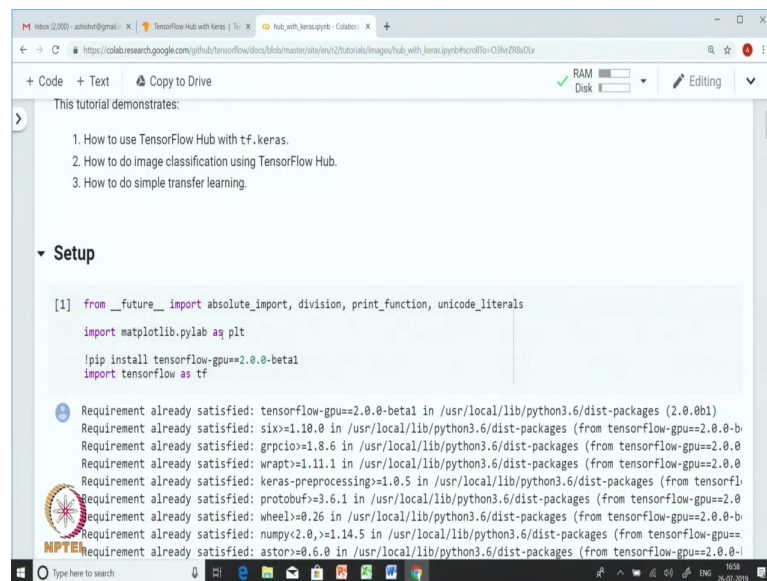


Practical Machine Learning
Dr. Ashish Tendulkar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

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)



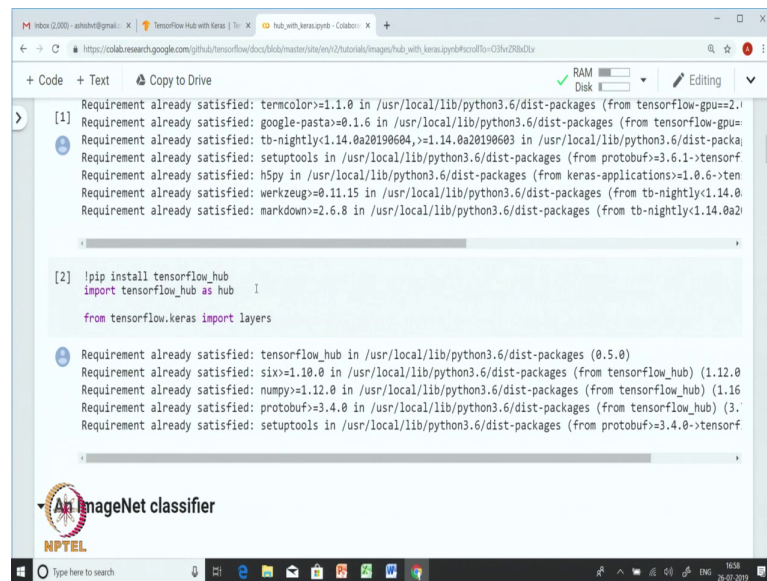
```
[1] from __future__ import absolute_import, division, print_function, unicode_literals
import matplotlib.pyplot as plt

!pip install tensorflow-gpu==2.0.0-beta1
import tensorflow as tf
```

Requirement already satisfied: tensorflow-gpu==2.0.0-beta1 in /usr/local/lib/python3.6/dist-packages (2.0.0b1)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-beta1) (1.10.0)
Requirement already satisfied: grpcio>=1.8.6 in /usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-beta1) (1.11.1)
Requirement already satisfied: wrapt>=1.11.1 in /usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-beta1) (1.11.1)
Requirement already satisfied: keras-preprocessing>=1.0.5 in /usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-beta1) (1.0.5)
Requirement already satisfied: protobuf>=3.6.1 in /usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-beta1) (3.6.1)
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-beta1) (0.26)
Requirement already satisfied: numpy<2.0,>=1.14.5 in /usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-beta1) (1.14.5)
Requirement already satisfied: astor>=0.6.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.0.0-beta1) (0.6.0)

