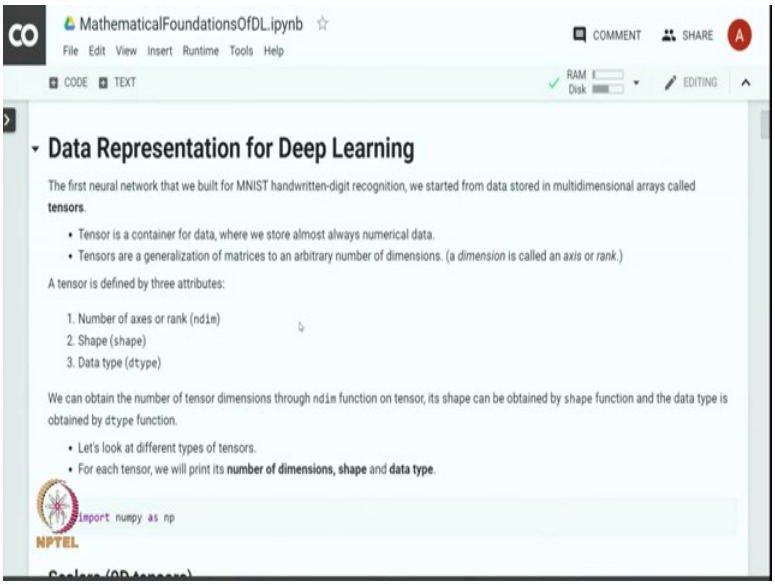
Practical Machine Learning with TensorFlow
Dr. Ashish Tendulkar Google
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

Lecture – 10
Introduction to Tensors

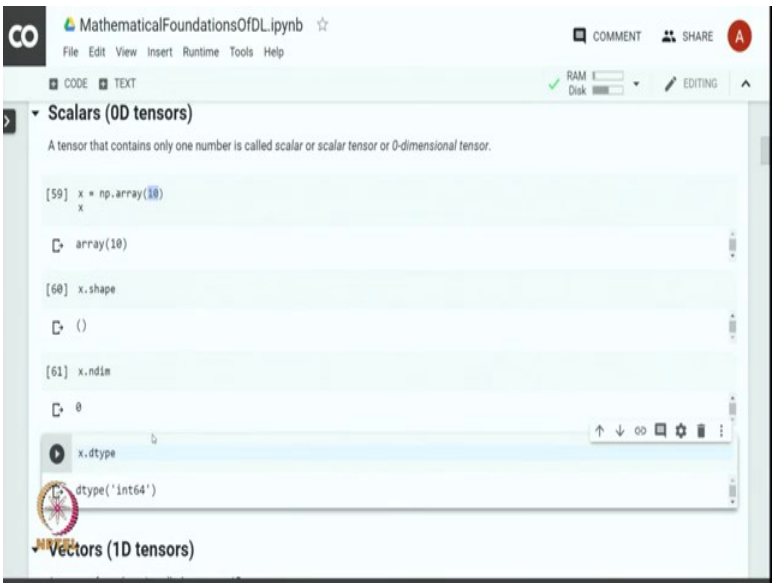
[FL]. This module focuses on mathematical foundations of deep learning and demonstrate these concepts through python code.

(Refer Slide Time: 00:35)



The first neural network that we built for MNIST handwritten-digit recognition. We started from data stored in multidimensional arrays called tensors. Tensor is a container for data where we store almost always numerical data. Tensors are a generalization of matrices to an arbitrary number of dimensions which are also called as axis or rank. A tensor is defined by 3 attributes: number of axis or ranks, shape and data type.

(Refer Slide Time: 01:38)

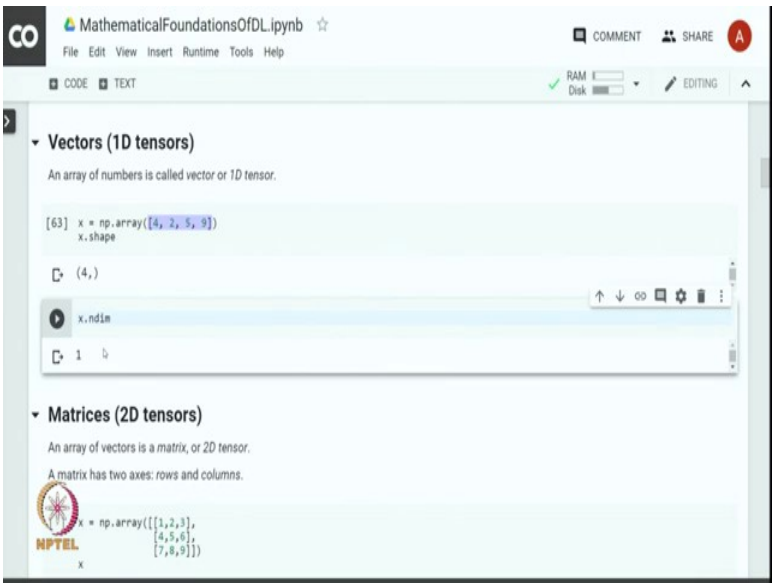


We can obtain the number of tensor dimensions through ndim() function on tensor. It shape can be obtained by shape function and data type can be obtained by dtype function.

For each tensor we will print the number of dimensions, shape and data types. We will import numpy library for some of the tensor operations. Let us first look at the simplest of the tensor which is 0-dimensional tensor, this tensor contains only one number. They are also called as scalar or scalar tensor or 0-dimensional tensor.

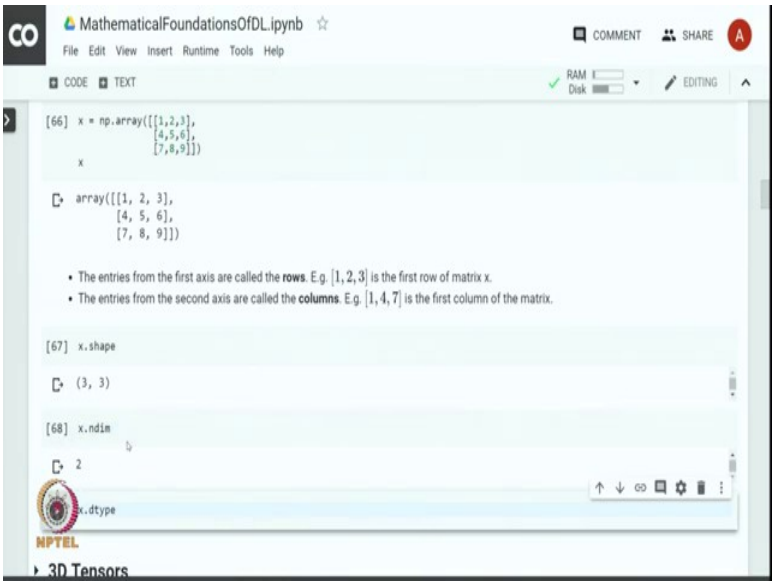
Let us take an example. Scalar tensor can be defined as follows we simply write np.array and we give a number. Let us execute this code. You can say that this is a scalar tensor, let us look at the shape of this tensor. You can see that since this is a scalar tensor the shape is empty. We expect the scalar tensor to have 0-dimensions and the data type in this tensor is a 64 bit integer.

