

Applied Accelerated Artificial Intelligence
Dr. Satyajit Das
Department of Computer Science and Engineering
Indian Institute of Technology, Palakkad

Lecture - 24
Introduction to TensorFlow Part - 1

(Refer Slide Time: 00:15)



The slide features a title bar with the TensorFlow logo on the left and the NPTEL logo on the right. The main content is a list of bullet points describing TensorFlow. At the bottom right of the slide, there is a small video inset showing a man speaking.

- TensorFlow 1.x/2.x
 - An open source Deep Learning library
 - >1,800 contributors worldwide
 - Apache 2.0 license
 - Released by Google in 2015
 - TensorFlow 2.0
 - Easier to learn and use
 - For beginners and experts
 - Available today

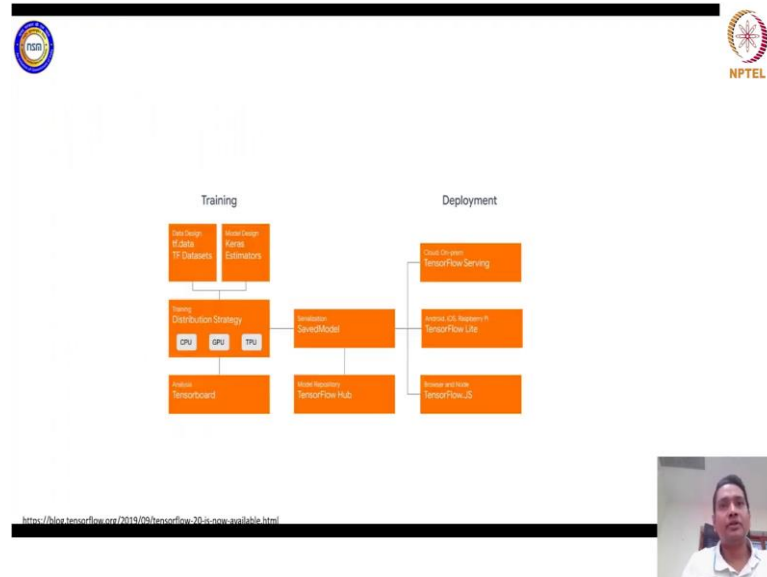
So, TensorFlow 1 was heavily based on making graphs and then introducing the lazy execution of the graphs, which mostly if you have followed the distributed programming of spark that is kind of same program paradigm. But in TensorFlow 2 that program paradigm of lazy execution has been transferred to your eager execution ok.

So, eager execution means as soon as you define something or any operation you define it will be evaluated in stand right. So, that is completely opposite of your lazy execution that was followed in terms of one and subsequent versions.

So, from the TensorFlow 2 we have eager execution and from the structural point of view TensorFlow 1 was a bit difficult to follow if you are coming from a completely PyTorch background. And by TensorFlow 2 is more of a pythonic approach and it can be very suitable for beginners to experts.

Because different levels of programming is there which you for you can follow to define your particular model that you want to train.

(Refer Slide Time: 01:44)



Now, so if you see the structure or architecture of TensorFlow. So, basically the TensorFlow architecture has 3 main blocks one is the training block is the safe and repository block in the middle you can see here. And the right hand side you can see the deployment block.

So, all these three blocks which have been introduced in 2.0 and subsequent versions this made the life much easier for the development and deployment of deep learning algorithms seamless ok. So, in the training module you will see that we have distributed training strategy core.

So, which is written in C++ and that supports CPU, GPU, TPU execution and of course, multi node CPU, multi node GPU, multi node TPU. So, all these kind of combinations that supports.

And for defining the model itself you have high level APIs as Keras and estimators, but it is recommended that you do not use estimators nowadays. Keras implemented from TensorFlow, so keras has two implementations one is native implementations. So, if you are using keras io and there is another implementation from TensorFlow itself and which we will use for defining our models.

And most of the tf 2 or TensorFlow 2 versions are very very compatible and efficient with this high level API defined by the keras. Now, we also have data design modules as

tf.data and TF data sets. So now, this data design in TensorFlow is bit different from that you have seen in PyTorch in the previous sections.

So, that also we will talk about and there are much more flexibility in terms of data access. Because the data sets all the necessary modules that are actually designed by the TensorFlow module designers itself. So, all the data that are imported by this data dot tf dot data will have the flexibility to directly convert into tensors which is the main data structure in TensorFlow and use it subsequently.

So, in the entire pipeline you do not need to convert it into with the tensors and deliver to your GPU or TPU right. So, in terms of flexibility this tf dot data adds one more level of flexibility and efficiency while you are accessing data. There is this analysis or visualization tool which is also available in PyTorch plugin that we have seen so far, in the previous classes and also we here also TensorBoard is available with TensorFlow is available with TensorFlow packages.

