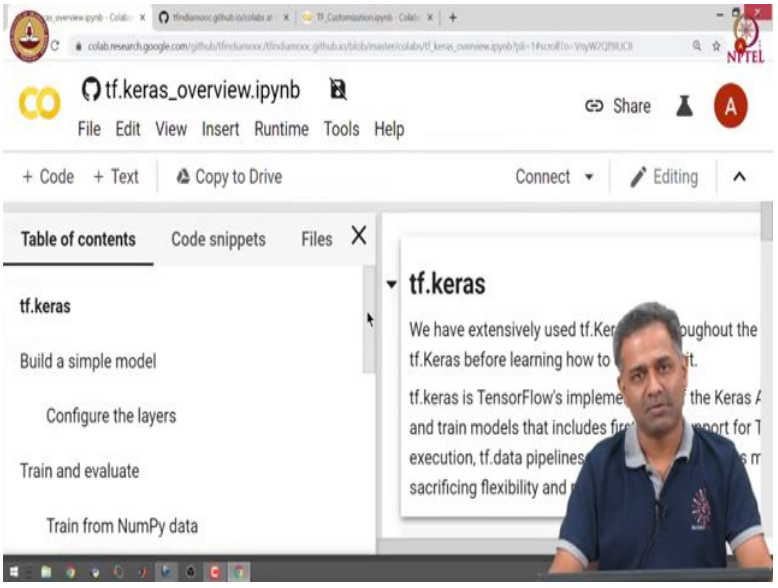
Practical Machine Learning with Tensorflow
Dr. Ashish Tendulkar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 37
Customizing tf.keras – Part 2

In this session, we will review some of the main concepts behind tf.keras. You have been using tf.keras quite extensively in this course so far and now we want to study how to customize tf.keras.

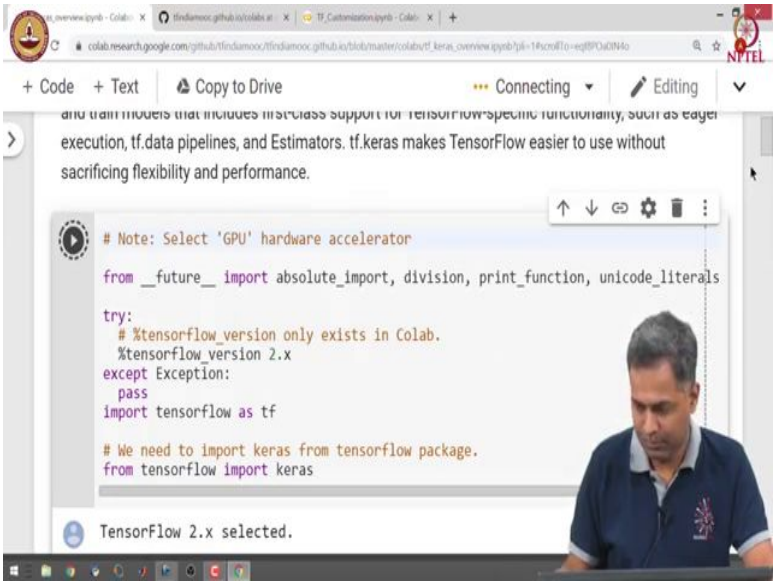
(Refer Slide Time: 00:34)



Before getting into customization it is a good idea to take a step back and review all the concepts of tf.keras that we have been using so far.

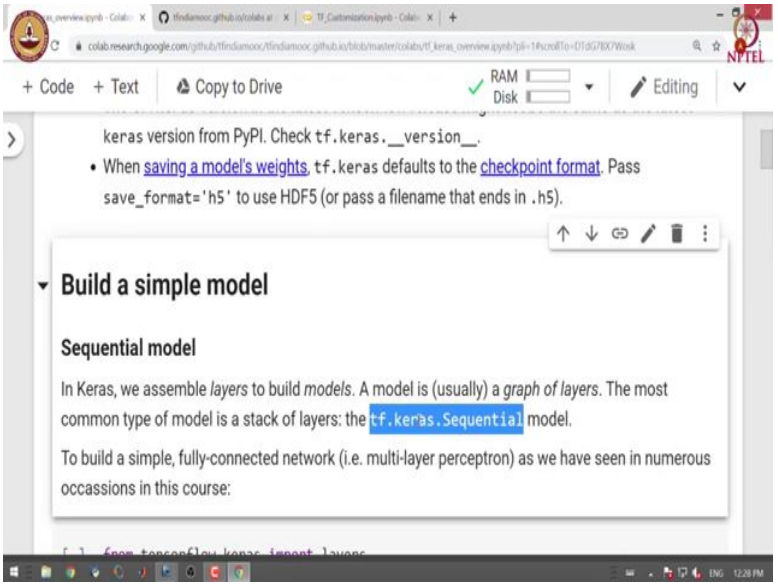
So, in this lecture what we will do is we will start with how to build a simple model with tf.keras, how to perform training and evaluation where we will read data from multiple sources like NumPy using tf.data.Dataset and we will see how to perform evaluation and prediction. We will also see how to define custom callbacks which are very important in obtaining interesting insights from training. Finally, we will look at how to save and restore the model and run the model in multiple GPU settings.

