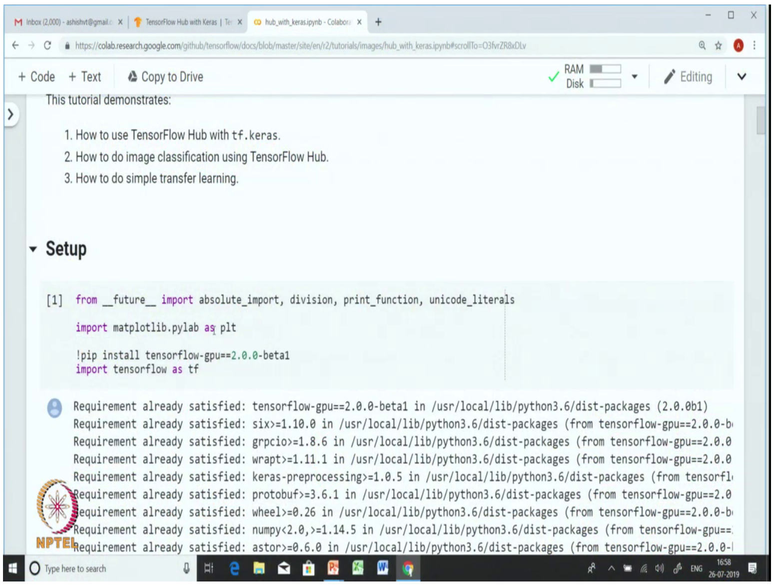
Practical Machine Learning
Dr. Ashish Tendulkar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 23
Transfer Learning with Tensorflow_hub

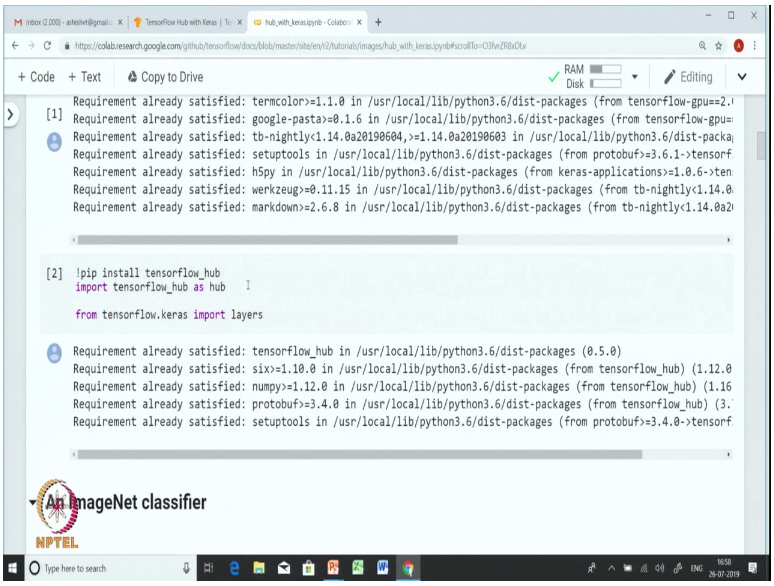
[FL]. In the previous session, we learnt how to use transfer learning in the context of convolutional neural network. In this session we will use tf.hub which is a way to share pre trained modeled components with the community. We will use pre trained models from tf.hub and use them for feature extraction as well as fine tuning. In this session we will learn how to use Tensorflow_hub with tf.keras API, how to do image classification using Tensorflow_hub and how to do a simple transfer learning.

(Refer Slide Time: 01:09)