(Refer Slide Time: 02:43)



Let us look at the next type of tensor which is 1D tensor, it is also called as vectors. We define 1D tensor with np.array function, where we give a vector as an argument. We have a list or vector here containing 4 elements. Let us execute this and look at the shape of this tensor. You can see that the shape of the tensor is (4,). So, this tensor has 4 elements in it. It is also called as a 4D vector. Let us look at the number of dimensions. Since, this is vector it has got one-dimension.

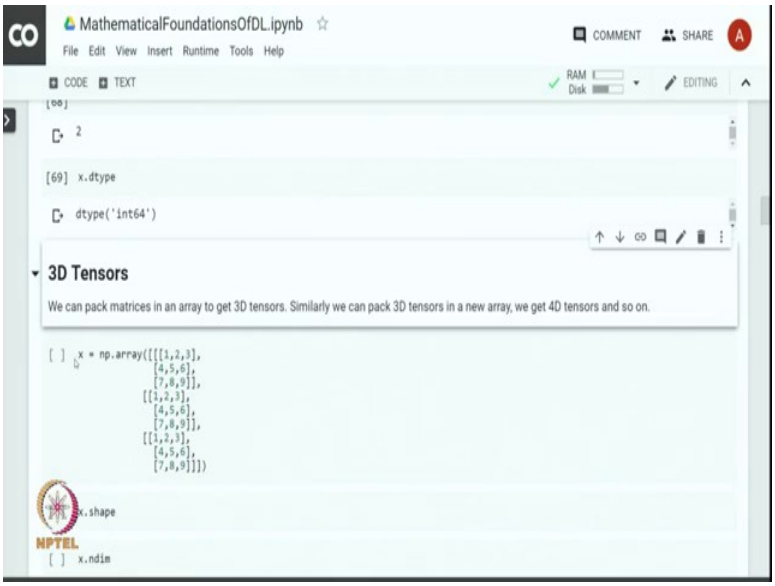
(Refer Slide Time: 03:41)



Let us look at the 2D tensors which are also called as matrix. We can define matrix as a vector of vectors. Here we have matrix which has got 3 rows and 3 columns. The entry from the first axis are called as rows, here 1, 2 and 3 constitutes the first row of the matrix X. The entries from the second axis are called the columns. So, 1, 4 and 7 is the first column of the matrix.

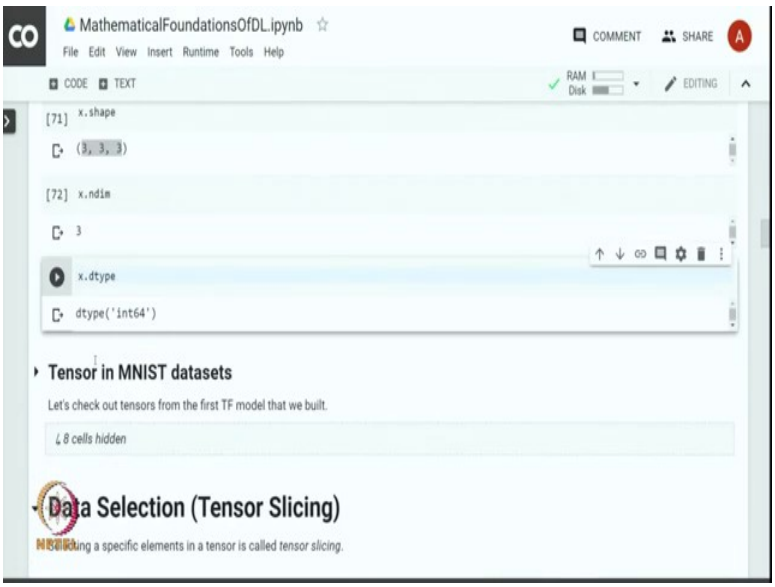
Let us look at the shape of the matrix. It is (3, 3), it is a 2D matrix. Let us look at the number of dimensions. Number of dimensions are two, and it holds 64 bit integers as elements of the tensor.

(Refer Slide Time: 04:35)



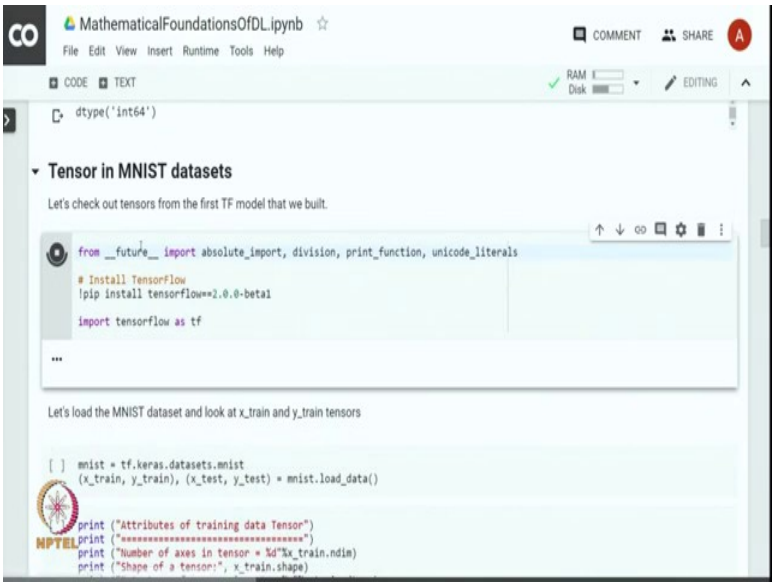
Let us look at the 3D tensors. We can pack matrices in an array to get 3D tensors. Similarly, we can pack 3D tensors in a new array to get 4D tensors and so on. We can pack tensors of *(n -1)* dimension in a new array to get a tensor of *n* dimension. Let us look at the 3D tensor. We have taken the matrix that we defined in the in the 2D tensor and we have copied it 3 times. So, we have packed the matrix in the array of matrix, so this becomes a 3D tensor for us.

(Refer Slide Time: 05:31)



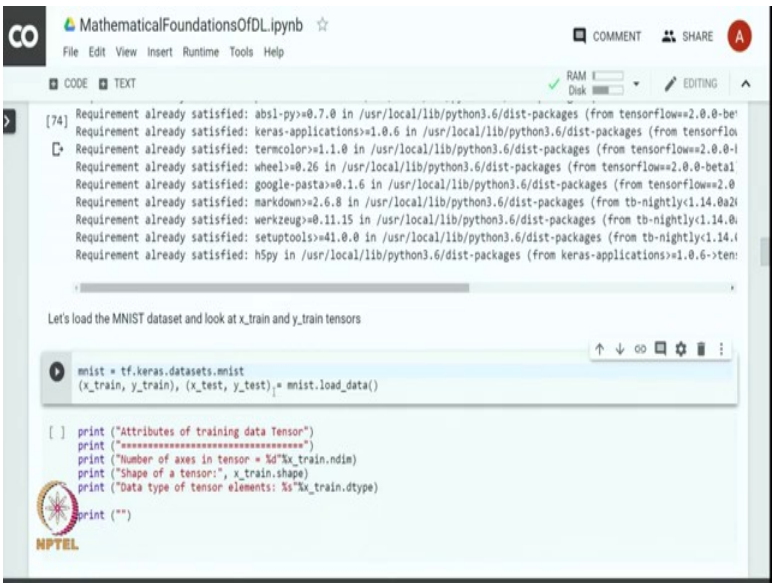
Let us look at its attributes. First, it has got the shape (3 , 3 , 3). There are 3 entries or there are 3 matrices and each matrix is of shape (3 , 3). So, that is how we have (3 , 3 , 3). The number of dimensions are obviously 3 because this is a 3D tensor and each element holds 64 bit integer as its data type. Now, that we have seen some example of the tensor, let us look at the tensors from the MNIST data set for which we built a model.