(Refer Slide Time: 01:31)



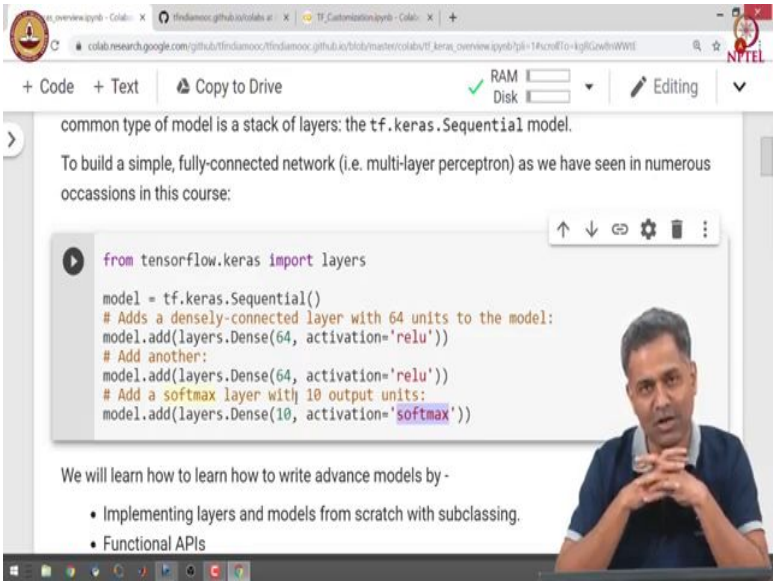
Let us begin with setup. We will import TensorFlow 2.0 and in order to use keras you have to import keras package from TensorFlow.

(Refer Slide Time: 02:05)



Let us begin by reviewing how to build a simple model with keras. So, we assemble layers to build simple model. A model is usually graph of players. The most common type of model is stack of layers, it is called tf.keras.Sequential model.

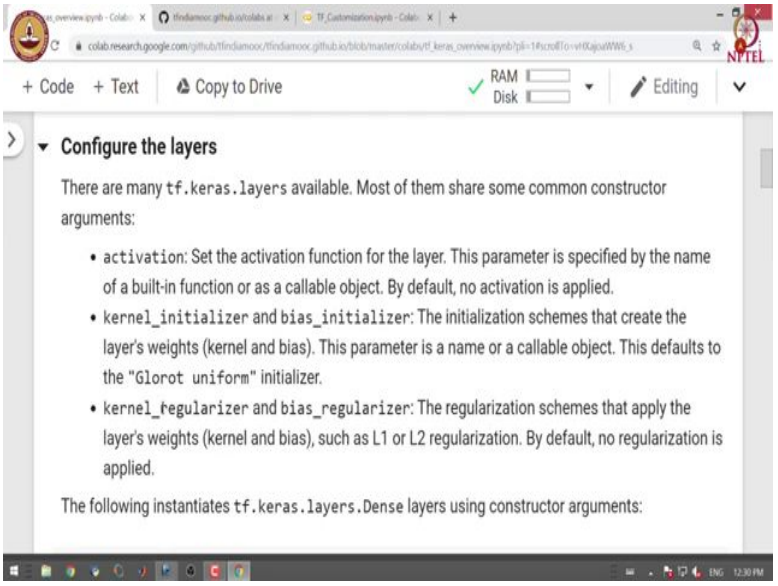
(Refer Slide Time: 02:34)



To build a simple fully connected network as we have seen in numerous occasions in this course we do the following. We define our model to be a sequential model and then we go on adding layers in the model. First we add a dense layer with 64 units in it and we use relu as an activation function. We add another layer with 64 dense units and finally, we have another layer which is output layer with 10 units. It uses softmax as an activation function.

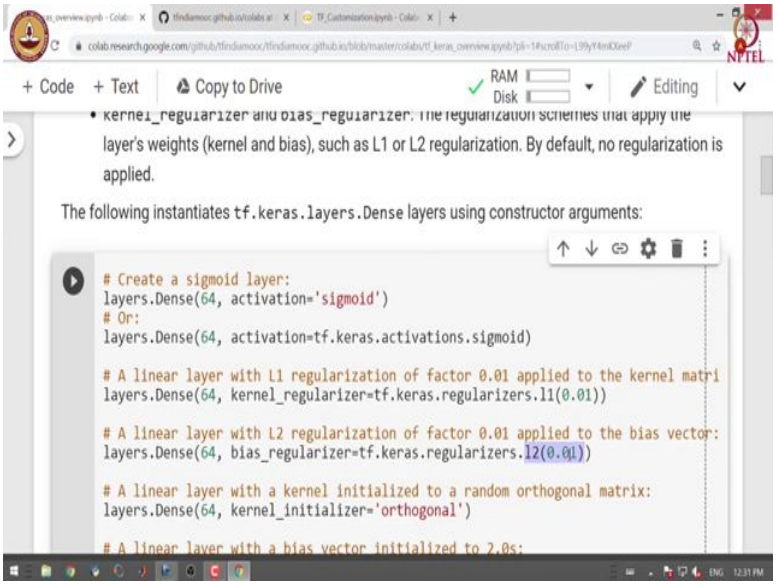
Now, what we will do is we will learn how to write advanced models. The couple of ways in which we can write advance model – first by implementing layers and models with subclassing or by using the functional APIs.

(Refer Slide Time: 03:52)



Some of the common arguments are activation, weight initializer and bias initializer; weight initializer is referred to as kernel initializer. Then, we also specify the regularizers in the layer.