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)



```
[1] Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.1.0)
Requirement already satisfied: google-pasta>=0.1.6 in /usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.1.0)
Requirement already satisfied: tb-nightly<1.14.0a20190604,>=1.14.0a20190603 in /usr/local/lib/python3.6/dist-packages (from tensorflow-gpu==2.1.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from protobuf>=3.6.1->tensorflow-gpu==2.1.0)
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages (from keras-applications>=1.0.6->tensorflow-gpu==2.1.0)
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-gpu==2.1.0)
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-gpu==2.1.0)

[2] !pip install tensorflow_hub
import tensorflow_hub as hub

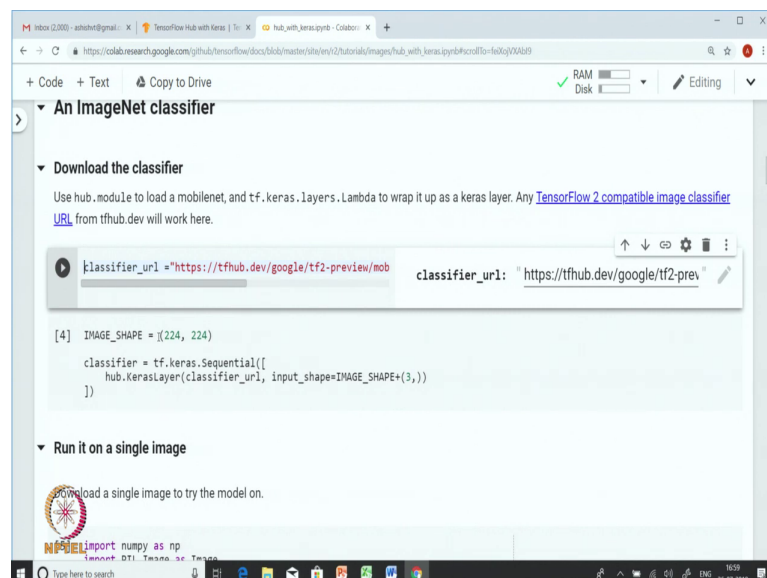
from tensorflow.keras import layers
```

Requirement already satisfied: tensorflow_hub in /usr/local/lib/python3.6/dist-packages (0.5.0)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow_hub) (1.12.0)
Requirement already satisfied: numpy>=1.12.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow_hub) (1.16.0)
Requirement already satisfied: protobuf>=3.4.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow_hub) (3.6.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from protobuf>=3.4.0->tensorflow_hub) (2.8.5)

An ImageNet classifier

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

(Refer Slide Time: 01:52)



```
An ImageNet classifier

Download the classifier
Use hub.module to load a mobilenet, and tf.keras.layers.Lambda to wrap it up as a keras layer. Any TensorFlow 2 compatible image classifier URL from tfhub.dev will work here.

classifier_url = "https://tfhub.dev/google/tf2-preview/mobilenet_v2_128_320/1"
classifier_url1 = "https://tfhub.dev/google/tf2-preview/mobilenet_v2_128_320/1"

[4] IMAGE_SHAPE = (224, 224)

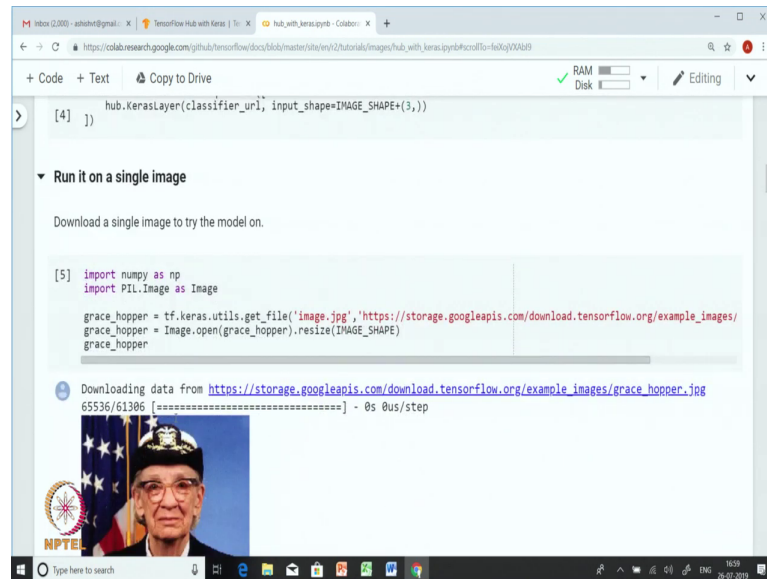
classifier = tf.keras.Sequential([
    hub.KerasLayer(classifier_url1, input_shape=IMAGE_SHAPE+(3,))
])

Run it on a single image
Download a single image to try the model on.
```