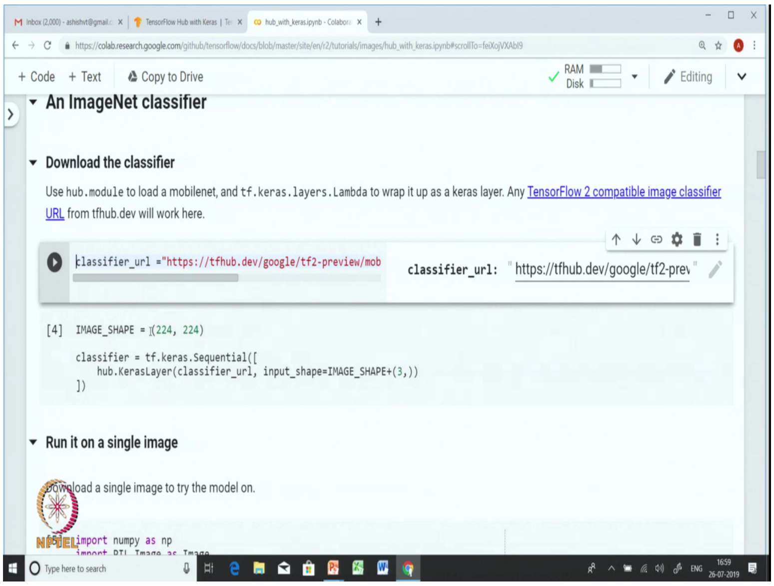
First we will import all the necessary libraries install TensorFlow 2.0 and import tensorflow. Note that we are installing tensorflow-gpu since we are training a CNN on images which runs faster on gpu we are using gpu as a hardware accelerator for this colab.

(Refer Slide Time: 01:36)



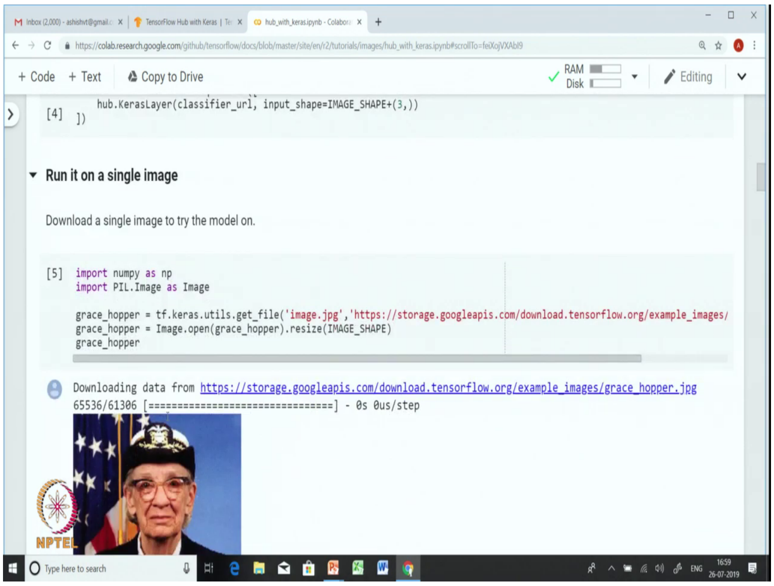
For tensorflow hub we import tensorflow_hub library and also import layers from the tf.keras library.

(Refer Slide Time: 01:52)



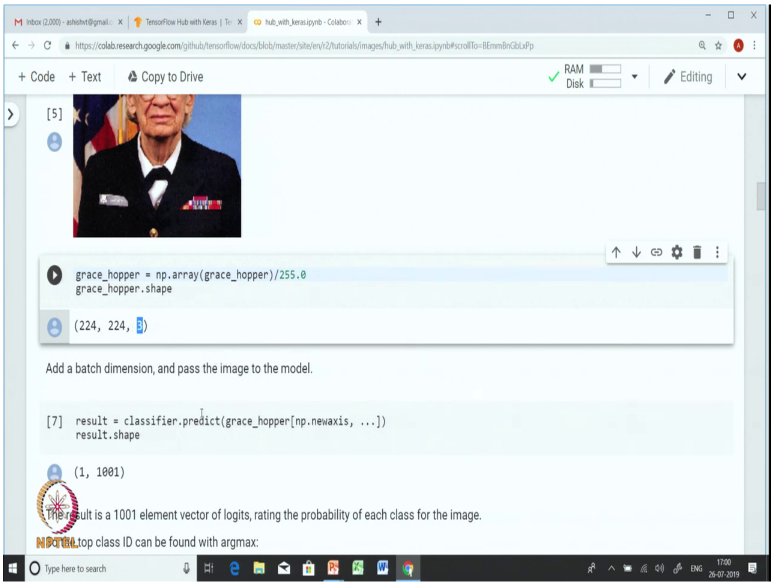
Next we download the classifier from tf.hub. We use hub.module to load a MobileNet and tf.keras.layers.Lambda to wrap it using a keras layer. So, this is the url for the classifier the MobileNet on tensorflow_hub. We define the shape of the image which 224 x 224 and we define a sequential model with the hub layer.