(Refer Slide Time: 04:12)

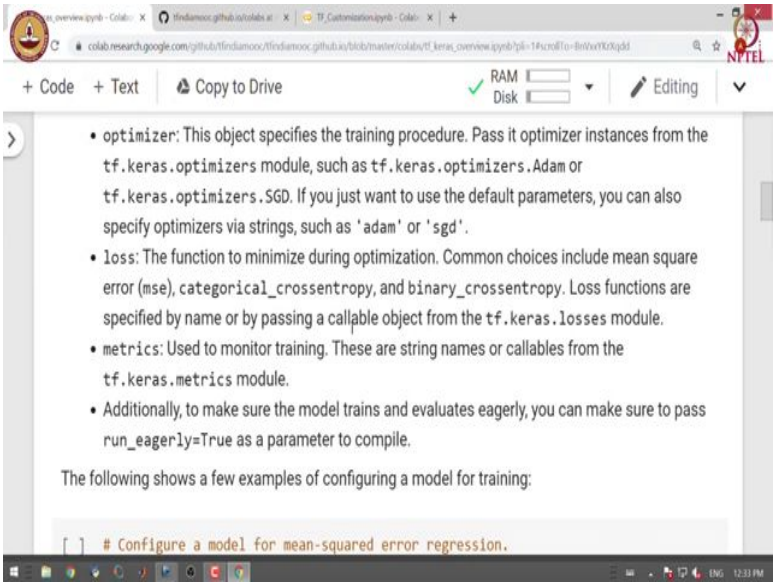


So, we can see that here we can use activation using a simple string or using expanded name like tf.keras.activations.sigmoid. So, that is the long form of sigmoid activation.

We can see above that we are adding l1 regularization with a regularization factor of 0.01; here we are adding a bias regularizer which is l2 regularizer with a factor of 0.01. We can also define the way to initializer to be orthogonal in this case. We can also initialize bias in the layer.

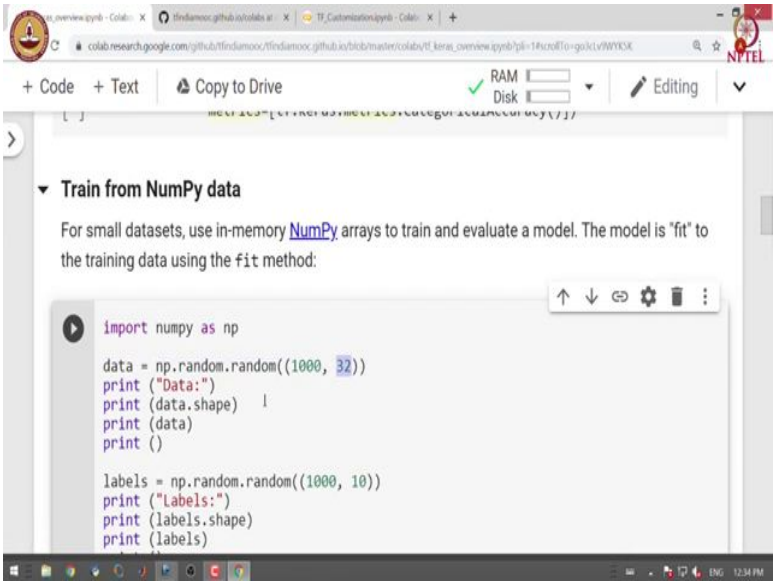
After the model is constructed we configure its learning process by calling the compile method. In compile method we specify the optimizer to be used then the loss function and metrics.

(Refer Slide Time: 05:37)



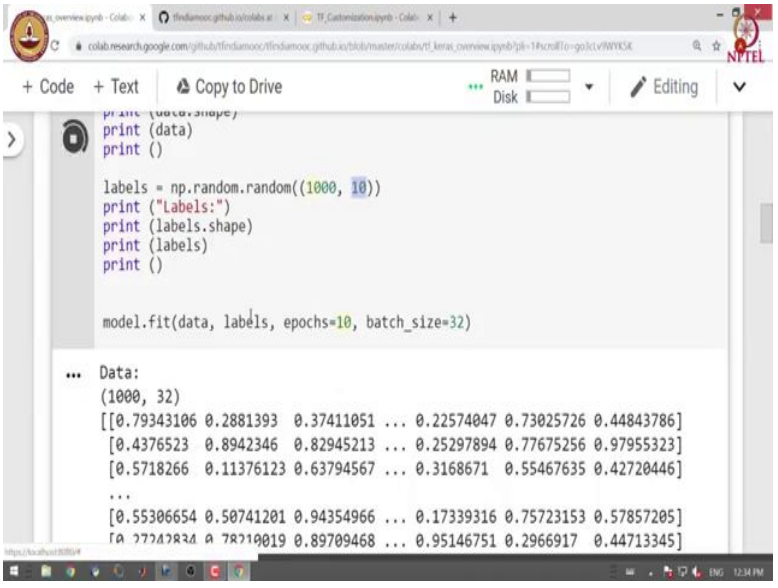
The optimizer object specifies the training process. We use optimizers like Adam or stochastic gradient descent. There are multiple loss functions that are defined already in tf.keras package. The common choices include mean squared error, categorical cross entropy loss and binary cross entropy loss. The loss functions are specified by name or by passing callable object from tf.keras.losses module. Then we will also specify metrics use to monitor the training. We can use string names or callables from tf.keras.metrics module.

(Refer Slide Time: 07:02)



Let us use a small data set which is in memory in NumPy array to train and evaluate model. We use the fit method for training the model. Here you define a NumPy data with random distribution. You have 1000 examples each with 32 features.

