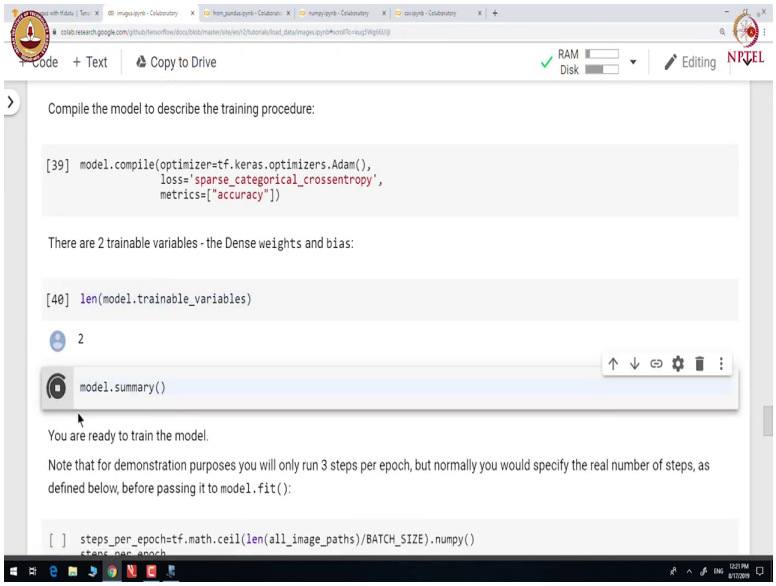

You can see that it returns 6 x 6 special grids. Finally, we built a model wrapped around MobileNet and GlobalAveragePooling2D to average over those special dimensions and then use a dense layer with softmax activation as output layer.

(Refer Slide Time: 29:33)



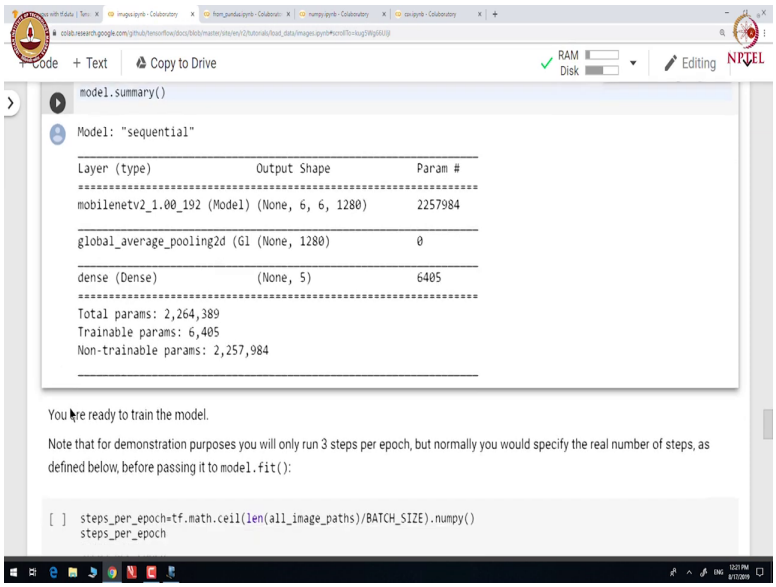

We can see that it produces output of the shape 32 x 5.

(Refer Slide Time: 29:45)

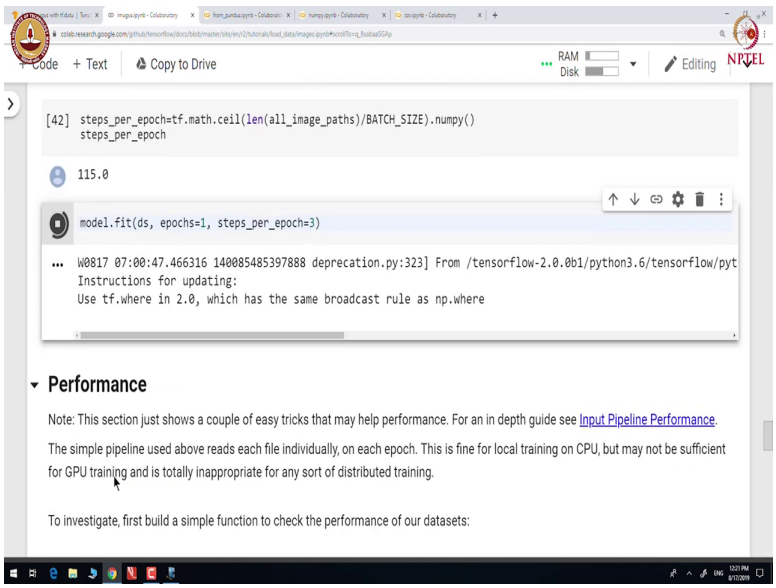



We compile the model to describe the training procedure and train the model.

(Refer Slide Time: 29:55)




(Refer Slide Time: 30:01)




For the demonstration purpose, we will only run three steps per epoch.

(Refer Slide Time: 30:11)