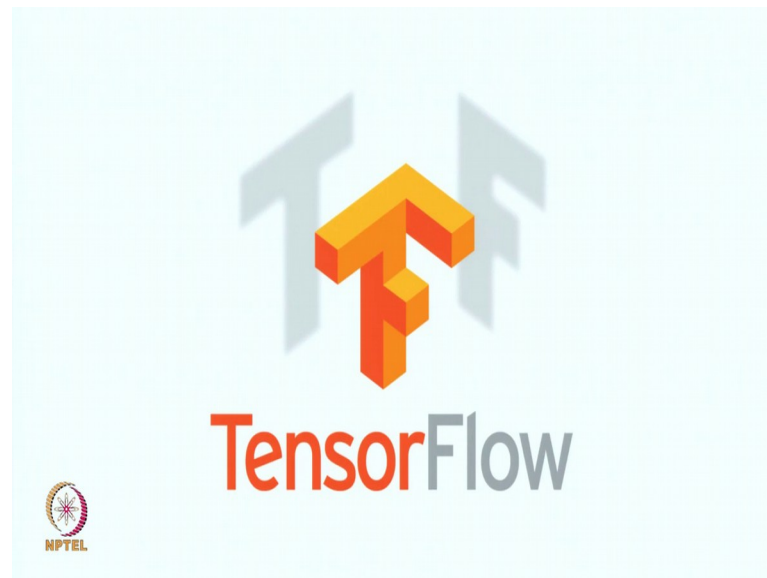


Practical Machine Learning with TensorFlow
Dr. Ashish Tendulkar Google
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

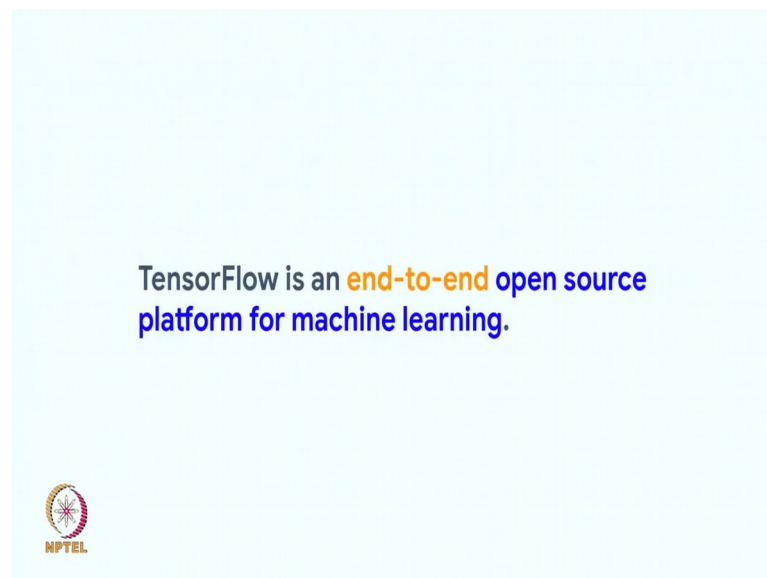
Lecture – 01
Overview of TensorFlow

[FL] Let us try to understand, what TensorFlow exactly is?

(Refer Slide Time: 00:46)

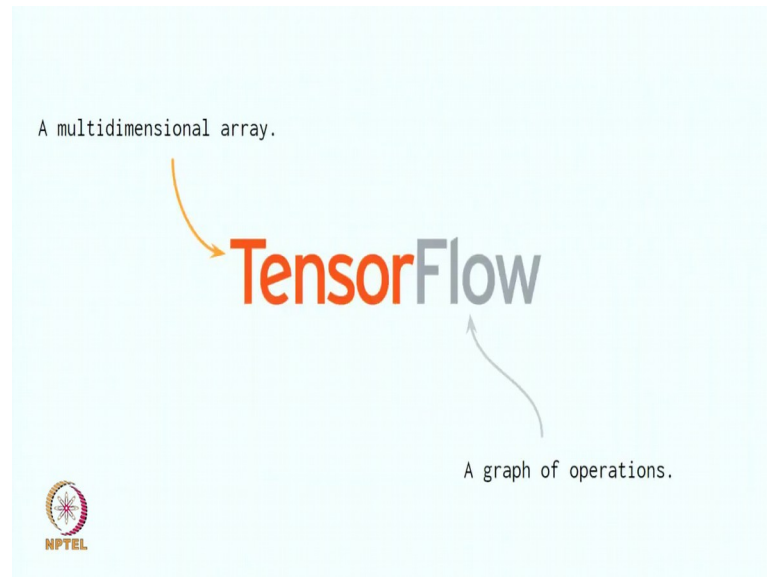


(Refer Slide Time: 00:48)



This is a logo of Tensor Flow, which is an end-to-end open-source platform for machine learning.