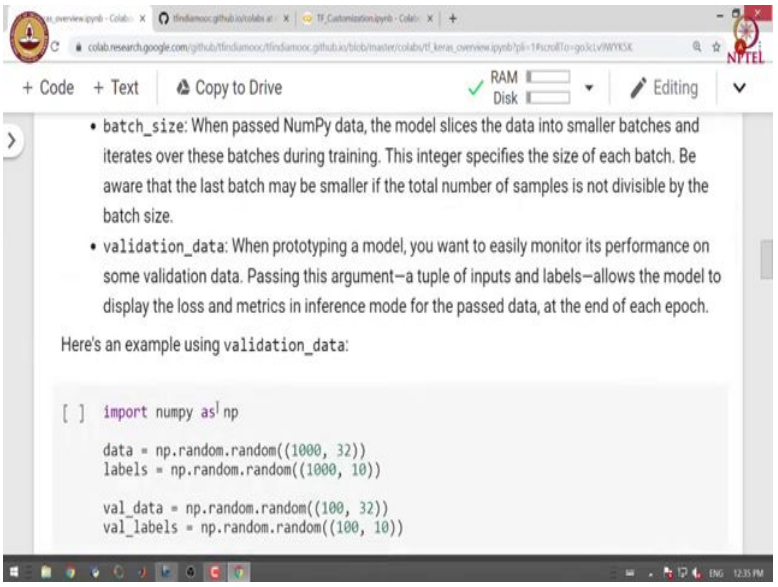
(Refer Slide Time: 07:32)



And, the output is again 1000 examples and there are 10 classes. We randomly initialize both labels and examples and train the model.

The fit function takes three important arguments. Epochs which is the number of times you have to iterate through the training set.

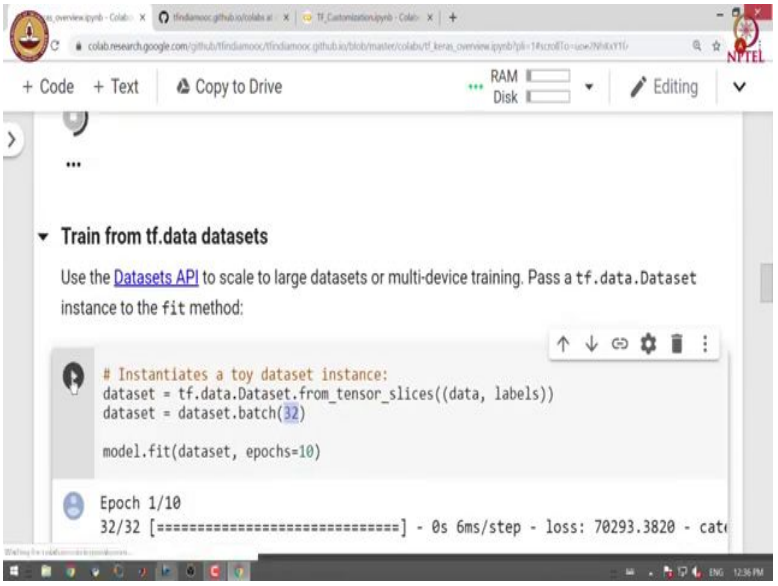
(Refer Slide Time: 07:59)



Then the batch size, and validation data on which we can calculate some of the metrics of interest.

So, we can specify the validation data with validation data and the corresponding labels apart from epochs and batch size.

(Refer Slide Time: 08:24)



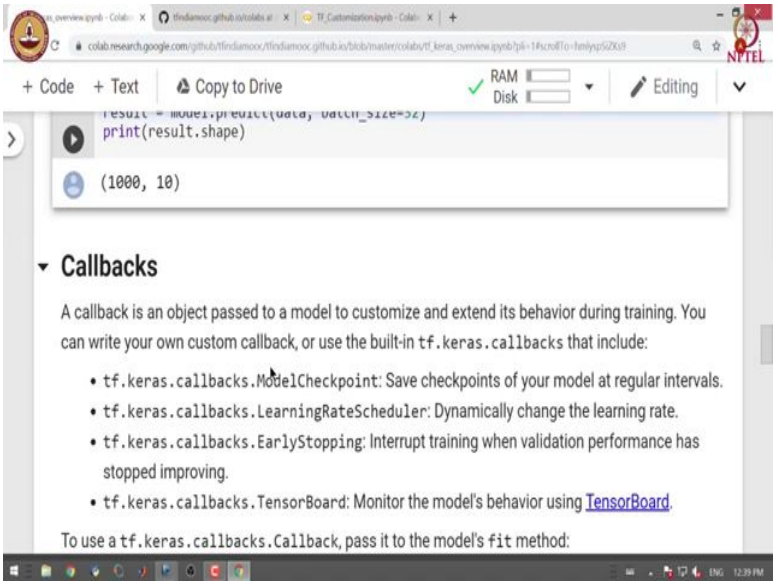
If we have a large data set on which we want to train the model we use Dataset API for scaling to large data sets or multi device training. We pass tf.data.Dataset instance to the fit method.

Let us look at a concrete example of the same. We create a dataset from tensor slices where we have data and corresponding labels and we batch the dataset in the batch of 32 examples each. We fit the model for 10 epochs. Since the data set yields batches of data we need not specify batch size along with other arguments in the fit function. We can also use dataset for validation.

In the same manner, we define a validation dataset just like the training data and we specify validation dataset in the validation data argument. We use evaluate method for evaluating the performance of the model and predict method for inference. Here we use evaluate method to find out loss under training data. In this part we calculate the loss on the training data when the data was stored in memory through NumPy arrays and in this particular part of the code we demonstrate how we can calculate the loss on the training data when we need the data through dataset API.

Finally, we use predict for inference.