(Refer Slide Time: 06:15)



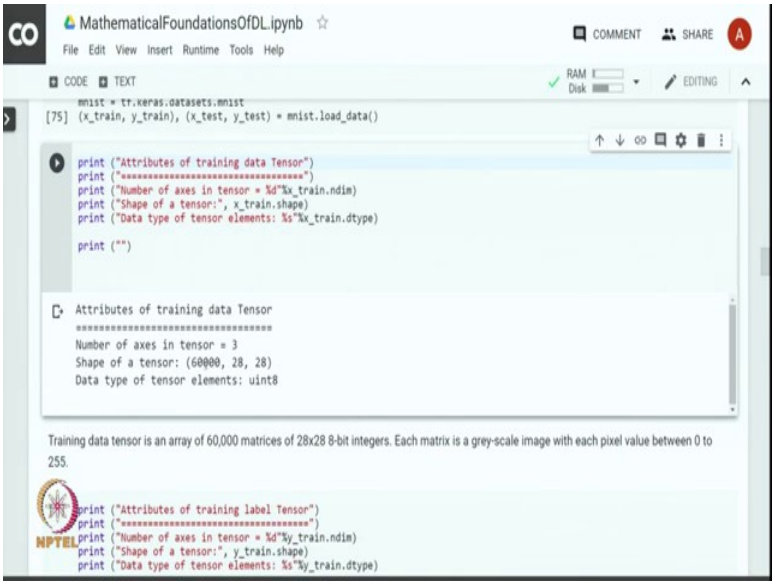
In order to load the MNIST data set let us first install tensor flow 2.0.

(Refer Slide Time: 06:22)



Let us load the MNIST data set. Load data function load is a MNIST data set as two tuples; one corresponding to training and the second corresponding to the test; *x_train* and *y_train* are features and labels of the training set. Whereas, *x_test* and *y_test* are features and labels of the test examples.

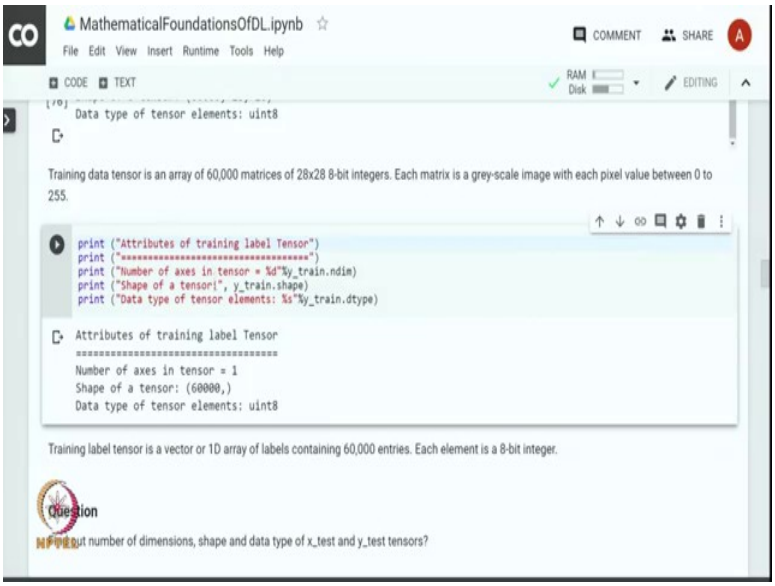
(Refer Slide Time: 06:53)



Let us look at the attributes of training data tensor, *x_train* is a training data tensor, so we look at the number of axis or number of dimensions of *x_train* tensor. You also look at its shape and the data type of each of the elements stored in that. You can see that the

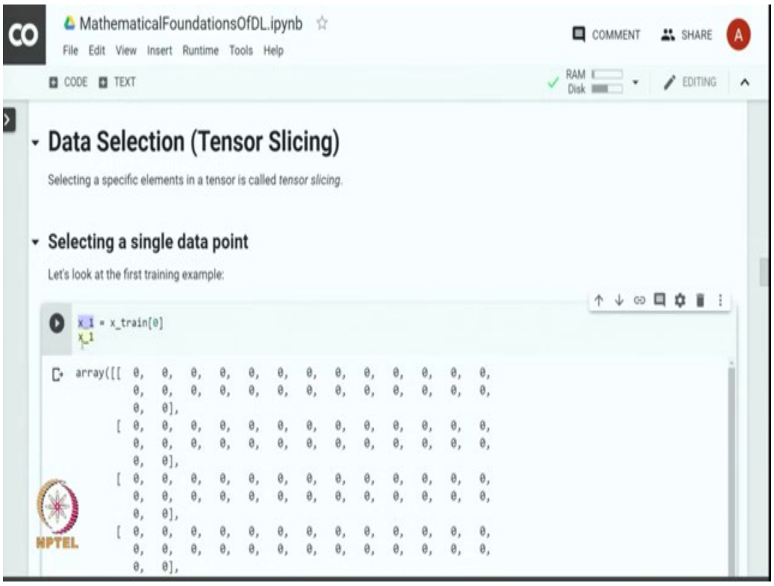
training tensor has a three-dimensions, it shape is (60000 , 28 , 28). As you might recall that this is an array of 60,000 matrices each of 28 by 28 size and each element in the tensor holds 8 bit integer between 0 to 255.

(Refer Slide Time: 07:45)



Let us look at the attributes of training label tensor. So, training label tensor is a vector of 1D array of labels containing 60,000 entries and each element is a 8 bit integer. Each element of the tensor represents the class of the image. There is a question for you now can you find out the number of dimensions, shape and data type of *x_test* and *y_test* which are tensors corresponding to the test data?

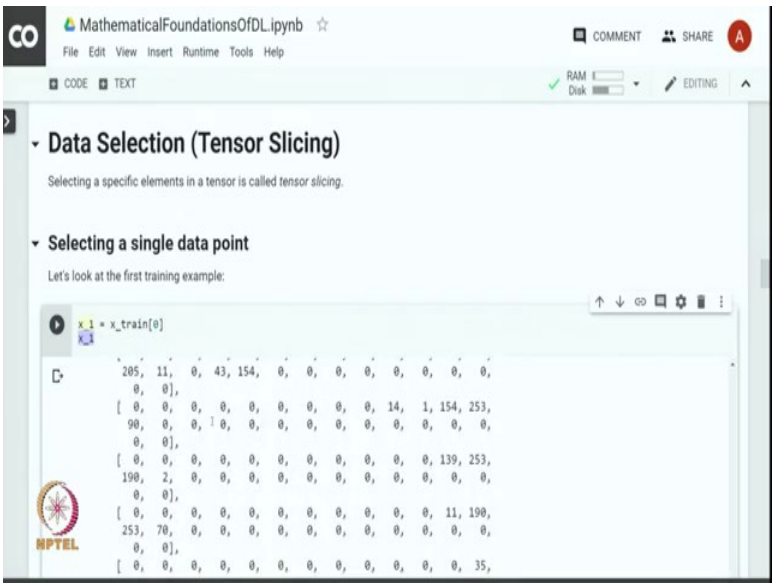
(Refer Slide Time: 08:22)