(Refer Slide Time: 00:55)



Tensorflow has a meaning, TensorFlow is made up of two words: tensor and flow. Tensor is the multidimensional array and flow is a graph of operations. Internally, TensorFlow implements machine learning algorithms as a graph of operations on the multidimensional array.

(Refer Slide Time: 01:18)

A brief history

- Developed by Google Brain
- Released under Apache 2.0 license in Nov 2015
- Current stable version: API 1 version 1.14
- Popular GitHub repo with 129k+ stars



This course covers concepts from **TensorFlow API version 2.0**, which is the newest version of TF .




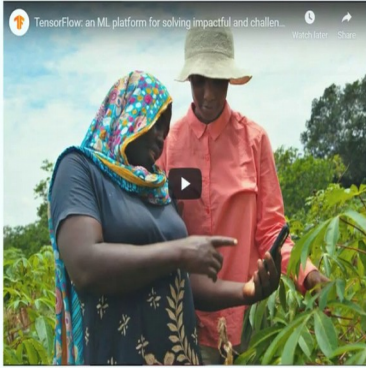
Tensorflow was developed by Google Brain and it was released under Apache 2.0 license in November 2015. The current stable version of TensorFlow is 1.14 and it is a popular GitHub repo with 100 and 29k plus stars. TensorFlow is a vibrant active community of developers with more than 1800 developers actively contributing to the code base. This course covers concepts from TensorFlow API version 2.0, which is the newest version of TensorFlow.

(Refer Slide Time: 01:55)

Why TensorFlow?

- Easy to build and deploy ML models
- Enables researchers to build state of the art ML models with Keras functional API and model subclassing API
- Supports ML production anywhere - web, servers or edge devices irrespective of platform or language used for training and building.





Why do we really care about TensorFlow? TensorFlow provides easy to build and deploy machine learning models for a newcomer in machine learning. If you are an expert in machine learning or a machine learning researcher, TensorFlow enables you to build a state of machine learning models with Keras functional API and model subclassing APIs.

Another important thing about TensorFlow is that it supports the production of machine learning models, anywhere from CPUs, GPUs to edge devices as well as web servers. TensorFlow API is available for Python, for Java and for Go programming languages. TensorFlow has a very flexible architecture. It enables easy deployment across different hardware platforms like CPUs, TPUs and GPUs, and computing devices like desktops, servers, mobile devices, and edge devices.



Another important thing about TensorFlow is that it supports production of machine learning models, anywhere from CPUs, GPUs, to edge devices as well as web servers.

TensorFlow API is available for Python, for Java and for Go programming languages. TensorFlow has a very flexible architecture; it enables easy deployment across different hardware platforms like CPUs, TPUs and GPUs, and computing devices like desktops, servers, mobile devices, and edge devices.

(Refer Slide Time: 02:54)

TF has flexible architecture

- Enables easy deployment across hardware platforms like
 - CPU
 - GPU
 - TPUs
- And computing devices like
 - Desktops
 - Servers
 - Mobile devices
 - Edge devices



(Refer Slide Time: 03:00)

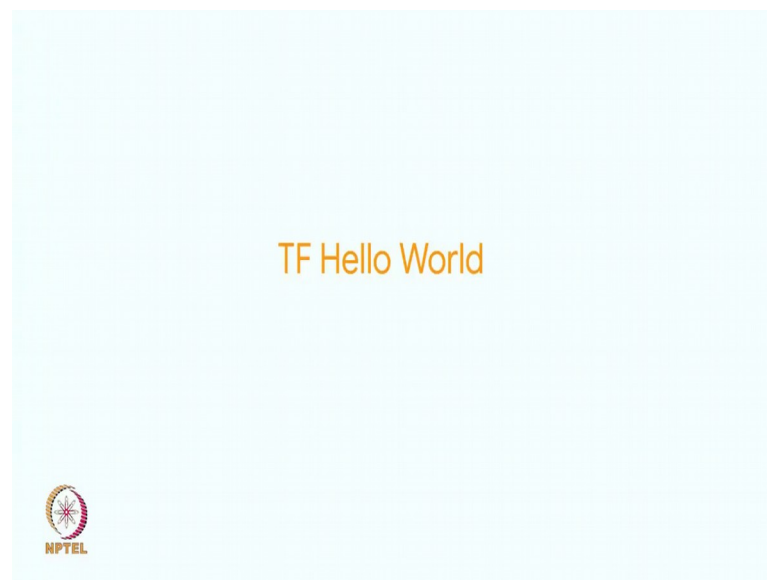
TensorFlow everywhere



Tensorflow is being used by lots of companies around a world; these companies operate in different domains and are using TensorFlow to carry out build machine learning models in different domains. For example, Google is using TensorFlow to better its products various products like Gmail or doc, Airbnb, for example, is using TensorFlow to classify images and detect objects in their set of photographs.