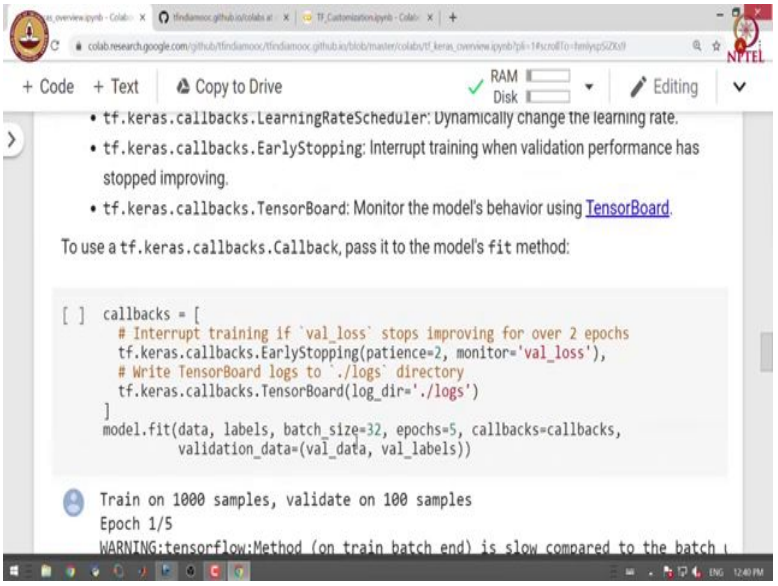
(Refer Slide Time: 11:10)



We use callbacks for customization and extending the behavior of model during the training process. We can also write our own callbacks or use some of the existing tf.keras.callbacks.

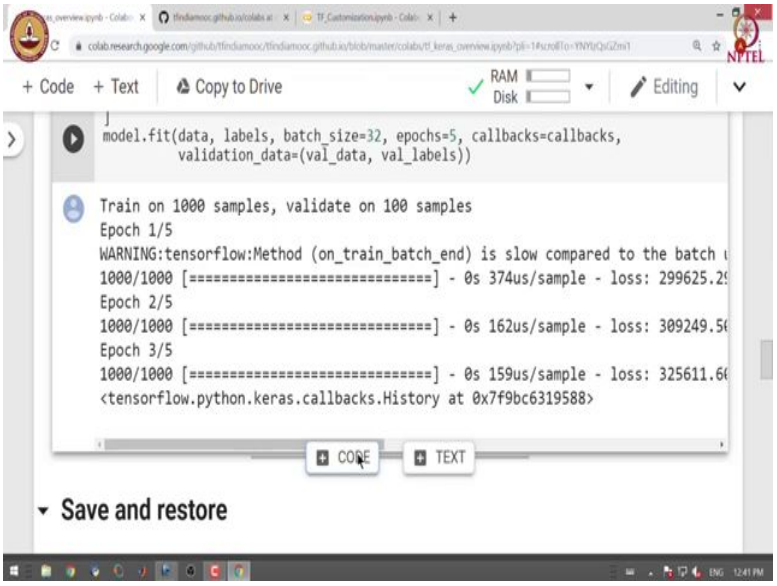
One of the well known callbacks is ModelCheckpoint which is used for saving the model at regular intervals. You can also use LearningRateScheduler for dynamically changing the learning rate. EarlyStopping scheduler can be used for stopping the training when validation performance has stopped improving and we can use TensorBoard callback for monitoring the behavior of the model during the training process.

(Refer Slide Time: 12:13)



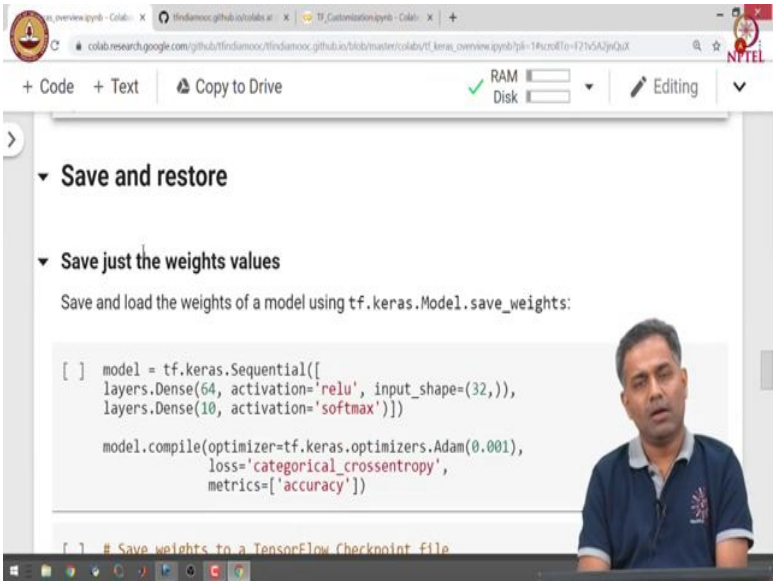
To use the callback we generally specify the callbacks in the fit function. So, we can define one or more callbacks in the callbacks list and then pass it callbacks list in the callbacks argument of the fit function. Here we use EarlyStopping and TensorBoard callback. EarlyStopping callback has patience of 2; that means, it will wait for 2 epochs for validation performance to improve and it monitors validation loss. So, if validation loss does not improve for 2 epochs, it will stop the training. And, this particular callback writes the TensorBoard logs to the logs directory.

(Refer Slide Time: 13:18)



You can look at the content of the directory to check if TensorBoard logs have indeed been written. Let us examine the content of logs directory. You should see that there are train and validation folders. If you look in the training folder, you will see that the events are already logged in and these events can be used and visualized through TensorBoard.

(Refer Slide Time: 14:08)



There are situations where a TensorFlow model is training for too long or after training the model we want to share the model with other collaborators in such cases we need to have functionality for saving and restoring the model.