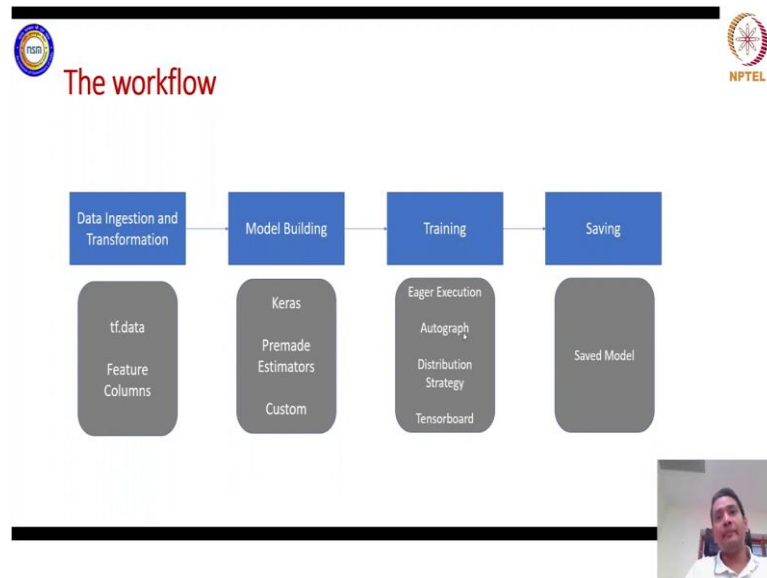
Now, for the saving model we have the serialization saving model. So, where you can save your model instantly and reuse it and also there is TensorFlow hub. So, TensorFlow hub is model repository. So, if you want to use already trained models which are already trained with defined parameters you can use those parameters you can tune them in your training depending on the target model that you are targeting and the target problem that you are targeting.

So, basically if you are targeting classification problem depending on the number of classes that is available you can tune the already existing parameters from the TensorFlow hub. Now, the deployment part is completely giving the TensorFlow a new dimension. So, the models that you have defined or trained or designed inside the TensorFlow modules you can seamlessly convert it or deploy it on onto your cloud based on browser.

So, if you are using browser you can deploy your browser with using TensorFlow.JS if you are try trying to deploy your models for android devices or ios devices or raspberry pi or even android arduino support it has with TensorFlow Lite micro. So, TensorFlow Lite and TensorFlow Lite micro will have you need to use just to save or just to use your saved models and convert to the targeted ways.

Now, for the target cloud platform you can use TensorFlow serving right. So, all these deployment strategies are inbuilt. So, different strategies there are and we will see how to use. So, all these three stacks make actually the TensorFlow a bit flexible and flexible for productable and product oriented projects right.

(Refer Slide Time: 06:50)



So, now the workflow is similar to that of your PyTorch. So, you have to feature your data and feature engineering your data.

Transform or whatever you want to normalize your data, so all these things you will do in the beginning and then you have to define the model build the model. So, as I was mentioning that you can use keras premade estimators. So, these estimators as I was mentioning that it is not recommended for the newer versions or TensorFlow 2 after TensorFlow 2.5 actually it is not recommended.

And also you can use some APIs to custom build your models. So, keras is having all the models mostly 90 percent of the models that are necessary to build your models and if you want to go a bit more flexible. And to have more control over model building you can use this custom models building APIs right.

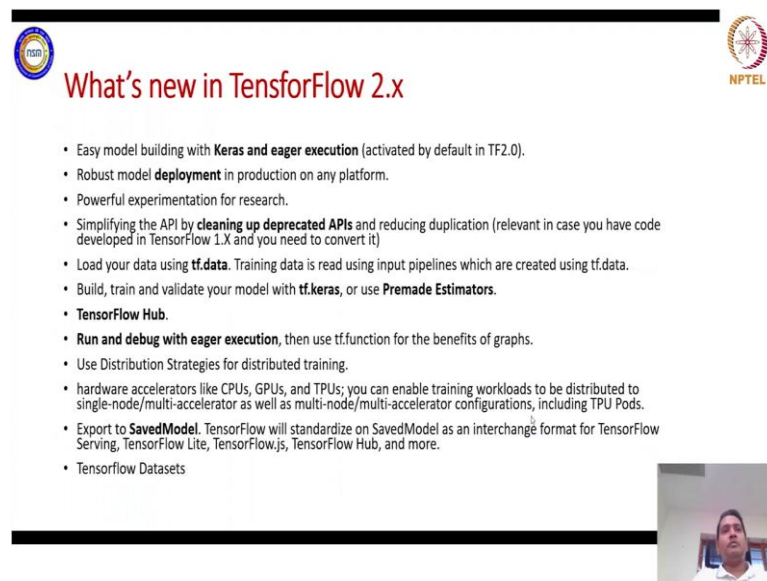
And then in the training stage of this pipeline you have this eager exhibition as well as autographs. So, this autograph will give you the efficiency of the graph based approaches of TensorFlow 1. So, once one way you have the eager execution which is actually the

instant evaluation. So, in compile time itself you will see what kind of errors are there and you can evaluate your model depending on the definition that you have defined.