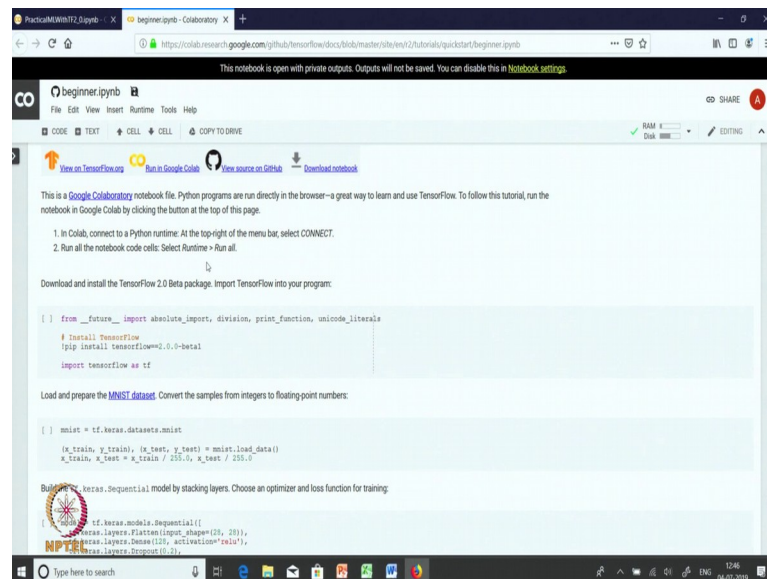
AIRBUS is using TensorFlow to detect interesting objects from the satellite imagery and make it available to its customer. TensorFlow is also used for a lot of social good applications as well as in the financial domain like PayPal is using TensorFlow for detecting fraudulent transactions, Twitter is using TensorFlow to run tweets. So, you can see that TensorFlow is a versatile product and is being used for developing and deploying machine learning models by companies across different domains. You can check out some of these case studies on [tensorflow.org](https://www.tensorflow.org) website.

(Refer Slide Time: 04:18)



Let us try to build our first machine learning model with TensorFlow, we call it as TensorFlow Hello World. We will train our first machine learning model with TensorFlow API.

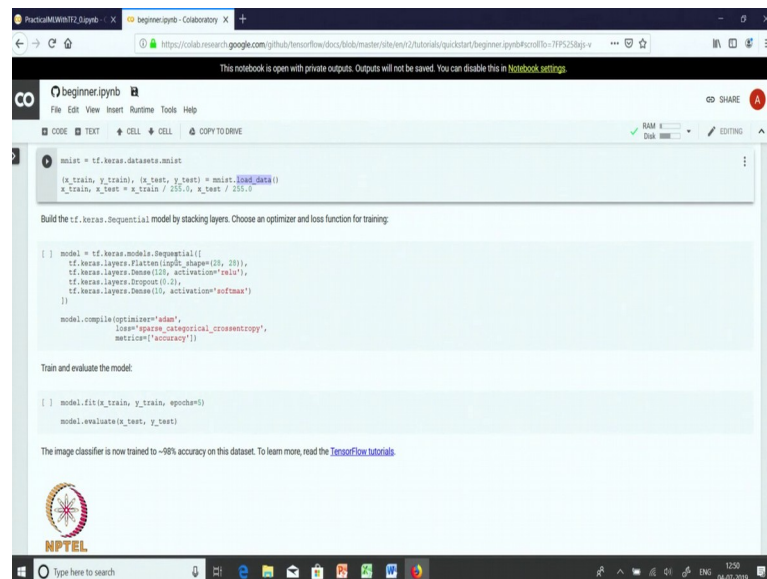
(Refer Slide Time: 04:30)



This machine learning module helps us recognize handwritten digits. We will train the machine learning model with famous MNIST dataset. MNIST dataset contains grace skilled images of handwritten digits. There are 60,000 images in the training set and 10,000 images in the test set of MNIST dataset. Each image is of size 28 x 28 pixel and each image is tagged with a label that is the actual number it represents.

So, this is a Google Colab environment, it allows us to run Python programs directly in the browser. Here we code our model in the Colab environment, it has got text and code cell. This is an example of a text cell and this is an example of a code cell. In the text-cell, we have written comments or some text that will help us understand what is going on in the Colab. In the code cell, we write essentially the Python code, we will first go through the Colab cell by cell and then run it. In the first code cell, we import the required packages and install TensorFlow 2.0. After installing TensorFlow 2.0 we import the TensorFlow package.