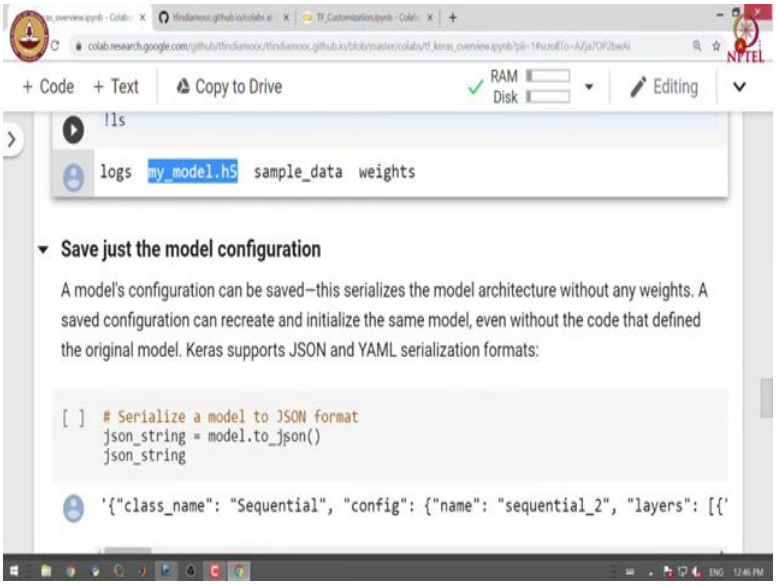
TensorFlow provides support for saving and restoring the models. So, model can be saved in one of the following forms. You may choose to only save the weights of the model; in that case you will be responsible for building the same architecture of the model and then using the weights to initialize the model. Or you may choose to only store the architecture of the model. Or you may choose to store both the architecture and weights of the model.

In addition to that you can save the model in different formats. You can save the model in JSON or in YAML files or in some other formats like H5. Let us train the model and save the weights of the model in the weights directory and after saving the model we can restore the state of the model by simply calling load_weight function and specifying the file where the weights are stored.

The weights are saved using save_weights method and the weights are stored in weights directory with the file name my_model. Let us check the content of the weights directory. There is a weights directory present and in weights directory you can see my_model and there are two files that are present which are produce storing the weights of my model.

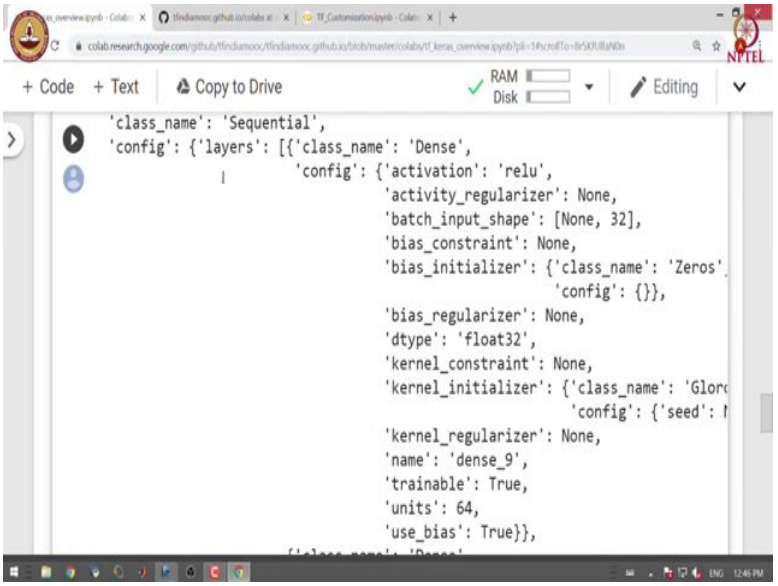
By default the models weights are stored in TensorFlow checkpoint format. Weights can also be saved to HDF5 format that is the default for multi backend implementation of Keras. Now, you can see that since we requested TensorFlow to save weights in HDF5 format, you can see a file with extends with the file with extension h5 over here.

(Refer Slide Time: 17:33)



You can also simply save the model configuration in JSON format you say model.to_json() and we get a json string.

(Refer Slide Time: 17:46)

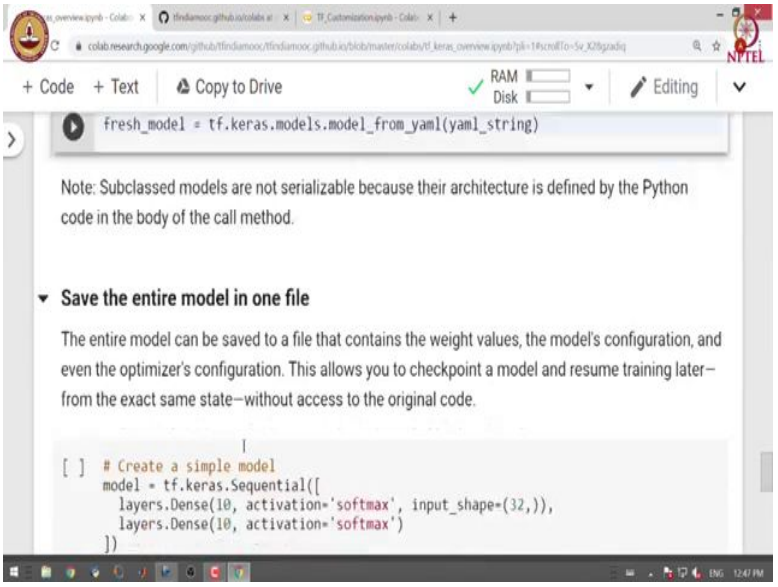


We can pretty print this json string and you can see the backend is tensorflow, the class name is Sequential and we can see the configuration of layers we can see that there are there is a

dense layer with batch input shape, data type, number of units and so on. Also, it is towards the keras version in which the model is defined.