And you can use autograph to enhance the performance of the model in the subsequent runs ok. So, we will see how to define all these explicit APIs to be used to enhance the efficiency. Distribution strategy as I was mentioning that distribution strategy was written in code C++.

And that gives it much more flexibility in terms of distributing the data parallelism model parallelism as well as the pipeline parallelism. And of course, TensorBoard is inbuilt tool which you will use for visualize and analyze your model and give it much more boost in the subsequent ones. And in the end you save the model and use it for development right.

(Refer Slide Time: 09:25)



The slide is titled "What's new in TensorFlow 2.x" and features the TensorFlow logo on the left and the NPTEL logo on the right. The main content is a bulleted list of new features and improvements in TensorFlow 2.x. A small video inset in the bottom right corner shows a person speaking.

- Easy model building with **Keras and eager execution** (activated by default in TF2.0).
- Robust model **deployment** in production on any platform.
- Powerful experimentation for research.
- Simplifying the API by **cleaning up deprecated APIs** and reducing duplication (relevant in case you have code developed in TensorFlow 1.x and you need to convert it)
- Load your data using **tf.data**. Training data is read using input pipelines which are created using tf.data.
- Build, train and validate your model with **tf.keras**, or use **Premade Estimators**.
- **TensorFlow Hub**.
- **Run and debug with eager execution**, then use tf.function for the benefits of graphs.
- Use Distribution Strategies for distributed training.
- hardware accelerators like CPUs, GPUs, and TPUs; you can enable training workloads to be distributed to single-node/multi-accelerator as well as multi-node/multi-accelerator configurations, including TPU Pods.
- Export to **SavedModel**. TensorFlow will standardize on SavedModel as an interchange format for TensorFlow Serving, TensorFlow Lite, TensorFlow.js, TensorFlow Hub, and more.
- Tensorflow Datasets

So, as I was mentioning that what why we are actually moving towards the 2.x version is that the keras, which is the newer implementation from the TensorFlow and the eager execution and this eager execution is highly efficient from the pythonic approach of programming. So, in PyTorch you have seen some and in TensorFlow 2 and subsequent versions we will see much more of that eager execution.

So, this is just to have a look of the entire model that you have defined in compiler right and also robust model deployment. So, deployment that third stack you have seen it is

actually giving you the extra boost to use it over PyTorch ok powerful experimentation for research. So, for researchers there is some custom appearance that is that they can use to build or define the models from the scratch itself.

So, that is also a similar to PyTorch, so whatever we have seen that we will discuss in the subsequent slides. Now, you have this deprecation of APIs because many of the APIs that you are if you are coming from your TensorFlow 1 code to your TensorFlow 2 code. Many of the APIs have been deprecated and cleaning or clean up your code is very easy with clean out cleaning up APIs.

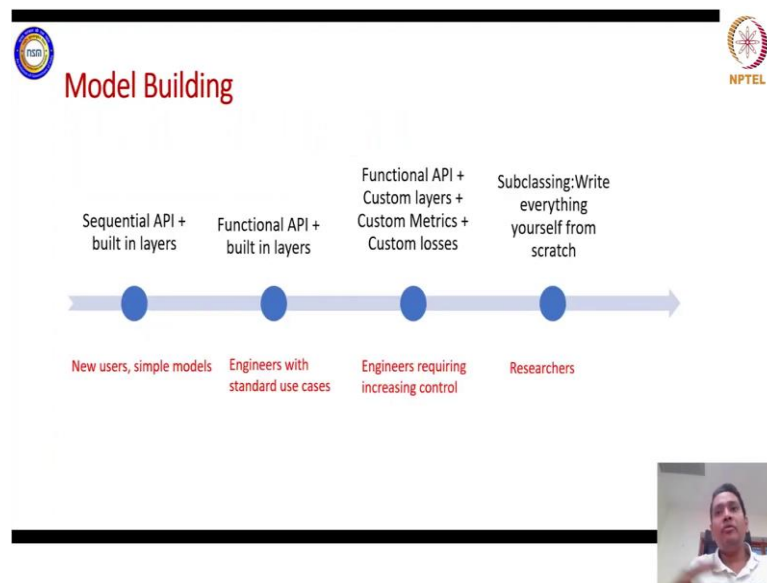
And `tf.data` as I was mentioning that loading data is giving another dimension because there are many APIs in `tf`, which is defined to distribute the data of which you are loading the to have prefetching options to have pipeline parallelism options. Also to have cache options whether you want to store your prefetch data into your ram or your disk. So, all these APIs are already built into `tf.data` `tf.keras` that I will just mentioned that you can build the models with `keras` API high level API.

TensorFlow hub which is the already saved repository already saved models and parameters that you can get from. Also run and debug with eager execution debugging also is very very much possible with some one or two lines of code addition. Now as I was mentioning that it is highly efficient to run in CPUs, GPUs, TPUs and distribution strategies is it is much more flexible and you can customize the distribution strategy.