(Refer Slide Time: 05:59)



```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

Build the tf.keras.Sequential model by stacking layers. Choose an optimizer and loss function for training:

[ ] model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

Train and evaluate the model:

[ ] model.fit(x_train, y_train, epochs=10)
model.evaluate(x_test, y_test)

The image classifier is now trained to ~98% accuracy on this dataset. To learn more, read the TensorFlow tutorials
```

Next, we load the MNIST dataset, MNIST dataset is available in the TensorFlow dataset package. So, we can directly load that with this particular command and so, essentially MNIST dataset is defined as `tf.keras.datasets.mnist` and we load MNIST dataset with the load data command. The load data command essentially gives us MNIST dataset in two tuples, the first tuple contains the training data and the second tuple contains a test data.

So, we have the training features in `x_train` matrix, `y_train` vector contains the label of the training examples, `x_test` metric contains the features and `y_test` vector contains the label corresponding to each of the examples. The i^{th} entry in `x_train` metric represents features for the i^{th} example and i^{th} entry in the `y_train` gives us the corresponding label. After loading the dataset we normalize the dataset by dividing each pixel value by 255. The normalization helps us achieve faster conversions during training.

Now, that we have loaded the dataset and pre-process state, the next task is to build a model. We will build `tf.keras.models.sequential()` model by stacking the layers. Next, we use loss function and an optimizer for the model, we select the `sparse_categorical_crossentropy` loss as the loss for this particular model and we choose Adam as an optimizer for this particular model. You can note that we first flattener input, so our original input is 28 by 28 pixels.

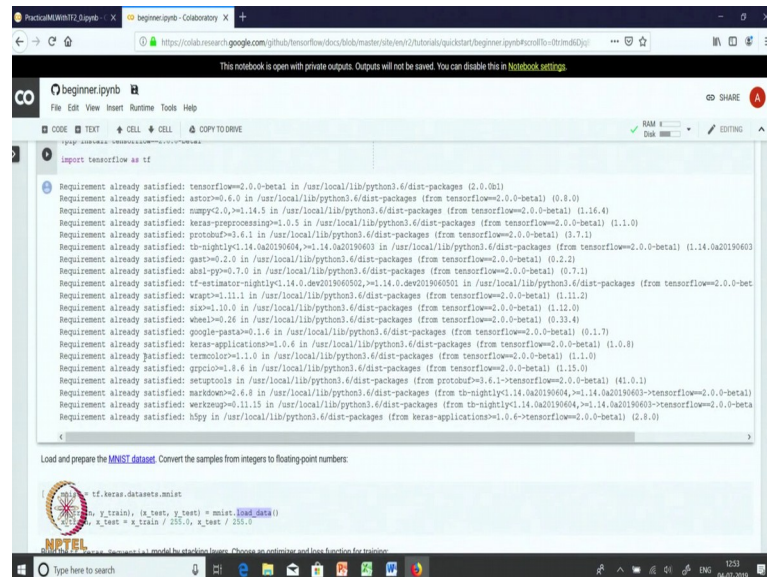
So, we will flatten it to make it into a vector of size 784 and this particular input is fed into the dense layer which has got 128 units. We use Relu activation in this particular

dense layer and this is the only hidden layer that we use in this particular neural network. In addition, we use dropout regularization with a dropout rate of 0.2, the output layer contains 10 units one corresponding to each of the digits between 0 to 9 and we use softmax as an activation function for the output layer.

Now that we have compiled our model the next step is to train the model. We use `model.fit()` function for training the model. The `fit()` function takes the training features and training labels as an argument along with the number of epochs, just to remind you an epoch is one full iteration of the training set. After training the model we evaluate the model on the test set which has the test features and the test labels. Notice that the model was trained on the training data and its performance was evaluated on the test data. This ensures that we have a fair estimate of the model performance on unseen data. You can observe that we have specified our model its training and evaluation all in less than 10 lines of code with `tf` API. This is the ease of use that makes TensorFlow an API of choice for machine learning developers.

Now that we have written the code for model specification, training and evaluation let us execute the code in the notebook to see what kind of performance we achieve on this model. So, in order to run the code, we have to first connect to the Colab environment which we have already connected here. After connecting to Colab environment we can execute this notebook cell by cell. This run button over here, if you press this the code in this particular code cell will get executed, alternatively, we can press the control enter button to execute the cell as a keyboard short cut.