(Refer Slide Time: 02:33)



If we run this particular model on a single image let us see what we get. We download the image using tf.keras.util.get_file() and resize the image by the image shape.

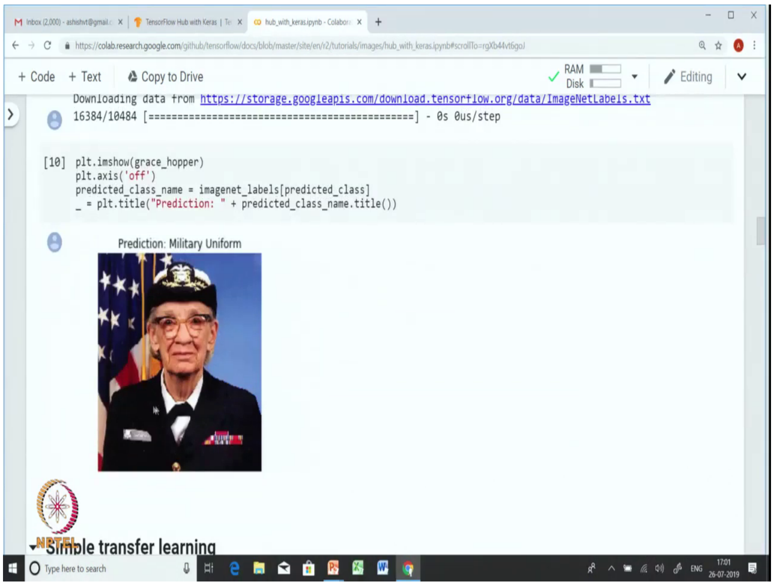
(Refer Slide Time: 02:55)



The input image is a colored image with three channels and has height and width of 224 each. We know that CNS take 4D tensor, so, we add a batch dimension and pass the image to the model. So, the result of the classifier is a 2D tensor which has 1001 elements corresponding to logics rating the probability of each class for the image. The top class ID can be found using argmax(). So, you can see that the class ID for the input image is 653.

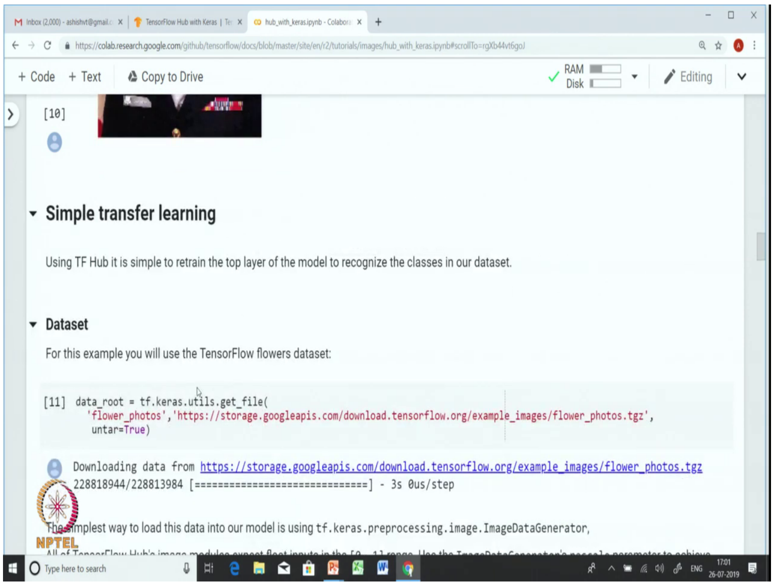
In order to get the text representation of the class we download the ImageNet labels file and use it to decode the name of the class.