We will spend most I think maybe 50 percent of the time in the next class to discuss this distribution strategy ok. And exporting saved models usually this will help you to deploy efficiently using TensorFlow.js or TensorFlow Lite or TensorFlow Lite micro depending on the target environment that you are targeting right. And TensorFlow data sets already it has numerous data sets that you want for your knowledge training most of the problems targeting if you are targeting.

Let us say classification you are targeting computer vision, problems NLP problems even segmentation or multi stage definition of let us say segmentation and captioning of videos. So, different data sets are available to give you much more flexibility to use this `tf.data` APIs to be used with the data sets that is already available.

(Refer Slide Time: 13:29)



So, now we will come to the model building.

So, model building part as I was mentioning that is giving you the options to start with small beginners. So, as you can see here from the left side you have beginners to the experts on the right side and for the beginners defining simple models ok it is very easy to use sequential API ok.

So, there are several APIs that is available in software as mentioning that it is to give you flexibility and control. So, this API if you are using sequential API most of the models you can build with this essentially because this API has all the built in layers ok. That it can be used and you can define and you can use to define your simple models.

Next stage, if you are let us say engineer with standard use cases and you want to use the built in layers of functional API. So, basically if you see with the sequential API the sequential API will give you the platform to define one model which has one input and one output ok. Of course, that input and output is multi dimensional no need to mention that, but the functional API has the flexibility to now you will see that the flexibility is increasing step by step in the APIs that we are going on the right side right.

So, now it has the improved flexibility to add of course, the built in layers that you will use, but now you will have the option to define multiple inputs and multiple outputs. And also those inputs and outputs can be at several stages you can merge them you can use to

branch them or you can define your model using multiple inputs and outputs depending on which stage you are using. So, as you can see sequential API simple models for beginners.

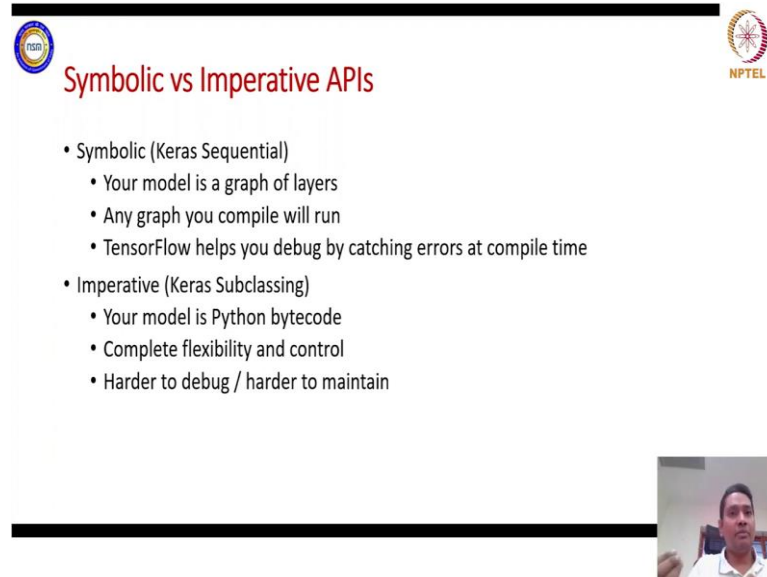
Functional API for built in layers you will use, but you can define the multiple inputs and outputs. Now, if you go to have much more control over this functional API you can have functional API plus custom layers plus custom metrics plus custom losses. So, metrics losses these are essentially very important or crucial parameters that you want to track while training a model right.

Now, you can define custom metrics which metrics you want to track most of the metrics that you use as we have seen in the last catalyst class. So, all the metrics were actually defined in the in that framework as well as in TensorFlow 2 point 0 will have many metrics that is already available you can define custom metrics as well custom losses and define custom layers used with functionality.

So, functional API is essentially call the layers as functions. So, the entire sequential api will give you the stack of layers. So, this is completely one data structure in the functional API you will have this the graph of the built in layers ok and you can call them ok. So, the layers are callable now and in subclassing this is completely relevant if you are coming from PyTorch. So, subclassing is same as programming from scratch whatever you have done in PyTorch ok.

Now, you can say that why we need functional API right, we can do it from scratch we can either use for simple models with sequential API then why we need functional API. But functional API has much more to it than you are seeing here ok. So, we will discuss how we can actually define one module in using functionality very very efficiently not going for bit sequential and why we will not go for subclass ok.

(Refer Slide Time: 18:01)



The slide features a title "Symbolic vs Imperative APIs" in red text. On the left is a circular logo with "NPTEL" inside. On the right is the NPTEL logo. The main content is a bulleted list comparing two API styles. At the bottom right, there is a small video inset showing a man speaking.