(Refer Slide Time: 10:50)



The screenshot shows a Jupyter Notebook interface with the title 'beginner.ipynb'. The code cell contains the following text:

```
import tensorflow as tf
```

Below the code, a list of requirements is displayed, indicating that various packages are already satisfied. The requirements include:

- Requirement already satisfied: tensorflow==2.0.0-beta in /usr/local/lib/python3.6/dist-packages (2.0.0b1)
- Requirement already satisfied: astor>=0.6.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta) (0.8.0)
- Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta) (1.16.4)
- Requirement already satisfied: keras-preprocessing>=1.0.5 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta) (1.1.0)
- Requirement already satisfied: protobuf>=3.6.1 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta) (3.7.1)
- Requirement already satisfied: tb-nightly<1.14.0a20190604,>=1.14.0a20190603 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta) (1.14.0a20190603)
- Requirement already satisfied: gast>=0.2.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta) (0.2.2)
- Requirement already satisfied: absl-py>=0.7.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta) (0.7.1)
- Requirement already satisfied: tf-estimator-nightly<1.14.0.dev2019060502,>=1.14.0.dev2019060501 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta) (1.14.0.dev2019060501)
- Requirement already satisfied: wrapt>=1.11.1 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta) (1.11.2)
- Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta) (1.12.0)
- Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta) (0.33.4)
- Requirement already satisfied: google-pasta>=0.1.6 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta) (0.1.7)
- Requirement already satisfied: keras-applications>=1.0.6 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta) (1.0.8)
- Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta) (1.1.0)
- Requirement already satisfied: grpcio>=1.8.4 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta) (1.15.0)
- Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from protobuf>=3.6.1->tensorflow==2.0.0-beta) (41.5.1)
- Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/dist-packages (from tb-nightly<1.14.0a20190604,>=1.14.0a20190603->tensorflow==2.0.0-beta)
- Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.6/dist-packages (from tb-nightly<1.14.0a20190604,>=1.14.0a20190603->tensorflow==2.0.0-beta)
- Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages (from keras-applications>=1.0.6->tensorflow==2.0.0-beta) (2.8.0)

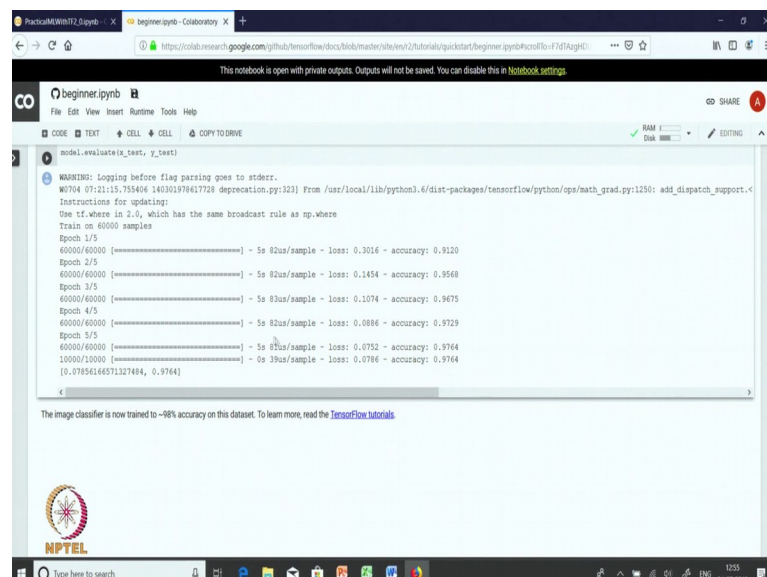
Below the requirements, a message states: 'Load and prepare the MNIST dataset. Convert the samples from integers to floating-point numbers.'

The code cell then contains the following text:

```
x_train, y_train, x_test, y_test = mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0
```

Let us download the TensorFlow 2.0 beta version which we have already download in this case. If this is not downloaded it will take some time to download the version from the internet and hence this particular code cell might take a bit longer for you. Next, we load the MNIST data and normalize it, next we compile our model.

(Refer Slide Time: 11:16)



The screenshot shows a Jupyter Notebook interface with the title 'beginner.ipynb'. The code cell contains the following text:

```
model.evaluate(x_test, y_test)
```

Below the code, a warning message is displayed: 'WARNING: Logging before flag parsing goes to stderr. W0704 07:12:15.705406 160301978437728 deprecation.py:323] From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_grad.py:1250: add_dispatch_support.< Instructions for updating: Use tf.where in 2.0, which has the same broadcast rule as np.where'.