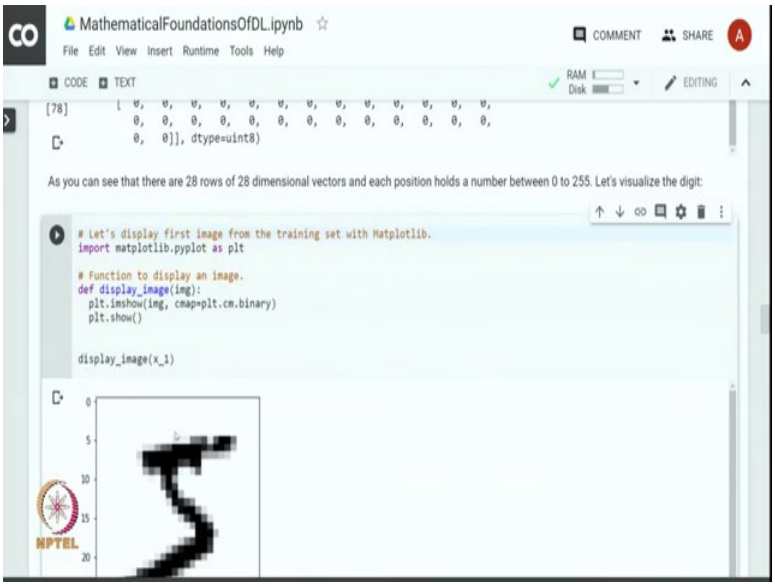
Now, that we have understood basics of tensors, let us look at how to select a specific element in a tensor. The process of selecting a specific element in a tensor is called tensor slicing. Let us select a single data point from a tensor. So, we can let us select the first data point from x_train tensor to be specific.

So, that can be selected simply by mentioning the name of the tensor and putting 0 as the index. So, this particular statement returns the first image that is stored in the *x_train* tensor. We know that this particular image is represented in 28 cross 28 matrix, each containing a value between 0 to 255. So, you can see that lots of elements are 0, and there are some elements which have got non-zero values up to 255.

(Refer Slide Time: 09:20)

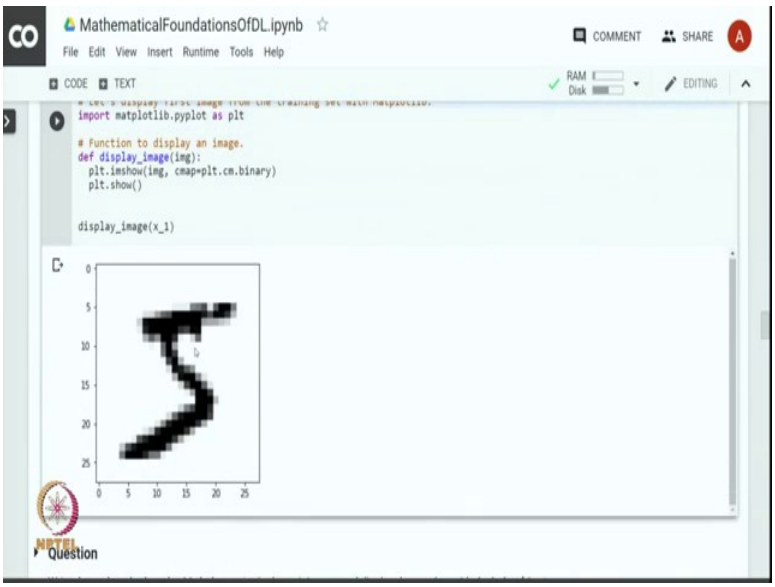


(Refer Slide Time: 09:26)



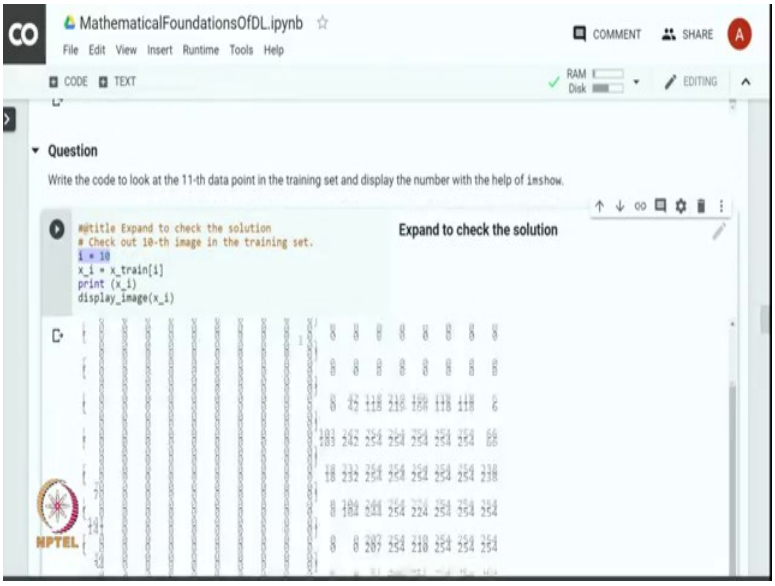
So, plt,imshow() function is used to display an image. So, will pass the tensor to display image that we have defined here that converts the tensor into an image. Let us look at the image corresponding to this.

(Refer Slide Time: 09:47)



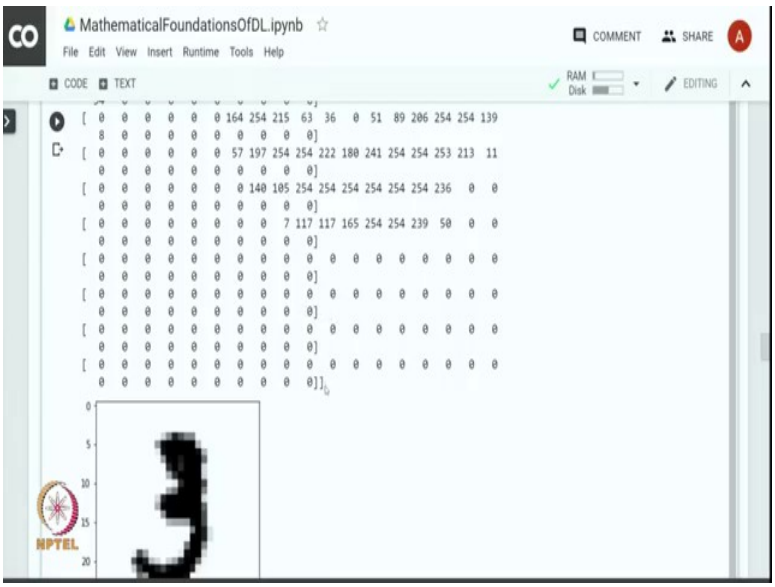
So, you can see that the first example in the training tensor is an image corresponding to handwritten digit 5. Now, that you have understood how to access an individual element from a tensor that. Here is a question for you, can you write a code to look at 11 data point in the training set and display the number with the help of imshow() command?