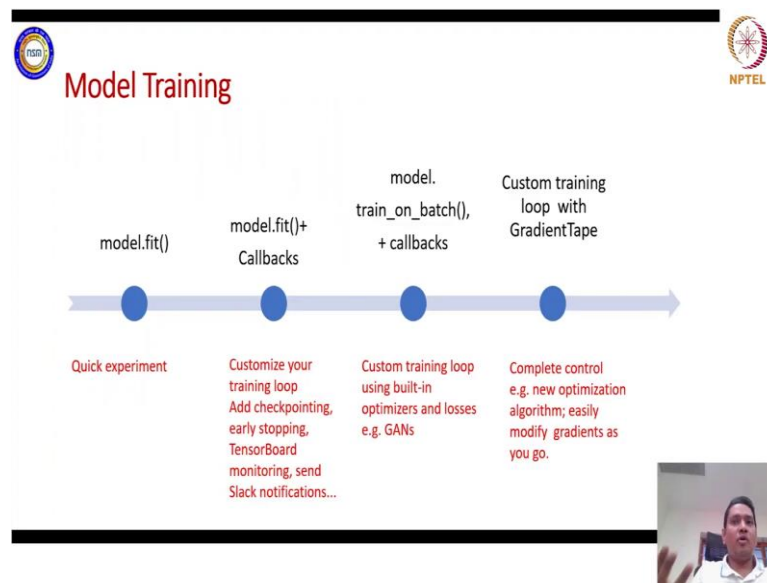
- Symbolic (Keras Sequential)
 - Your model is a graph of layers
 - Any graph you compile will run
 - TensorFlow helps you debug by catching errors at compile time
- Imperative (Keras Subclassing)
 - Your model is Python bytecode
 - Complete flexibility and control
 - Harder to debug / harder to maintain

Now, this sequential way of defining. So, there are two programming paradigms that is supported in TensorFlow, that I mentioned in the first slide itself. So, tensor is essentially the data structure and the flow is essentially with the graph computational graph that you want to execute.

Now from TensorFlow 2 we have this symbolic API. So, which is keras sequential API that you will use to define, where as your model is a graph of layers and any graph you can compile will run. And TensorFlow helps you to debug by catching errors at compile dimension because it is supporting the eager execution not the lazy execution which was in TensorFlow 1, now in the imperative API.

So, sub classing is imperative API. So, this is kind of object oriented programming style defining your models and using. So, in that sub classing your model is essentially converted into your Python bytecode. And you will have much more control over the models that you are defining and since it is completely customized, according to your requirement it is bit difficult to debug, But of course, the if you know the ways to develop it is not that hard.

(Refer Slide Time: 19:34)



Now, the training part. So, model defining as you have seen there are four kind of approaches or ways you can define it and training the model also is kind of tricky if you are coming from PyTorch system.

So, model dot fit this function is for quick expect you just call the forward function and it will do automatic gradient descent calculating the gradients updating the weight parameters and biases. So, all these things will be completely automated just call the fit model over the model or feed function over the model that is now callbacks. So, callbacks you have seen in the previous class also in the catalyst.

So, that same notion is also here you can use callbacks you can customize your training loop by adding checkpoints early stopping you can have Tensorboard monitoring slack notifications. So, this is called quite interesting you can define your model you can now start that training hit the training with callbacks you can define one callback which will give you slack notification for each or let us say you talk Python ok.

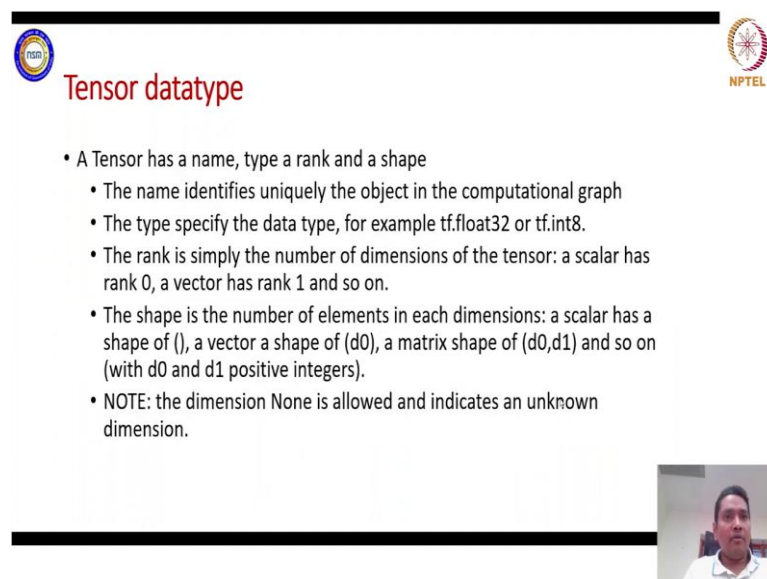
So, for each epoch what is the metric status or status of your training it will give you a notification. So, you can all add all these callbacks in inside here. So, you can see model dot fit is just simple way of training the vanilla way of training or maybe automated completely automated in the callbacks you can add checkpointing early stopping. So, all these customizations you can add now `model.train_on_batch()`.

So, this method plus callbacks. So, you can now customize the training loop also. So, customizing training loop with training on batch because training loop has several options batch optimizers, which loss function you are using. So, all these are actually automated here in this fit functions train on batch plus callbacks if you use you can customize your optimizer you can customize your losses you can customize your interactive training loop batch.