The output of the code cell shows the training progress of the model on the MNIST dataset. The output is as follows:

```
Train on 60000 samples
Epoch 1/5
60000/60000 [=====] - 5s 82us/sample - loss: 0.3016 - accuracy: 0.9120
Epoch 2/5
60000/60000 [=====] - 5s 82us/sample - loss: 0.1454 - accuracy: 0.9568
Epoch 3/5
60000/60000 [=====] - 5s 83us/sample - loss: 0.1074 - accuracy: 0.9675
Epoch 4/5
60000/60000 [=====] - 5s 82us/sample - loss: 0.0886 - accuracy: 0.9729
Epoch 5/5
60000/60000 [=====] - 5s 87us/sample - loss: 0.0752 - accuracy: 0.9764
10000/10000 [=====] - 0s 39us/sample - loss: 0.0786 - accuracy: 0.9764
[0.07854166571327484, 0.9764]
```

Below the output, a message states: 'The image classifier is now trained to ~98% accuracy on this dataset. To learn more, read the [TensorFlow tutorials](#)'.

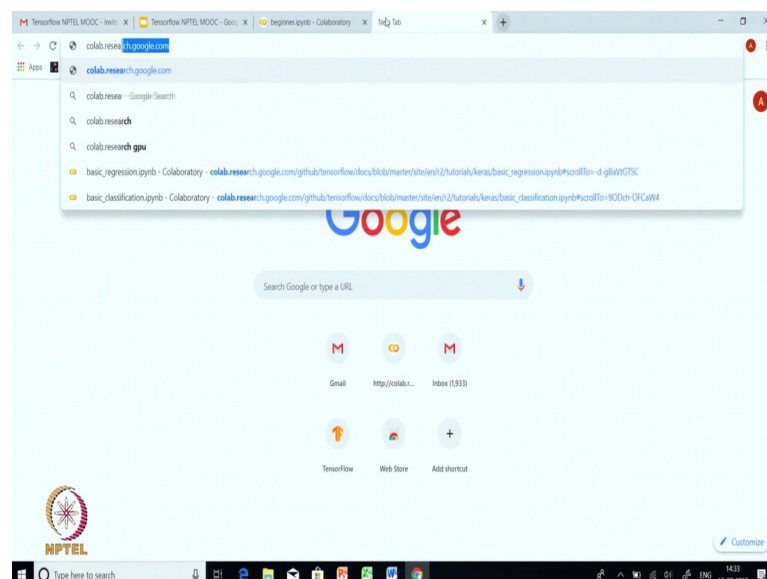
As we press the train button we can see that the model is getting trained and the progress of the model is shown with a progress bar and in each of the epoch, we see some statistics about loss and accuracy and amount of time the training takes per sample. So,

you can observe that the loss is going down with each epoch starting with 0.3 we got the loss down to 0.07 and the accuracy is going up, we started with an accuracy of 0.91 and accuracy has climbed up all the way up to 0.97.

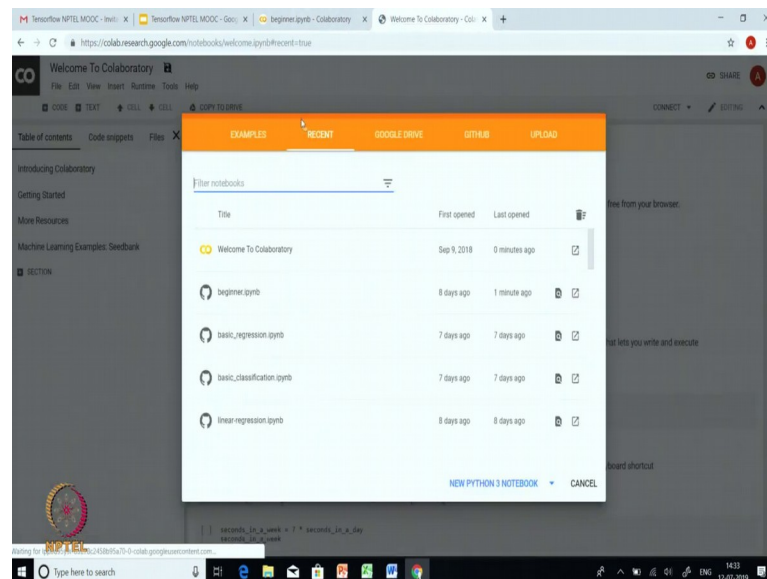
We started with 91 per cent accuracy and after 50 epoch we have an accuracy of 97 per cent. After training the model when we evaluated the model we achieve similar performance on the test data. One can see that the loss on test data is very close to the loss on the training data as well as the accuracy that we are getting on the test data is comparable to the accuracy that we are getting on the training data.