(Refer Slide Time: 10:17)



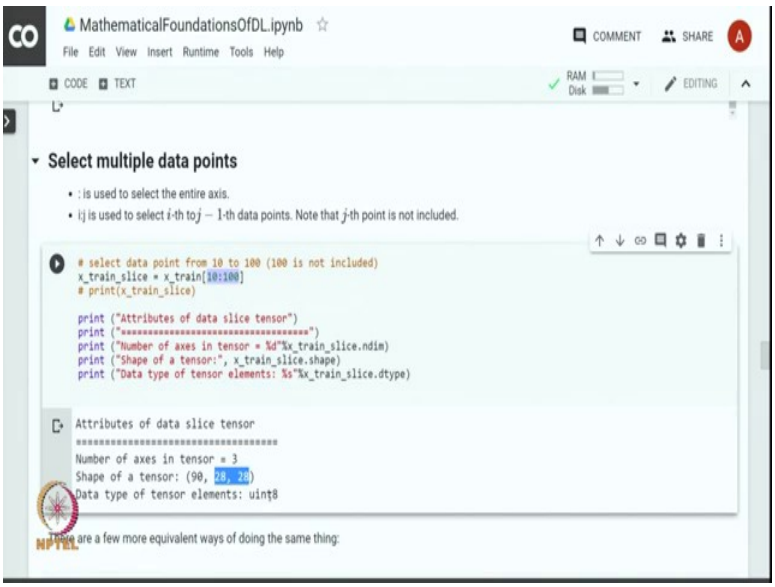
So, we can simply access 11th element by specifying index equal to 10 and we can; so, because the index are set 0, i = 10 represent 11th image in the training set and it will give us the tensor representation as well as image representation.

(Refer Slide Time: 10:36)



Now, that we have understood how to select a single data point, we are often required to select multiple data point from the tensor. Let us see how to do that.

(Refer Slide Time: 10:42)

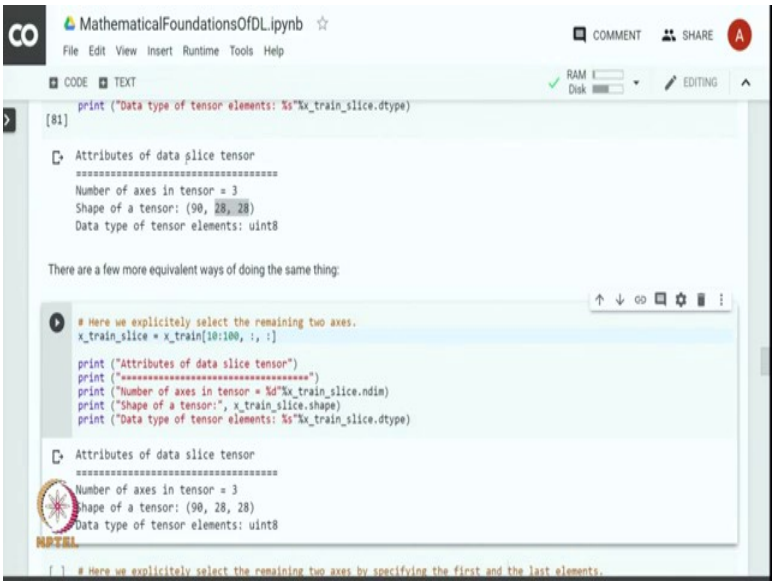


So, we use a colon operator to select the entire axis or alternatively we can specify the start and end of the tensor. If we specify i : j, we can select the tensor or we can slice the tensor from i^{th} position to j - 1 position. Note that the j^{th} position is not included in the slice.

Let us see a complete example of selecting multiple data points. So let us try to select data points from 10 to 100. Note that 100 will not be included in this. So, we can simply put 10:100 as the selection criteria and slice the tensor. Let us look at attribute of this tensor slice.

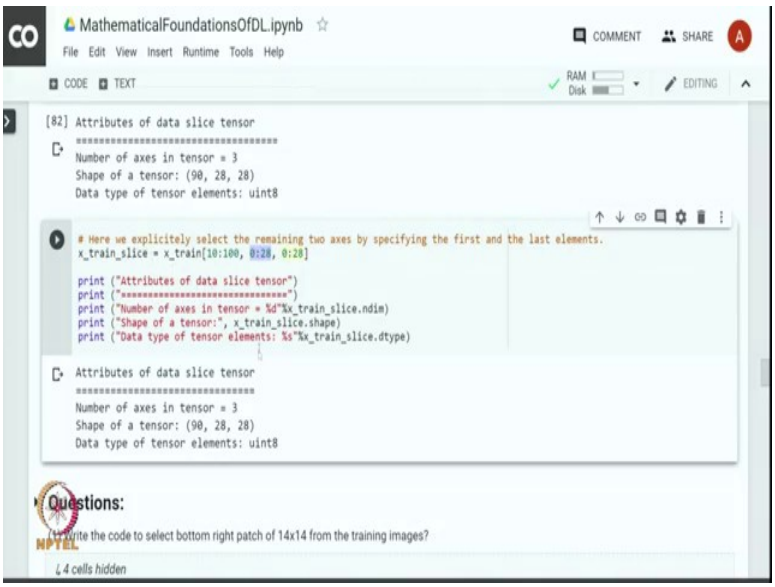
So, we can see that we get the same tensor, we get the tensor of the same dimension as the original tensor, so *x_train* was a three-dimensional tensor. We got the slice also as a three-dimensional tensor. The shape of the tensor is different from its original tensors. Since, here we are only selecting 90 elements we can see that we have 90 samples each having 28 X 28 matrix in them.

(Refer Slide Time: 12:25)



And each of the element in the tensor is a 8 bit integer. There are a few more equivalent ways of doing the same thing. Since, we have a 3D tensor we can also specify exquisitely the remaining two axis using a colon command. It will result in the same selection. You can compare that the data slice or the tensor slice that we obtain from the earlier method and this particular method has exactly the same result.

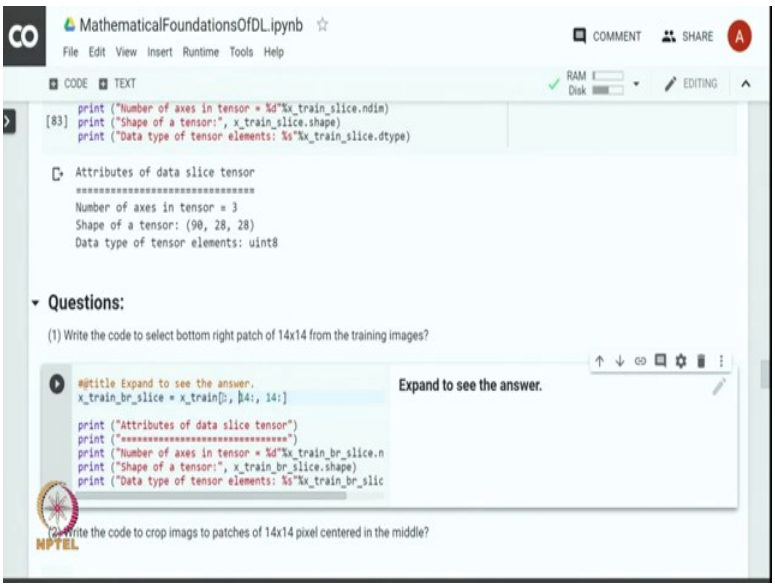
(Refer Slide Time: 12:52)



Let us look at one more alternative to the selection. Now, instead of selecting this axis using the colon command, we will specify the start and end point. We know that both this axis have 28 elements each, so we specify or we ask to slice the axis completely or get all the elements in the axis using the start and end which is matching the length of the axis.