And completely scratch development of your training loop also you can do with custom training. So, here with gradient tape so this is a very simple API which keeps track of all the parameters which are differentiable ok. So, using this gradient tape we will see how to program with gradient tape. So, we using gradient tape we can customize or completely write your training module or training loop from scratches. So, optimizations modifying gradients how you want to define your loss function.

So, loss function as I was mentioning. So, different loss functions are there mse loss, cross entropy loss right. Now, if you want to define your own loss function you can use custom training loop with gradient tape, where you can define your functions for your loss functions and you can call that function inside gradient tape to be accounted for your optimizing.

(Refer Slide Time: 23:05)



The slide features a title 'Tensor datatype' in red text. To the left is a circular logo with 'NPTEL' inside, and to the right is the NPTEL logo. Below the title is a bulleted list:

- A Tensor has a name, type a rank and a shape
 - The name identifies uniquely the object in the computational graph
 - The type specify the data type, for example `tf.float32` or `tf.int8`.
 - The rank is simply the number of dimensions of the tensor: a scalar has rank 0, a vector has rank 1 and so on.
 - The shape is the number of elements in each dimensions: a scalar has a shape of `()`, a vector a shape of `(d0)`, a matrix shape of `(d0,d1)` and so on (with `d0` and `d1` positive integers).
 - NOTE: the dimension `None` is allowed and indicates an unknown dimension.

In the bottom right corner, there is a small video inset showing a man speaking.

Now, as this is the basic of TensorFlow, so tensor data type. So, tensor data type is you have seen also in PyTorch tensors definition of tensors right, but tensors in TensorFlow

is similar it has few features different from PyTorch tensors, but it is completely same as your numpy error it is exactly same.

However, you have defined your multi dimensional simple single dimensional array in numpy it is the same way of definition same way of function calling same way of slicing. So, everything you can use exactly same in tensor in tensor flow. So, tensor data type has a name and type and a rank.

So, type is essentially the data type rank is essentially how many elements it has and a shape. So, the name identifies uniquely the object which you want to use in your computational graph the computational graph means. So, you if you recall the computational graph from the first PyTorch class you have let us say a tensor you have let us say weight tensor w .

You have your tensor input as x , so w into x if you are doing. So, that is one flow of operation. So, this data is fit into the multiplication operation. So, that is simple flow of data and this flow of data is represented in computational graph. Now, in computational graph if you have what kind of data type you are using as tensors. So, these are tensors the input tensor, the weight tensor, the bias tensor. So, all these are tensors.

And what data type you are using there now it has lot of support for data type. That is float32 int8 int32 float16 you can look into TensorFlow's website TensorFlow.org. To get more of these data types what are the database that are available and how you can define your tensors for a particular data, rank is simply the number of dimensions of the tensor. So, basically scalar means it has 1 element, so rank is 0.

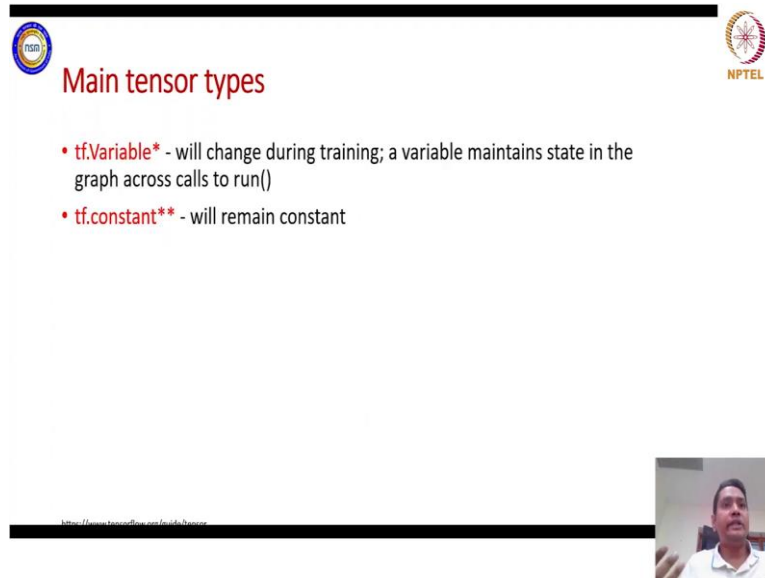
Vector has rank 1. So, basically one dimension it has right the shape is the number of elements in each dimension. So, let us say rank 2 tensor, so 2 dimensions are there. Now in each dimension how many elements will be there. So, that will be defined by the shape. So, let us say a vector of shape d_0 , so basically d_0 is the number of elements.

Mac matrix of shape $d_0 d_1$. So, d_0 is the elements number of elements in the first dimension maybe it row and column let us say d_1 . So, how many elements are there, so as simple as that and the dimensions are not none. So, there can be none also as dimension ok is allowed and it indicates unknown dimension. Because in some of the programming of models you will be using some tensors where you do not know how

many batches will come or how many output of in particular dimension for first dimension how many elements will be given as output.