Import numpy as np

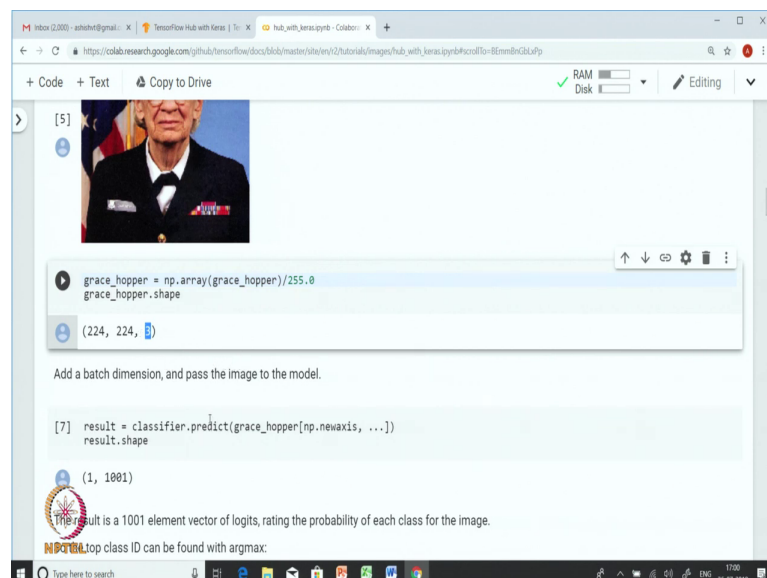
Next we download the classifier from tf.hub. We use hub.module to load a MobileNet and tf.keras.layers.Lambda to wrap it using a keras layer. So, this is the url for the classifier the MobileNet on tensorflow_hub. We define the shape of the image which 224 x 224 and we define a sequential model with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

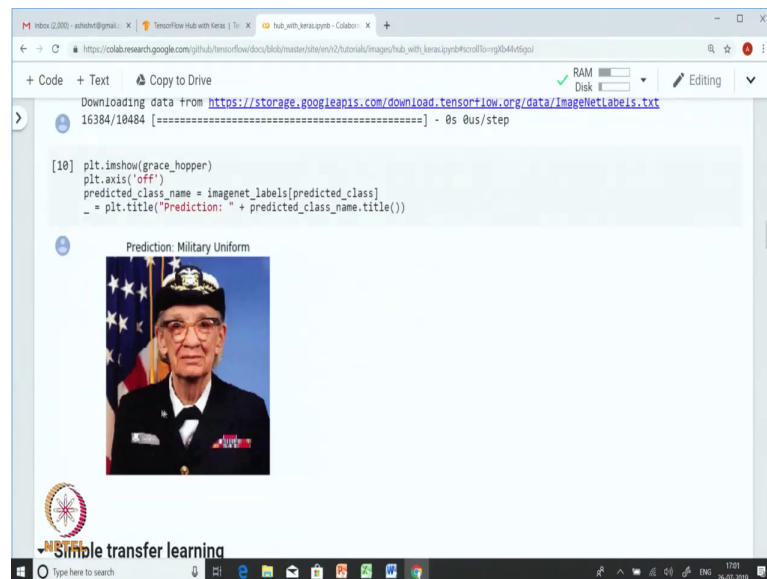


The input image is a colored image with three channels and has height and width of 224 each. We know that CNN take 4D tensor, so, we add a batch dimension and pass the image to the model. So, the result of the classifier is a 2D tensor which has 1001 elements corresponding to logits rating the probability 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.