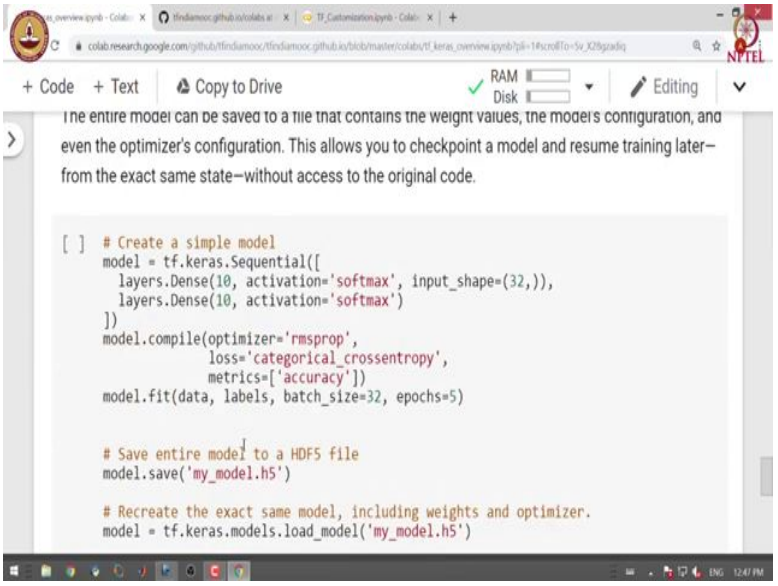
We can recreate the model from the JSON file by simply using models.model_from_json method. We can serialize the model to YAML format simply using model.to_yaml method. We can recreate the model from this YAML format.

(Refer Slide Time: 18:56)



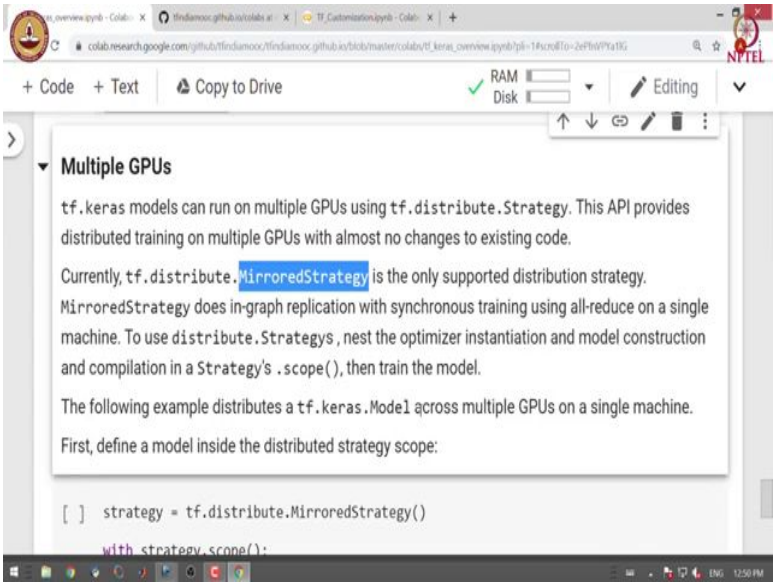
You can save the entire model to a file.

(Refer Slide Time: 19:00)



It will store both architecture and weights to the file. In order to save everything you simply call the save function on model and that will save the entire model in HDF5 format.

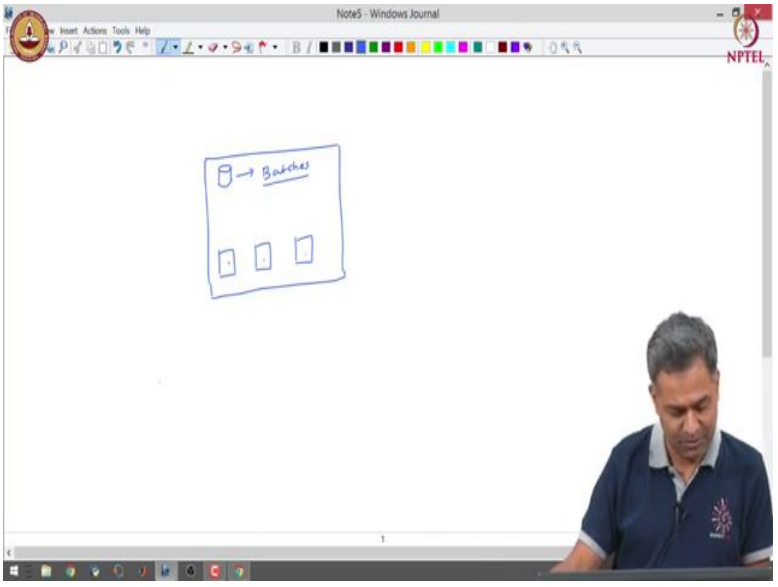
(Refer Slide Time: 20:13)



You are often required to train TensorFlow models on large data in practice. The models that we define are also complex in terms of number of parameters. In order to train this complex models faster, we use hardware accelerators like GPUs and TPUs.