So, you do not know in let us say defining time of your model you can use none in those places ok.

(Refer Slide Time: 27:03)



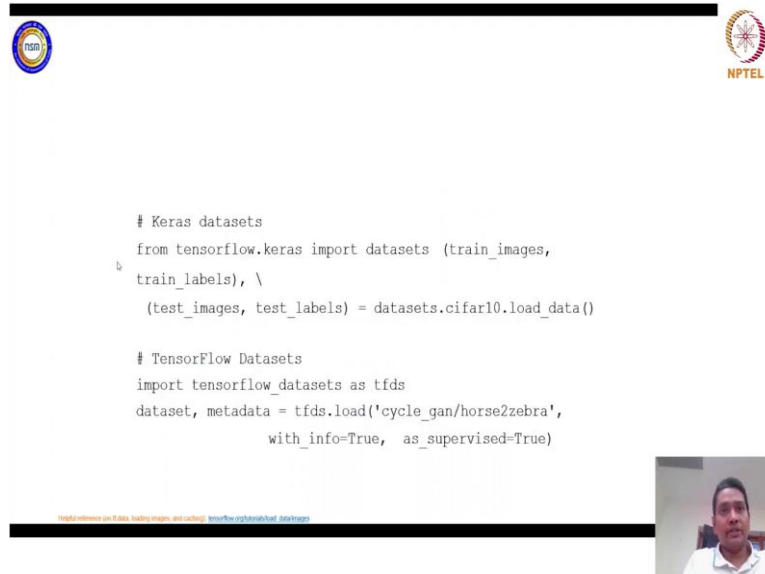
The slide is titled "Main tensor types" and features two bullet points. The first bullet point is "• `tf.Variable*` - will change during training; a variable maintains state in the graph across calls to `run()`". The second bullet point is "• `tf.constant**` - will remain constant". The slide also includes a small video inset of a man in the bottom right corner and logos for IITM and NPTEL.

Next, we will see the different types of tensors. So, there are some immutable tensors there are mutable tensors. So, immutable tensors means you cannot change your values like constants. So, `tf.constant` will be used to define constant tensors you can define variable tensors, but remember that `tf.Variable` which you are defining. But `V` is capital `V` this is just the notion that TensorFlow uses, but for the constants `c` is small.

So, variables the data that you are defining tensor as you can change their values ok. But most of the trainings you do not need to worry about that you do not need to define your tensors as lower level as variable or constants if you are starting with sequential or even functional event. If you are defining your tensors from scratch without using your TensorFlow data type `tf.data` you can use these variable definitions for constants and variables.

But most of the time you will not use them ok, but it is nice to know. And now this `tf.data` is as mentioning that `tf.data` is much more flexible in terms of data loading.

(Refer Slide Time: 28:36)



The slide displays two code snippets for loading datasets. The first snippet shows how to load CIFAR-10 data using Keras datasets. The second snippet shows how to load a custom dataset using TensorFlow Datasets. A small video inset in the bottom right corner shows the speaker.

```
# Keras datasets
from tensorflow.keras import datasets (train_images,
train_labels), \
(test_images, test_labels) = datasets.cifar10.load_data()

# TensorFlow Datasets
import tensorflow_datasets as tfds
dataset, metadata = tfds.load('cycle_gan/horse2zebra',
with_info=True, as_supervised=True)
```

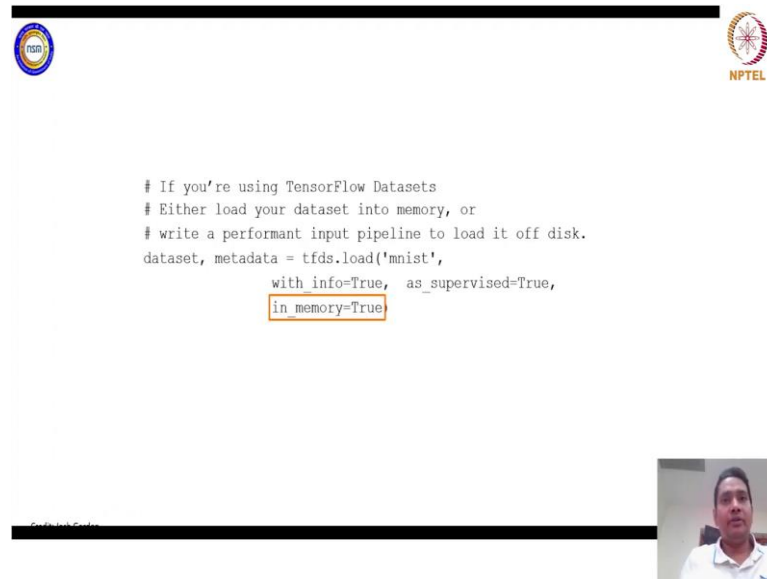
And how it is. So, we will just see a few examples then because this is just to show you the flexibility how to define that, but we will use them explicitly and define how to define your TensorFlow data sets TensorFlow data loaders, which actually storing the data not such data loaders that you have seen in PyTorch.