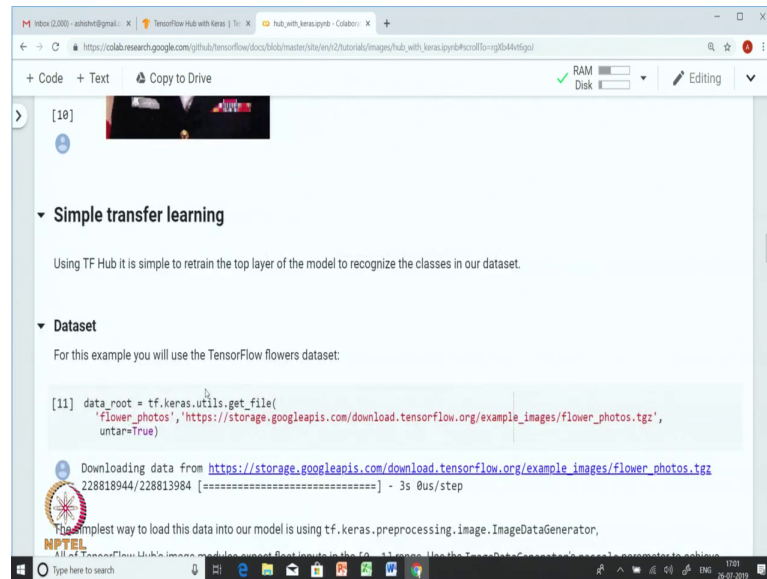
In order to get the text representation of the class we download the ImageNet labels file and use it to decode the name of the class.

(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)



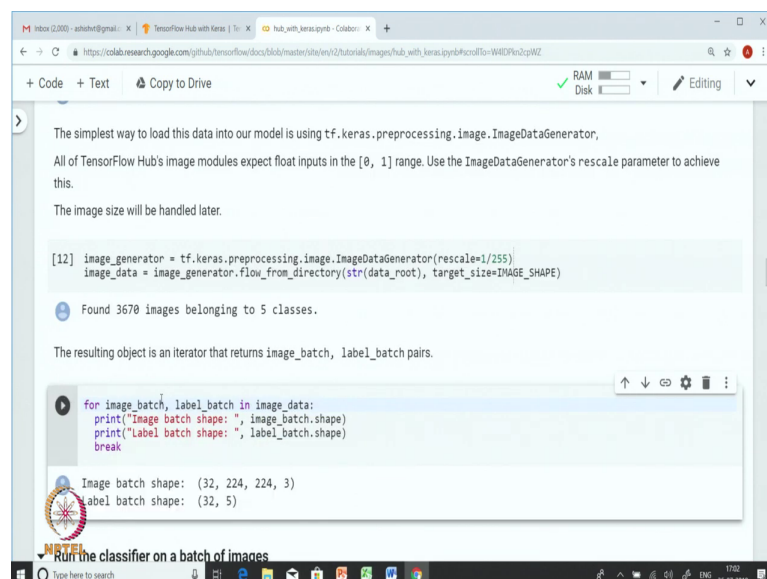
The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/hub_with_keras.py#scrollTo=rg6k4v5gsl. The notebook content includes a section titled "Simple transfer learning" with the text: "Using TF Hub it is simple to retrain the top layer of the model to recognize the classes in our dataset." Below this is a section titled "Dataset" with the text: "For this example you will use the TensorFlow flowers dataset:". A code cell [11] contains the following Python code:

```
data_root = tf.keras.utils.get_file('flower_photos', 'https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz', untar=True)
```

 Below the code, a progress bar indicates the download status: "Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz 228818944/228813984 [=====] - 3s 0us/step". A message below the code states: "The simplest way to load this data into our model is using tf.keras.preprocessing.image.ImageDataGenerator, All of TensorFlow Hub's image modules expect float inputs in the [0, 1] range. Use the ImageDataGenerator's rescale parameter to achieve this." The bottom of the notebook shows a taskbar with various application icons and a system clock displaying 11:01 on 26-07-2019.