(Refer Slide Time: 04:06)



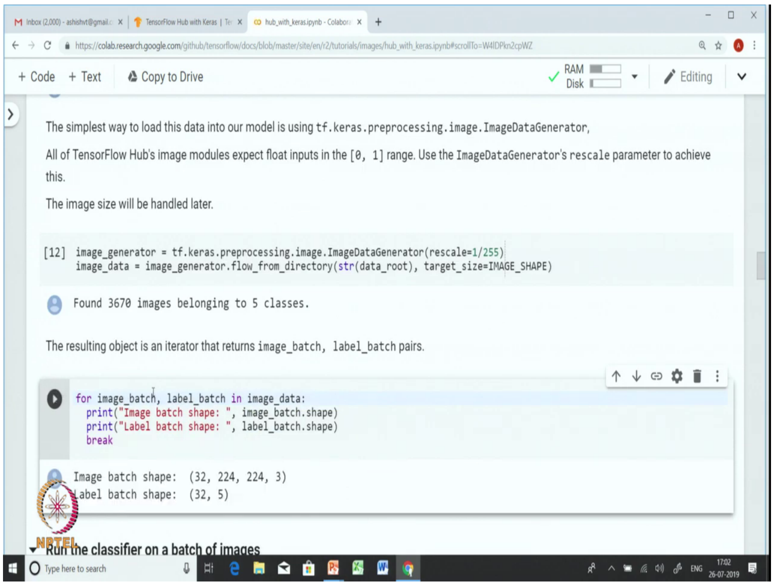
So, you can see that the prediction which was ID 653 corresponds to Military Uniform. We can use tf.hub to retrain the top layer of the model to recognize the classes in a dataset.

(Refer Slide Time: 04:27)



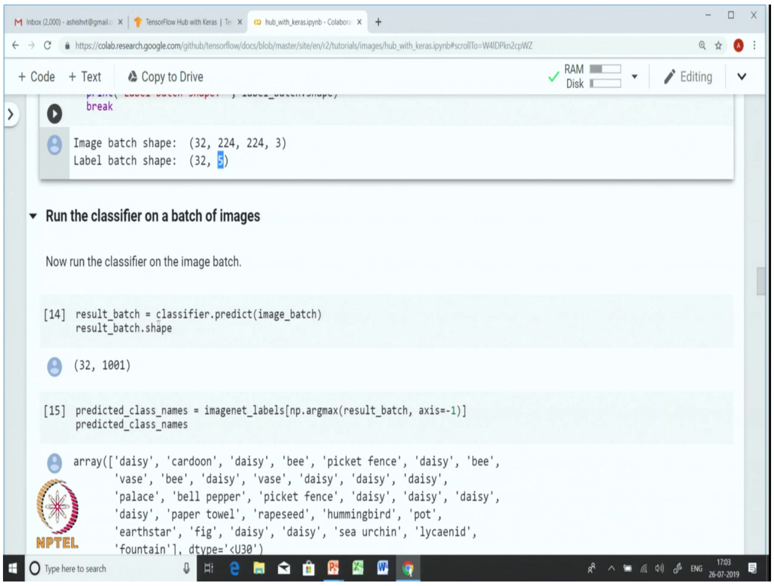
Let us download a flower dataset and demonstrate transfer learning with tf.hub. We load the data into our model using mage data generator which you can see here and we pass the rescaling parameter to it. The TensorFlow Hub's image modules expect a float input between 0 to 1, hence we rescale the input image. We also resize the image to the desired shape.

(Refer Slide Time: 05:06)