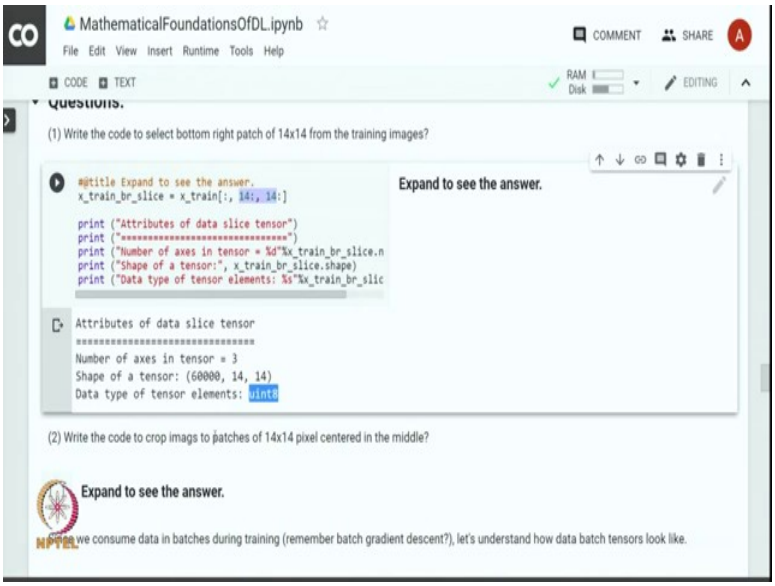
So, we can once you run the code you will realize that it also results in the same thing where we get a 3D tensor having 90 examples containing 28 cross 28 matrix and each tensor element is a 8 bit integer. So, this gives you multiple ways of achieving the same thing. Now, here is a question for you. Can you write the code to select the bottom right patch of 14 by 14 from the training image?

(Refer Slide Time: 14:02)



So, here is the answer. So, we can simply get a bottom right part by specifying the slice as follows. You want to select the first axis as it is, but we want only 14 by 14 patch from bottom right, so we specify 14 colon and 14 colon. So, everything after the 15th element up to the last will be selected in both the cases. So, let us look at the attributes of the data slice tensor.

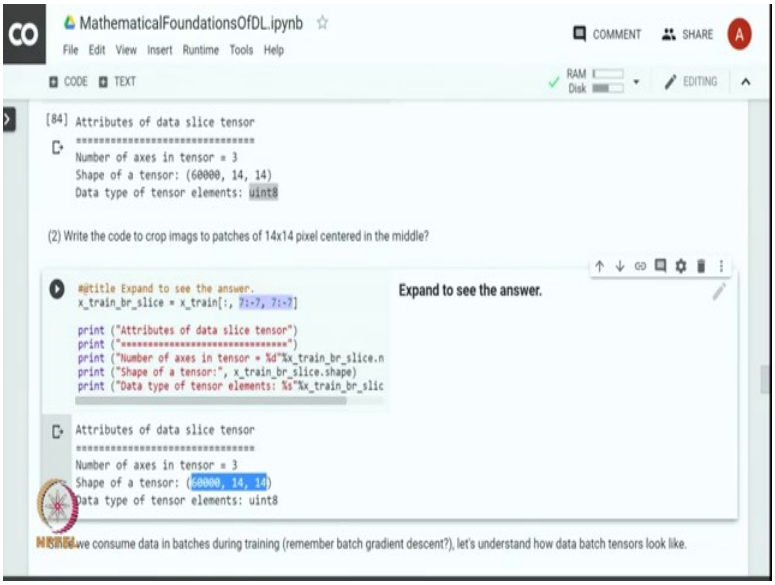
(Refer Slide Time: 14:38)



So, you can see that we have a 3D tensor having 60,000 images with 14 x 14 patch from the bottom right and each of this tensor contain integers of 8 bit integers. Can you write

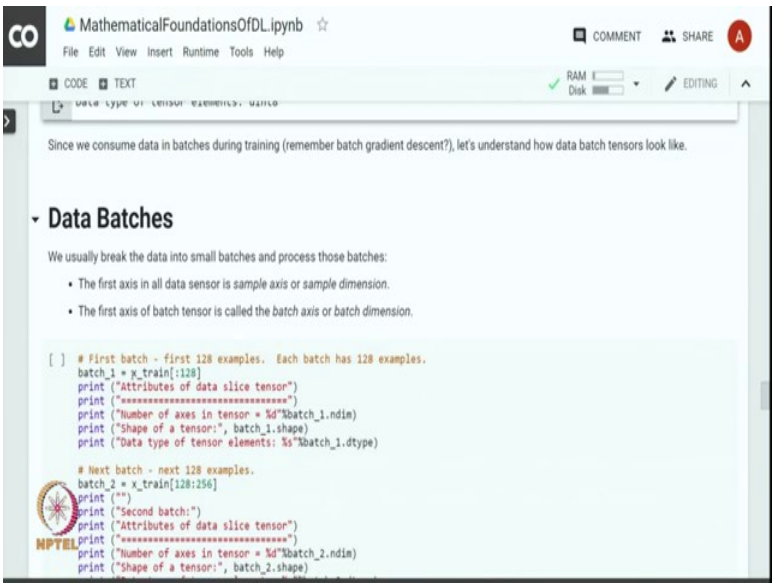
the code to crop images to patches of 14 cross 14 pixels centered in the middle? Let us see how to do that.

(Refer Slide Time: 15:07)



Since, we want the patch in the middle we go from 7th element up to all the elements except last 7 elements in both the axis here and let us see what it results in. We essentially get a 14 cross 14 patch for all the images which is centered in the middle.

(Refer Slide Time: 15:31)

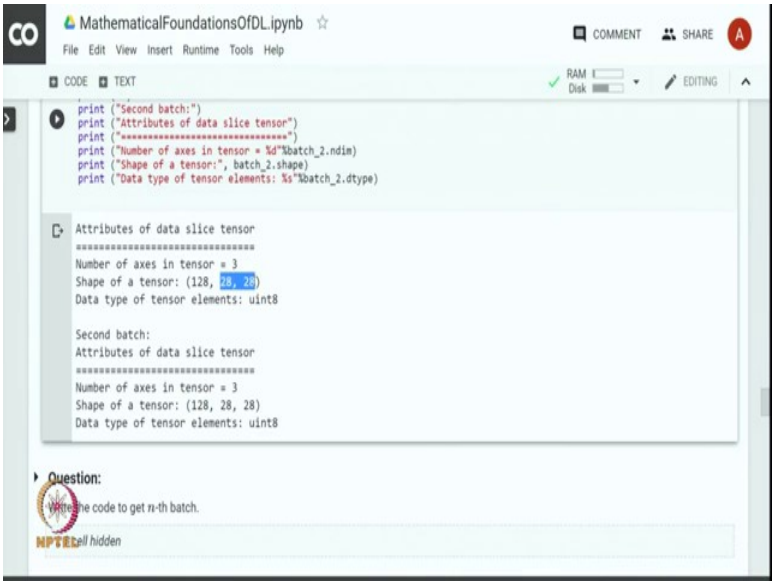


Since, we consume data in batches during training let us understand how data batch is how data batch tensors look like. We usually break the data into small batches and

process those batches. If we have the complete data, the first axis in all data tensor is a sample axis or sample dimension. The first axis on the other hand for batch tensor is called the batch axis or batch dimension.