In this module, we build our first TensorFlow model for recognizing handwritten digits from MNIST dataset. Just now, we finished building our first machine learning model with TensorFlow, we called it a TensorFlow Hello World. You must have observed that we used Python in our browser. So, for most of the exercises in this course, we are going to use this tool called Google Colab. Colab is a Jupyter notebook that can be run from the browser. It uses cloud run time and can run in the browser without you needing to do a lot of complicated setup on your machine. Let us try to understand the basic features of Colab.

(Refer Slide Time: 13:23)

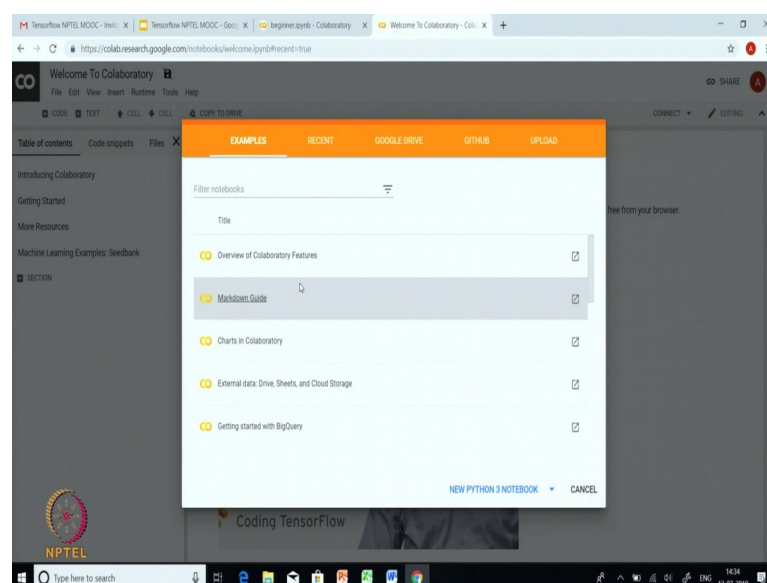


(Refer Slide Time: 13:30)



Colab can be accessed by colab.research.google.com URL and when you open Colab you can either load one of your existing notebooks that you can get it from the drive, you can load notebooks from the drive. You can also load notebooks from the GitHub. All that you have to do is you have to enter the GitHub URL of the notebook and the notebook will be open for you.

(Refer Slide Time: 13:50)

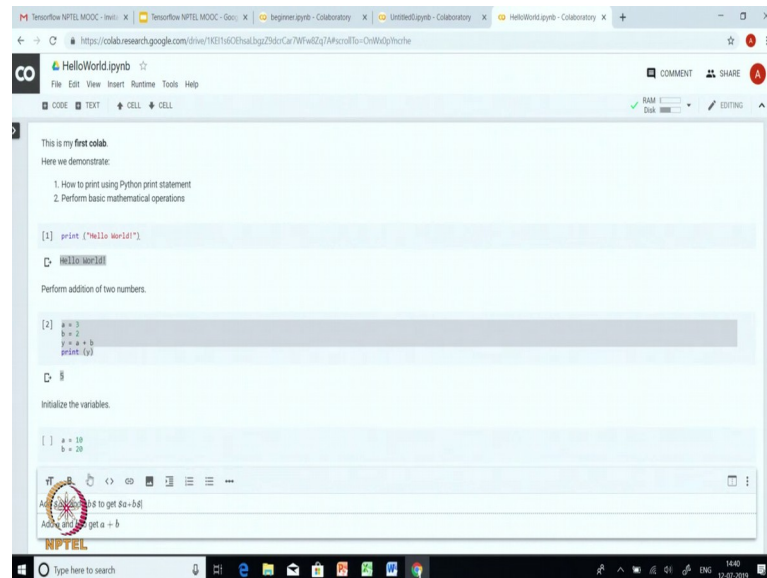


You can also upload your existing Jupyter notebooks through the upload tab and then the notebook will be available to you in the browser that you can run. Then we also have

several example Colabs that shows the functionality of Colab like reading external data from drive, sheet and cloud storage, Colab for getting data from Google Big Query or creating interesting input and output forms in the Colab.

So, let us try to start a new Python notebook.

(Refer Slide Time: 14:38)



This is how we start Colab we start a new Python notebook. We can save a copy in drive or we can save a copy in GitHub. Let us call this as “HelloWorld”. So, Colab file has an extension ipynb which is exactly same as the extension of Jupyter notebooks and in Colab we can seamlessly mix text and the code this mix it a very nice platform to write the documentation along with the code. So that it is easier for the reader to follow what is going on in the notebook. In addition to that, there are elements of collaboration inbuilt in the Colab, one can comment on the cell or one can share the Colab with their collaborators with the share button here.