Let us download a flower dataset and demonstrate transfer learning with tf.hub. We load the data into our model using image 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)



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/hub_with_keras.py#scrollTo=VWADPkn2pMZ. The notebook content includes the text: "The simplest way to load this data into our model is using tf.keras.preprocessing.image.ImageDataGenerator, All of TensorFlow Hub's image modules expect float inputs in the [0, 1] range. Use the ImageDataGenerator's rescale parameter to achieve this. The image size will be handled later." A code cell [12] contains the following Python code:

```
image_generator = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1/255) image_data = image_generator.flow_from_directory(str(data_root), target_size=IMAGE_SHAPE)
```

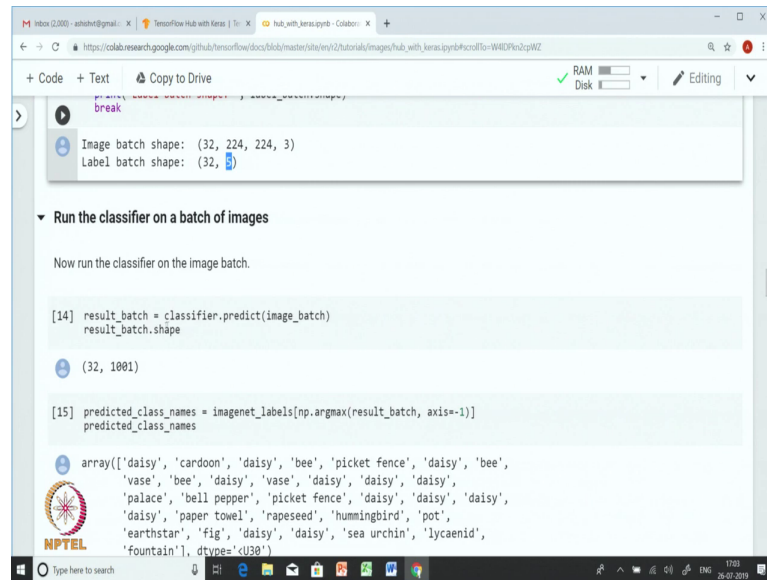
 Below the code, a message states: "Found 3670 images belonging to 5 classes." Another message states: "The resulting object is an iterator that returns image_batch, label_batch pairs." A code cell below this contains the following Python code:

```
for image_batch, label_batch in image_data: print("Image batch shape: ", image_batch.shape) print("Label batch shape: ", label_batch.shape) break
```

 Below the code, a message displays the output: "Image batch shape: (32, 224, 224, 3) label batch shape: (32, 5)". The bottom of the notebook shows a taskbar with various application icons and a system clock displaying 11:02 on 26-07-2019.

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)



```
break
Image batch shape: (32, 224, 224, 3)
Label batch shape: (32, 5)

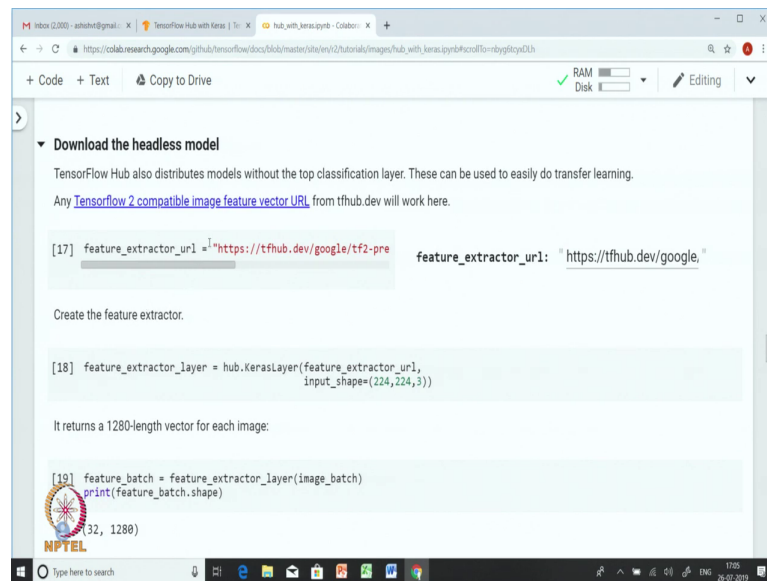
▼ Run the classifier on a batch of images
Now run the classifier on the image batch.

[14] result_batch = classifier.predict(image_batch)
result_batch.shape
(32, 1001)

[15] predicted_class_names = imagenet_labels[np.argmax(result_batch, axis=-1)]
predicted_class_names
array(['daisy', 'cardoon', 'daisy', 'bee', 'picket fence', 'daisy', 'bee',
      'vase', 'bee', 'daisy', 'vase', 'daisy', 'daisy', 'daisy',
      'palace', 'bell pepper', 'picket fence', 'daisy', 'daisy', 'daisy',
      'daisy', 'paper towel', 'rapeseed', 'hummingbird', 'pot',
      'earthstar', 'fig', 'daisy', 'daisy', 'sea urchin', 'lycaenid',
      'fountain'], dtype='<U30')
```