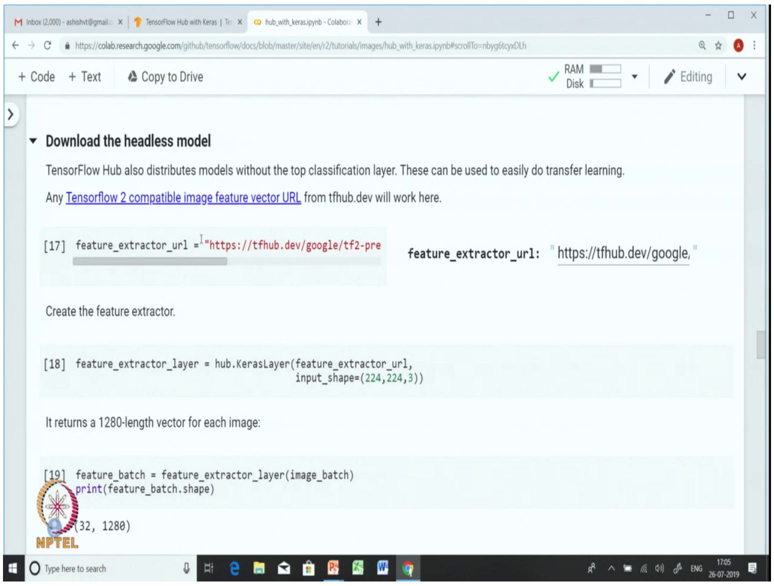
We can look at the shape of image batch and the label batch. The image batch is the 4D tensor each having 32 images with height and width of 224 and 3 channels. So, for each image we have a vector of size 5.

(Refer Slide Time: 05:35)



So, the flower dataset has 5 classes and each class is represented in one hot encoding. Let us run the classifier on the image batch. Note that currently the classifier only contains the keras layer from hub. If you apply the classifier on the image batch we get a 2D tensor of shape (32, 1001).

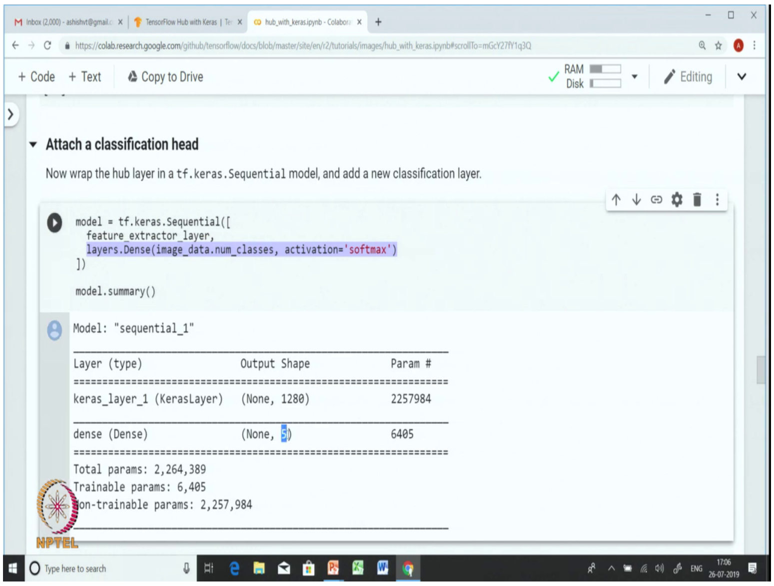
(Refer Slide Time: 06:26)



Tensorflow_hub distributes model without a top classification layer. This can be used for transfer learning.

So, we create a feature extractor as a keras layer with the input shape of 224 x 224 x 3 it returns a 1280 length vector. The feature batch is a 2D tensor. So, for every image we have 1280 length vector.

(Refer Slide Time: 07:08)

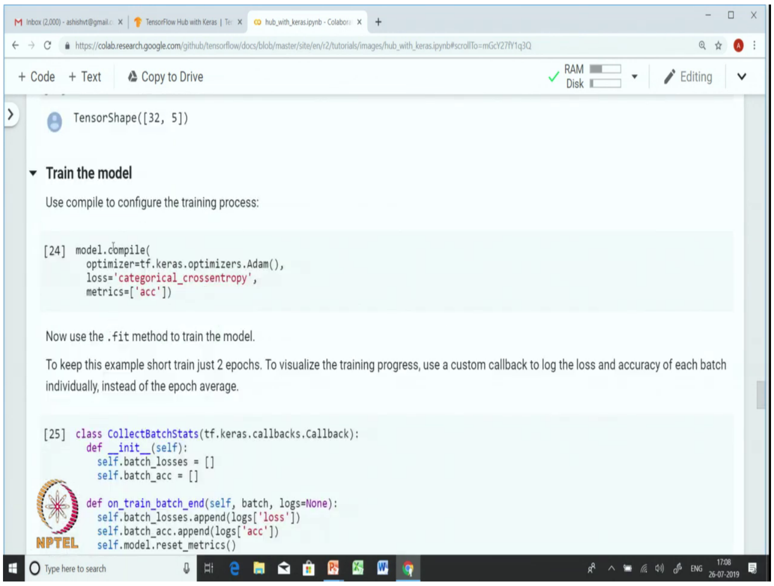


You freeze the variable in the feature extraction layer, so that the training only modifies the new classifier layer. So, we attach a new classifier layer to the model. The new classifier layer has units equal to the number of classes in the images and we use Softmax as an activation function. Since we have 5 different classes, the dense layer outputs 5 probabilities one corresponding to each class.