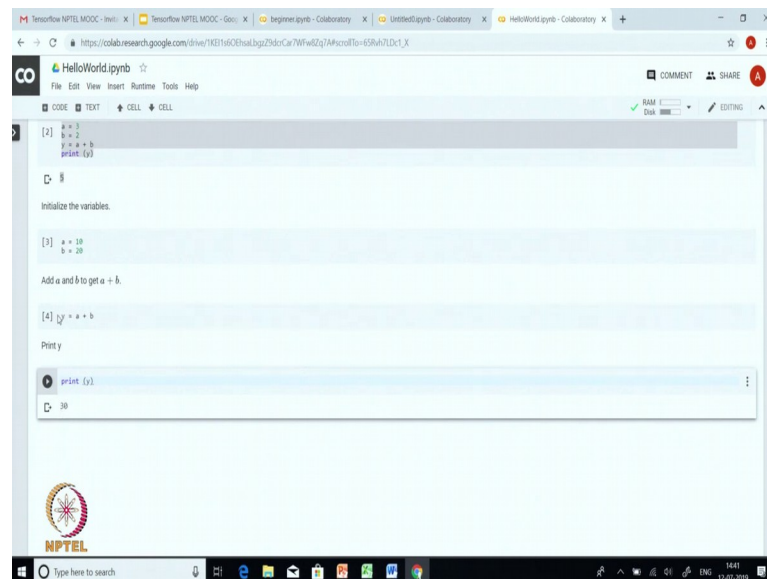
The Colab has mainly two cells, one is called a code cell and the second is called a text cell. In code cell, we essentially write the Python program whatever we write in the code cell and if we execute the code cell with the Run button over here the content of the code cell is interpreted by the Python interpreter and if it is the valid Python code then only that particular code would be run. So, let us try to print “Hello World”, in order to run the Colab we need to first connect to the cloud run time it is right now connecting it has connected and now initialize it is initializing.

You can see that you are connected to a cloud run time and we see the status of the RAM and disk in this particular cloud run time. In order to execute the cell, we simply need to run the cell and you can see that hello world is being printed over here. So, the other type of cell is a text cell; In the text cell, we can write content using what is called mark-down. So, I can write some sample content here: “This is my first Colab” and I can highlight some of the content using a visual editor here or if you are familiar with mark-down you can straight away use markdown syntax in Colab.

Apart from text we can also insert links or we can also insert images, we can also have list of items in Colab. So, we can say that in my first Colab here we demonstrate, “how to print using Python print statement” and “perform basic mathematical operations” and you can see that as I was typing the actual output is visible in this part of the screen. So, if you go to the other cell the output of the markdown can be seen over here. So, let us try to perform addition of two numbers and say that $a=3$, $b=2$, $y = a + b$. And we can simply print and we can see that the addition operation is carried out and we can see here finally, print y we can also write the content of this particular code cell across different code cells. Let us try to see that.

So, I can write comments like “initialize the variables” and then I can have simple code cell initializing a to 10 and b to 20, then I can say add a and b , let us say I want to write this a . So, I can also insert mathematical equations using latex I can simply write the code saying that y is equal to a plus b and I can say print y .

(Refer Slide Time: 20:18)



The screenshot shows a Google Colab notebook interface. The browser address bar displays a URL from research.google.com. The notebook has several tabs open, including 'TensorFlow NPTEL MOOC - Intro', 'TensorFlow NPTEL MOOC - Geo...', 'beginner.ipynb - Colaboratory', 'Untitled.ipynb - Colaboratory', and 'HelloWorld.ipynb - Colaboratory'. The 'HelloWorld.ipynb' tab is active, showing a file menu and a toolbar with options like 'CODE', 'TEXT', 'CELL', and 'CELL'. The code editor contains the following Python code:

```
[1] a = 10  
    b = 20  
    y = a + b  
    print(y)
```

Below the code editor, there are instructions: 'Initialize the variables.', 'Add a and b to get a + b.', and 'Print y'. The code is executed, and the output is displayed in a cell: 30. The NPTEL logo is visible in the bottom left corner of the notebook interface.

Let us write the code to print the addition of two numbers here. Now, we can run each of the cells and then perform the addition and finally, print the number, you can see that a was 10 and b is 20, we added this 2 numbers to get 30 in the resulting variable. So, this is the Colab environment, we can also import useful Python libraries like TensorFlow in Colab and then we can execute or we can build practical machine learning applications in Colab. One of the great point about Colab is that you can execute your Python code in the browser and you do not need to do a lot of complicated set up on your own machines. So, this brings us to the end of our first module. Hope you enjoyed it [FL].