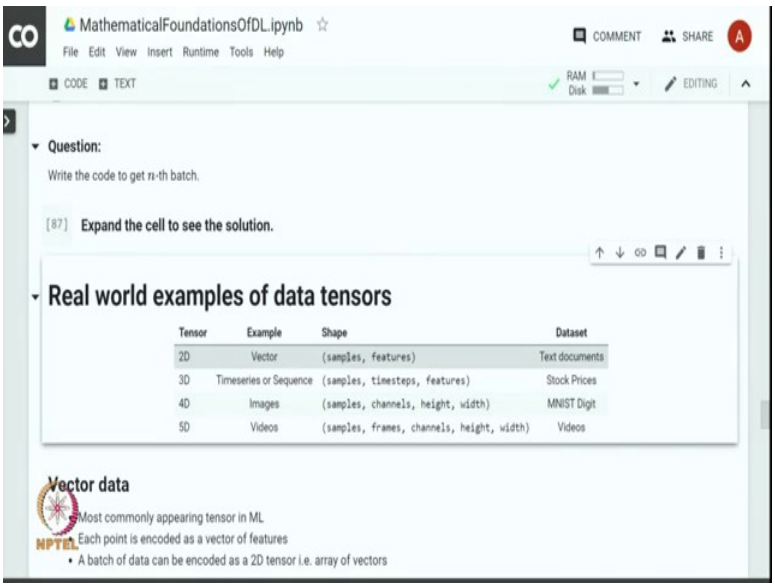
Let us look at the first batch of 128 examples here you have a batch of size 128. So, we simply specify 128 batch with this particular slicing criteria and then we select the next 128 example with this particular slicing criteria where we specify the start and the end position.

(Refer Slide Time: 16:20)



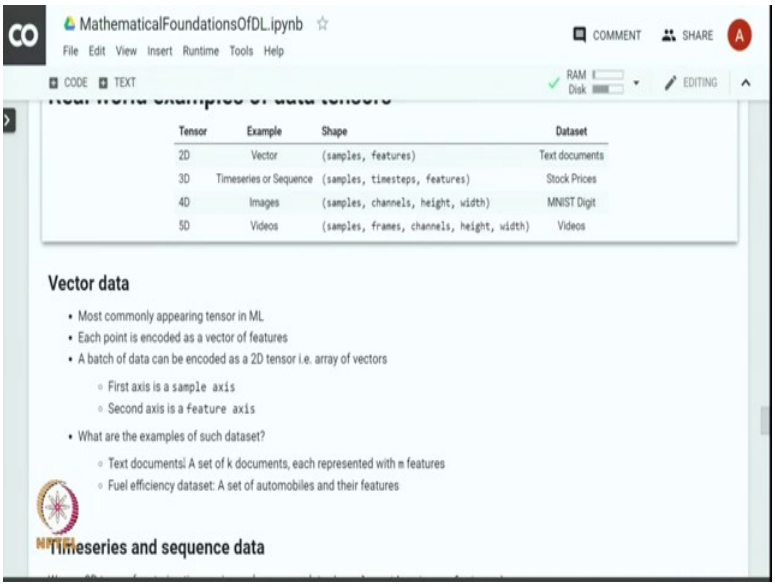
Let us look at the attribute of this data slices for both the batches. You can see that both the batches have 3D tensors having exactly the same shape, each batches contain 128 examples, each example is represented in the metric form of 28 x 28. And each data type of tensor is an 8 bit integer. Here is a question for you. Can you write the code to get n^{th} batch?

(Refer Slide Time: 17:14)



Let us look at some of the real life examples of data tensors. So, in real life while building machine learning models, we often come across tensors of dimensions between 2 and 5. 2D tensors are most commonly appearing tensors in machine learning. 2D tensors have shaped (samples, features). So we have essentially each sample represented with bunch of features.

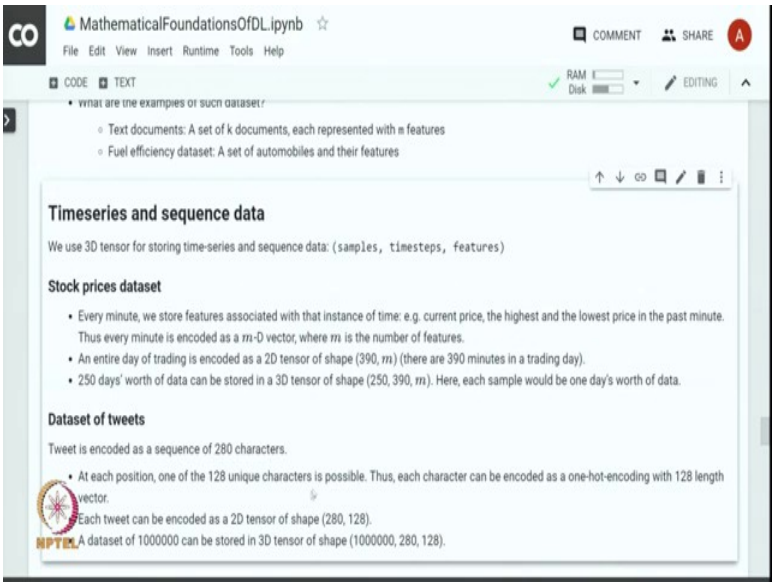
(Refer Slide Time: 17:44)



Let us take a couple of examples. One is representing text document. Let us say we have a set of k documents each represented with m features. So, this particular set of text

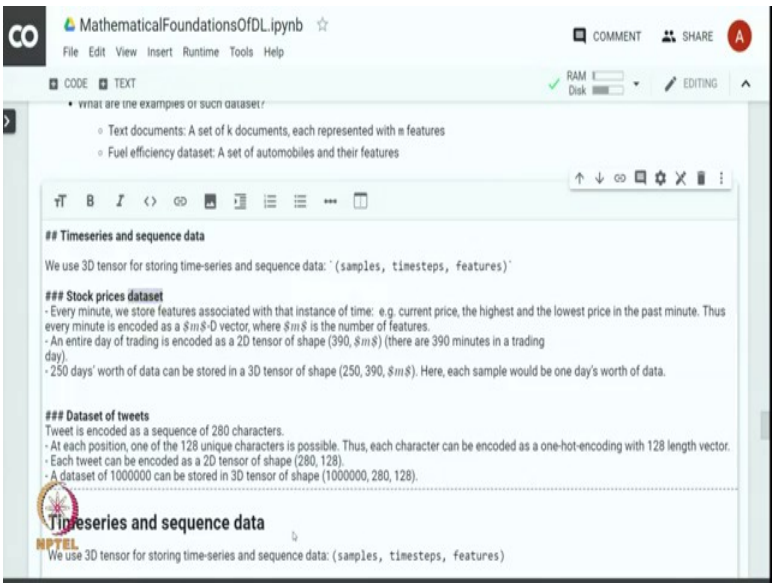
document can be represented by k , m tensor because there are k samples in this data set and each sample is represented with m features. The other example could be a fuel efficiency data set that contains set of automobiles and their feature. So, the defining feature of the 2D tensor is that there are samples and each sample has a list of features. Then we have 3D tensors that we often encounter in time series data set or in sequence data sets.