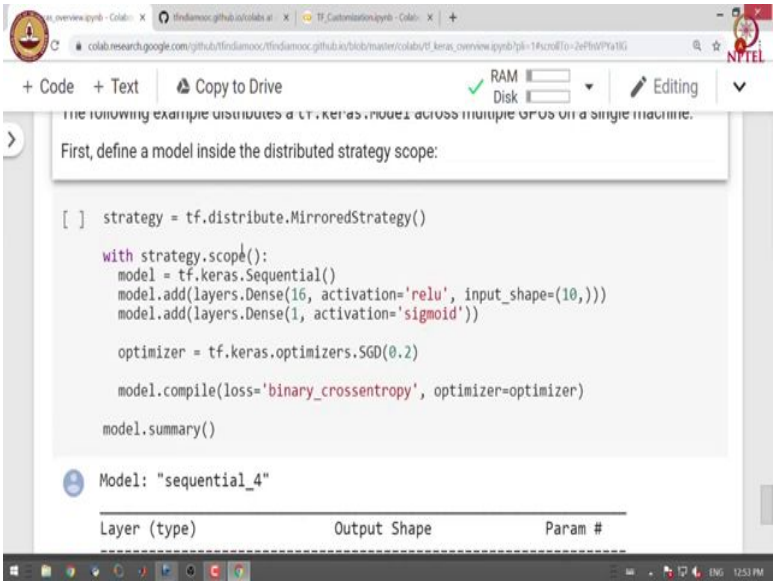
We can use multiple GPUs for training TensorFlow models. For that we define a distribution strategy. So, one of the distribution strategy is MirroredStrategy that is supported by TensorFlow and we construct the model in the context of this particular strategy. Let us concretely look at what is happening here.

(Refer Slide Time: 21:20)



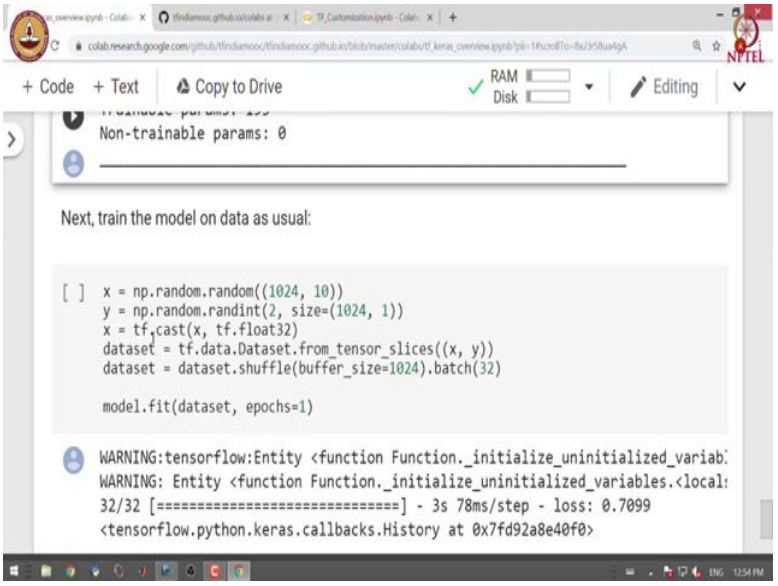
Say in MirroredStrategy we have a single machine with multiple GPUs, and we need to use all the GPUs to train model on a large dataset. So, for this particular scenario, we use MirroredStrategy. MirroredStrategy is a synchronous training strategy.

(Refer Slide Time: 21:49)



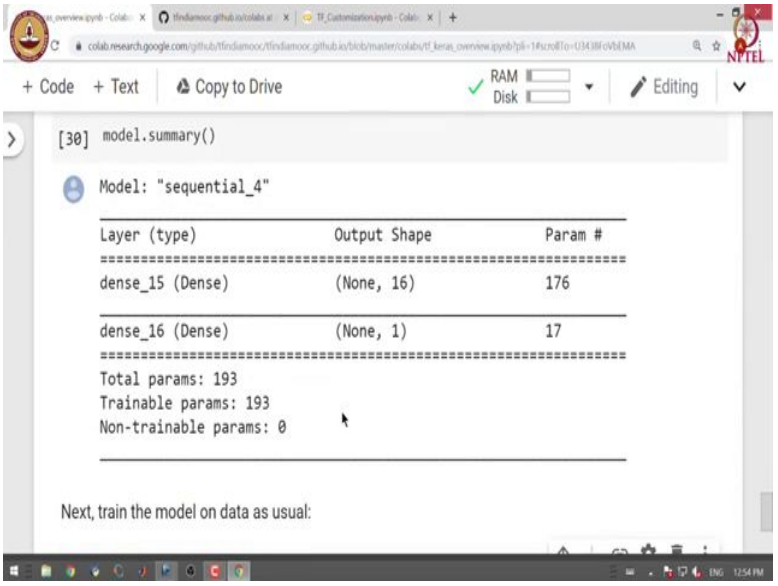
In order to use mirrored strategy what we do is in the scope of the strategy to be a MirroredStrategy and in the scope of the strategy we define our model and specify the optimizer.

(Refer Slide Time: 22:10)



And, then we train the model on that data as usual.

(Refer Slide Time: 22:21)



So, when model.fit is called internally the data is copied on multiple GPUs and training is carried out on multiple GPUs and then the output is combined from multiple GPUs to come up with the weight update. This is how we can train machine learning model in a multi GPU setting.

In the next class, we will be studying more details of this distributed training strategies. So, in this session we reviewed on the main concepts behind Keras; we looked at how to define the model in Keras; how to specify optimizers; how to save and restore the model; how to use callbacks and how to use distributed training for Keras model. In the next session, what we will do is, we will study how to customize the Keras models using functional APIs and by writing models and layers by subclassing.