So, the number of parameter for keras layer is equal to the number of parameters in the MobileNet. MobileNet has 2.2 million parameters and the dense layer has 1280 inputs. So, for every unit we have this 1280 parameters corresponding to each of the input plus 1 bias. So, there are 1281 parameters per unit and we have 5 units making it to 6405 parameters.

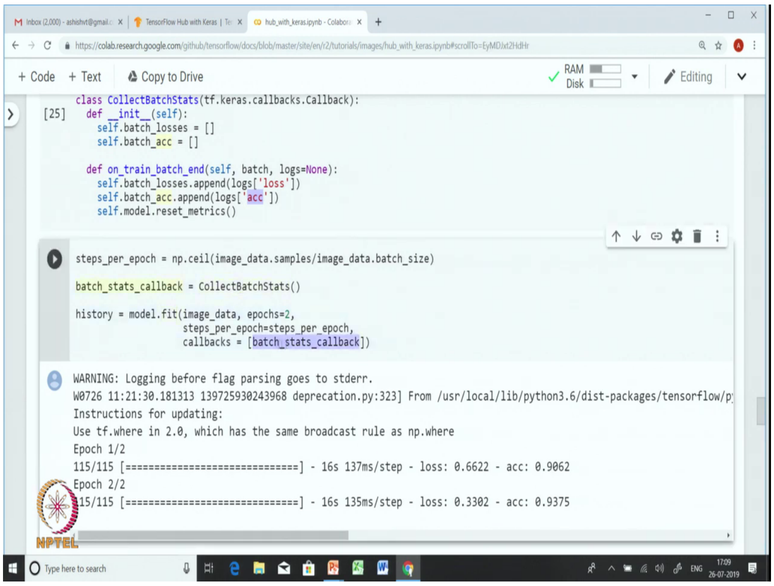
So, the total parameters are sum of the parameters in the keras layer and the parameters in the dense layer. Out of this total parameters the parameters in the keras layer are non-trainable were as the parameters in the dense layer are trainable.

(Refer Slide Time: 08:51)



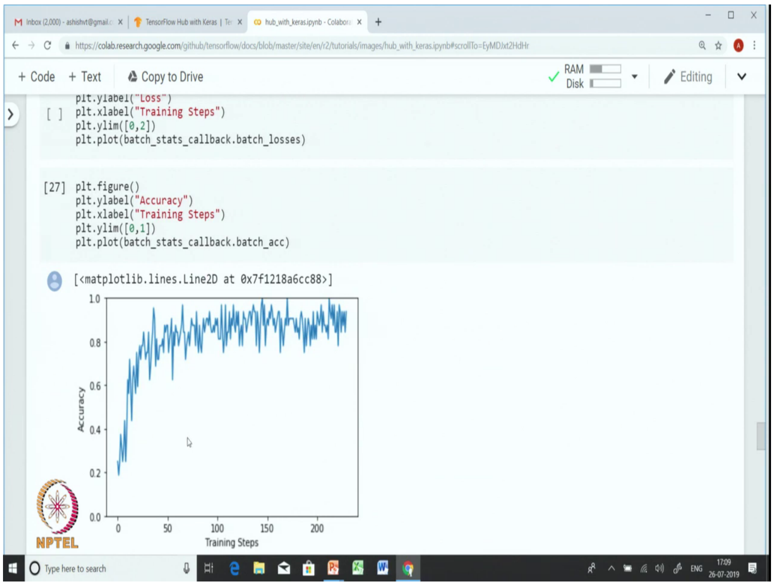
Let us compile the model. Since we have 5 classes, we use *categoricalcrossentropy* loss. We use *Adam* as an optimizer. Let us fit the model. We will fit the model just for 2 epochs.