(Refer Slide Time: 18:35)



Let us take an example of a time series data set, where you have to store the talk stock prices. The 3D tensors have 3 axis, the first axis is the sample axis, the second axis is the time step axis and the third axis is the feature axis.

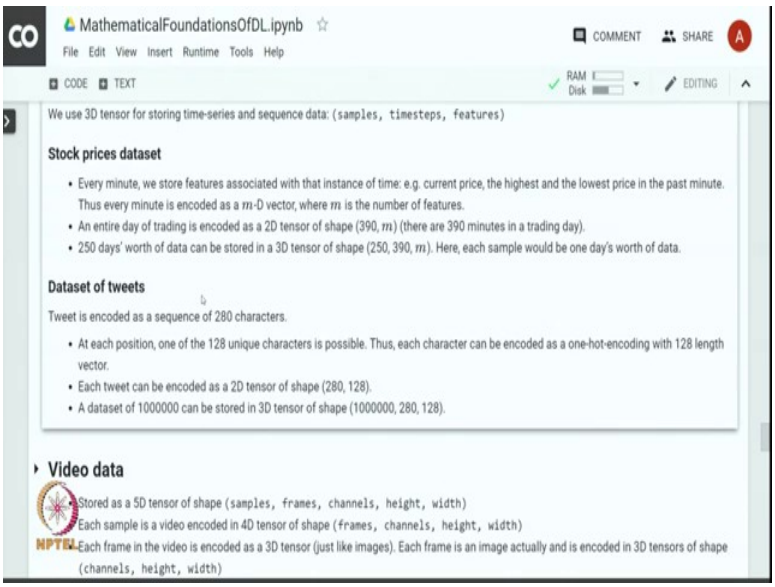
(Refer Slide Time: 18:47)



What happens is in a time series data we have a sample across multiple time steps, specifically for each time step we have a list of features. Let us say in stock price data set, we store the features associated with the stock at every minute. The features could be current price, the highest and lowest price in the past minute or some such kind of features.

Every minute is encoded as a mD vector where m is the number of features. An entire day of trading is encoded as a 2D tensor of shape (390, m), where there are 390 minutes in a trading day. In order to store data for 250 days it can be stored in a 3D tensor of shape, 250 corresponding to 250 days and for each day there are 390 data points each containing m features.

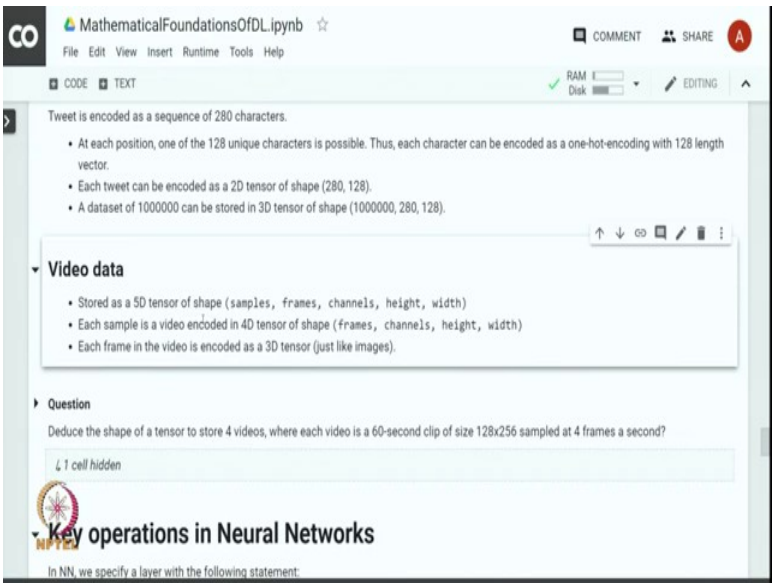
(Refer Slide Time: 20:01)



Let us see how we can use 3D tensor to store data set of tweets. Here we encode each tweet as a sequence of 280 characters, at each position one of the 128 unique characters are possible. Thus each character can be encoded as a one hot encoding with 128 length vector. Each tweet can be encoded as a 2D tensor of shape (280 , 128). The data set of one million tweets can be stored in a 3D tensor having shape (1 million , 280 , 128).

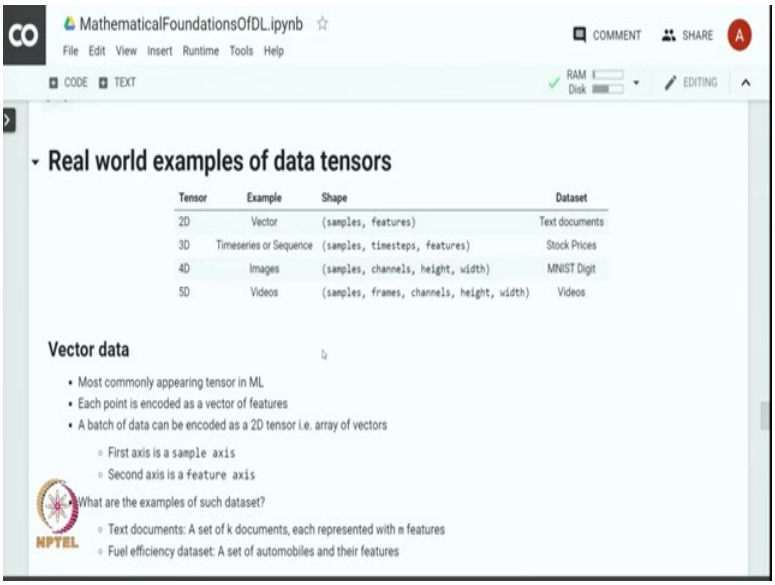
In each of the 1 million tweets we have 280 characters and in each of the 280 character is represented as a 128 dimensional feature vector, and this 128 are all the possible characters that is why we are using one hot encoding to represent every position in the tweet.

(Refer Slide Time: 21:09)



Let us look at how do we store the video data.

(Refer Slide Time: 21:13)



Images are stored as 4D tensors where the first access is samples where which is dedicated to every image, so we have sample and for every sample if you have a color image we have channels and height and width. There is a channel corresponding to red, green and blue in the colored image. In grey scale image there is a single channel and for every channel we have height and width, that is defined which is the resolution of the image, and at every cell we store the value of the illumination of the pixel.

Video data is an extension of the image data, where in addition to the image data which is channels height and width, we store the image information at a frame level. And each of the video has multiple frames, and in each frame we store the image information. Each frame in the video is encoded as a 3D tensor. Now, there is a question for you. Can you reduce the shape of a tensor to store 4 videos where each video is a 60 second clip of size 128 by 256 sampled at 4 frames a second?