Now, keras data set you can use also you can use TensorFlow datasets. So, I mentioned that keras has its own implementation and keras has its TensorFlow implementation in our cases, we will use keras TensorFlow implementation. And of course, keras TensorFlow, TensorFlow.keras has lots of datasets available.

Now, the if you are importing data sets from keras it is different if you are importing data sets from TensorFlow datasets. So, tf dot t, so tfds if you are using tfds that is different. And if you are using keras dataset that is different because TensorFlow datasets are already tensors you do not need to you do not need to convert them or transfer them for your GPU or TPU which you are using.

And for your keras data sets it is not the tensors actually they will be imported as datasets as or numpy data structures.

(Refer Slide Time: 30:14)



The slide features a code editor with the following Python code:

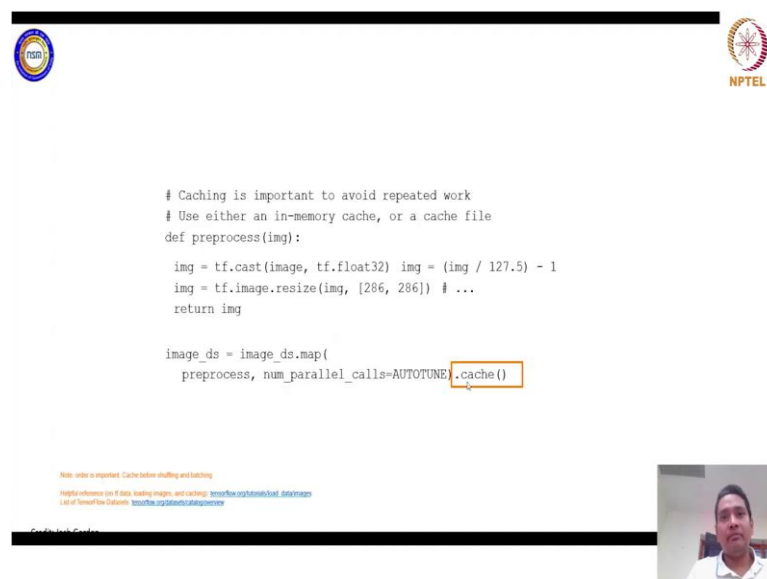
```
# If you're using TensorFlow Datasets
# Either load your dataset into memory, or
# write a performant input pipeline to load it off disk.
dataset, metadata = tfds.load('mnist',
                             with_info=True, as_supervised=True,
                             in_memory=True)
```

The code is framed by a black border. In the top left corner is a circular logo with 'NPTEL' inside. In the top right corner is the NPTEL logo. In the bottom right corner, there is a small video feed of a man speaking.

Now, if you are using TensorFlow datasets you have the flexibility to define the flag in memory true or false because tfds can load your data into your memory into RAM to give you much more easy and faster access ok to the data. Let us say you are trying your data to be processed by CPU and if you are using keras data set then you are actually defining to be stored into the disk you then load into the RAM.

And here you directly load into the ram as tensors and then you there you go ok.

(Refer Slide Time: 30:58)



The slide features a code editor with the following Python code:

```
# Caching is important to avoid repeated work
# Use either an in-memory cache, or a cache file
def preprocess(img):
    img = tf.cast(image, tf.float32) img = (img / 127.5) - 1
    img = tf.image.resize(img, [286, 286]) # ...
    return img

image_ds = image_ds.map(
    preprocess, num_parallel_calls=AUTOTUNE).cache()
```

Below the code, there is a note: "Note: order is important. Cache before shuffling and batching". There are also links for "Helpful reference (on TF data loading images, and caching)" and "List of TensorFlow Datasets". The slide is framed by a black border. In the top left corner is a circular logo with 'NPTEL' inside. In the top right corner is the NPTEL logo. In the bottom right corner, there is a small video feed of a man speaking.

You can cache as well by using dot cache function. Now, cache you can cache your data set whichever you are mapping for your let us say for this pre process function. So, let us say I want some images some image classification problem I am trying to address, where I need to pre process the image data before applying them into the models.

Now, these pre process function will be used several times depending on the number of batches or number of images that you are using inside your model. Now, if you are having this pre process inside I do not know where you need to access that many times you need to call them right. Now, you have the flexibility to cache it inside ram or disk depending on let us say you are trying to cache anything as file inside your disk or cache anything inside your RAM you can do that.

And here we are using map function to actually map this pre process with this many number of parallel calls with cache cached version inside your ds ok direct. So, that is also possible with image ds which is the tf.dataset right.

(Refer Slide Time: 32:24)

The slide features a header with the IITM logo on the left and the NPTEL logo on the right. The word "DEMO" is prominently displayed in the center. Below the title, three lines of text provide links to Google Colab notebooks, each labeled as a demo. A small video inset in the bottom right corner shows a man speaking, likely the presenter.