(Refer Slide Time: 09:12)



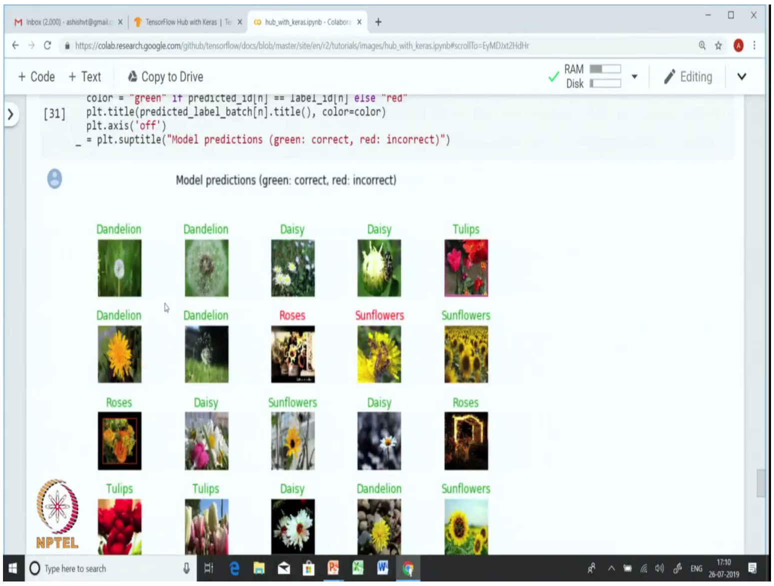
To visualize the training progress we use a custom call back to log the loss and the accuracy of each batch individually instead of epoch average. We also compute steps per epoch and define a CollectBatchStats() callback. We use the callback in the fit and the steps per epoch computed over here. You can see that after two steps we reached an accuracy close to 94%.

(Refer Slide Time: 09:55)



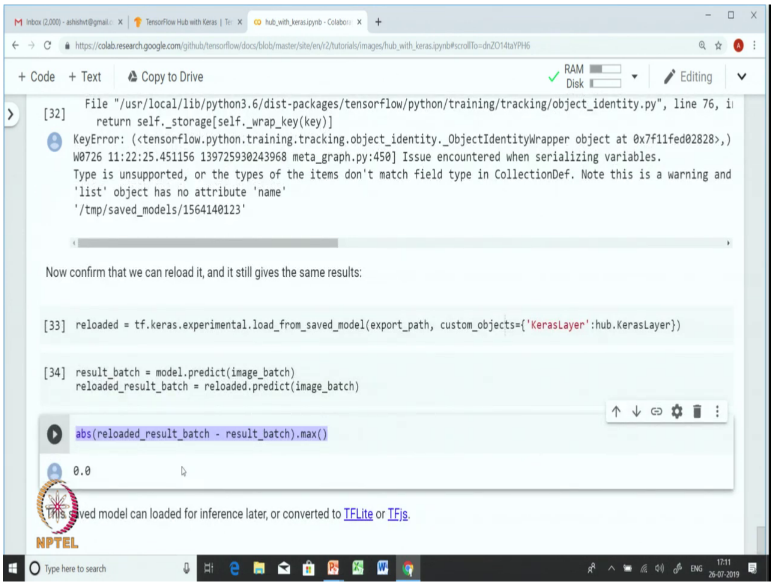
If you look at the training accuracy by the steps, we can see that it is increasing as we progress further in the training. Let us get the prediction for the image batch and plot the results.

(Refer Slide Time: 10:16)



If the model prediction is correct we use the green color and we use and we use red color if the predictions are incorrect. Now, you can see that most of the predictions are correct.

(Refer Slide Time: 10:48)



Now, that the model is trained you can export it as a saved model so that we can use it for deployment on some other device or we can also reload it for the future use. After saving the model we reload it and we check whether the results of the reloaded model and the earlier model matches that we do by taking the difference between the results.

So, in this session we looked at tf.hub and understood how to use the models saved in tf.hub for transfer learning on CNS.