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)



The screenshot shows a Google Colab notebook with the following content:

- Download the headless model**
TensorFlow Hub also distributes models without the top classification layer. These can be used to easily do transfer learning.
Any [TensorFlow 2 compatible image feature vector URL](#) from tfhub.dev will work here.
- Code cell [17]:

```
feature_extractor_url = "https://tfhub.dev/google/tf2-pre"
feature_extractor_url = "https://tfhub.dev/google,"
```
- Text: "Create the feature extractor."
- Code cell [18]:

```
feature_extractor_layer = hub.KerasLayer(feature_extractor_url,
                                         input_shape=(224,224,3))
```
- Text: "It returns a 1280-length vector for each image:"
- Code cell [19]:

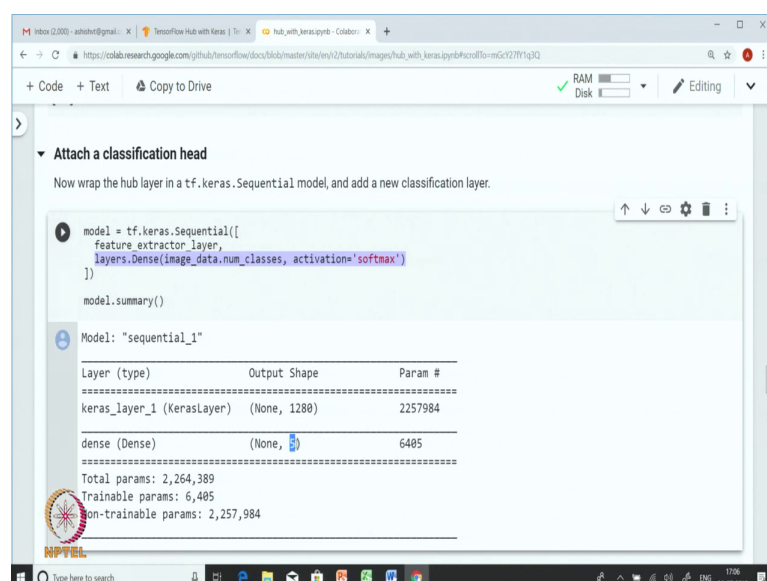
```
feature_batch = feature_extractor_layer(image_batch)
print(feature_batch.shape)
```
- Output of [19]:

```
(32, 1280)
```

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)



The screenshot shows a Google Colab notebook with the following content:

- Attach a classification head**
Now wrap the hub layer in a tf.keras.Sequential model, and add a new classification layer.
- Code cell:

```
model = tf.keras.Sequential([
    feature_extractor_layer,
    layers.Dense(image_data.num_classes, activation='softmax')
])
model.summary()
```
- Model summary output:

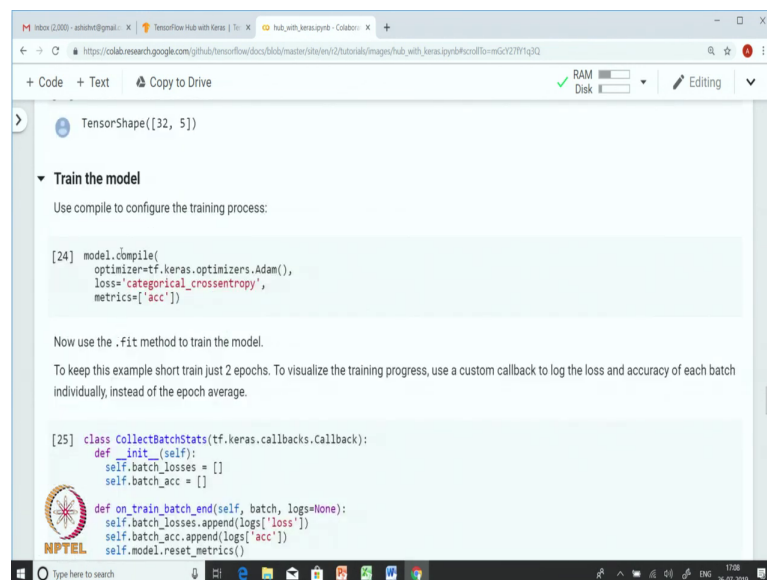
```
Model: "sequential_1"
Layer (type)                Output Shape                Param #
-----
keras_layer_1 (KerasLayer)   (None, 1280)                2257984
dense (Dense)                (None, 1000)                6405
Total params: 2,264,389
Trainable params: 6,405
Non-trainable params: 2,257,984
```

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)



```
TensorShape([32, 5])

▼ Train the model
Use compile to configure the training process:

[24] model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss='categorical_crossentropy',
    metrics=['acc'])

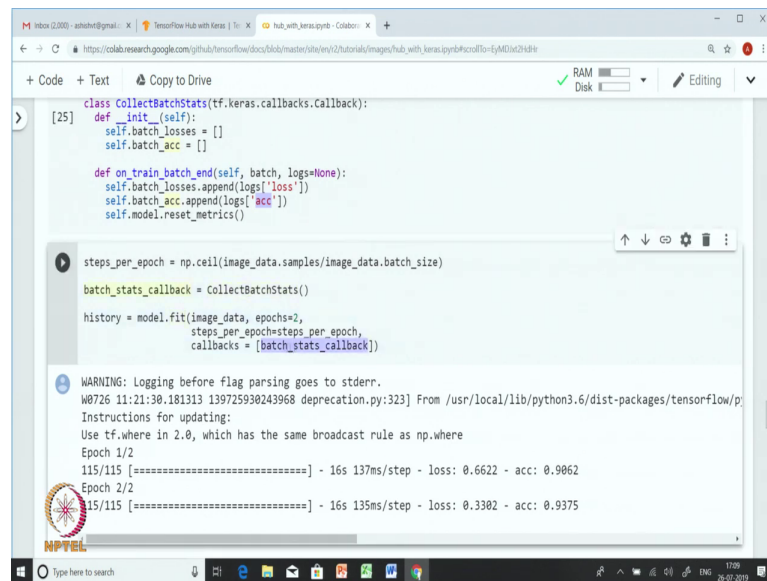
Now use the .fit method to train the model.
To keep this example short train just 2 epochs. To visualize the training progress, use a custom callback to log the loss and accuracy of each batch individually, instead of the epoch average.

[25] class CollectBatchStats(tf.keras.callbacks.Callback):
    def __init__(self):
        self.batch_losses = []
        self.batch_acc = []

    def on_train_batch_end(self, batch, logs=None):
        self.batch_losses.append(logs['loss'])
        self.batch_acc.append(logs['acc'])
        self.model.reset_metrics()
```

Let us compile the model. Since we have 5 classes, we use *categorical_crossentropy* 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)



```
class CollectBatchStats(tf.keras.callbacks.Callback):
    def __init__(self):
        self.batch_losses = []
        self.batch_acc = []

    def on_train_batch_end(self, batch, logs=None):
        self.batch_losses.append(logs['loss'])
        self.batch_acc.append(logs['acc'])
        self.model.reset_metrics()

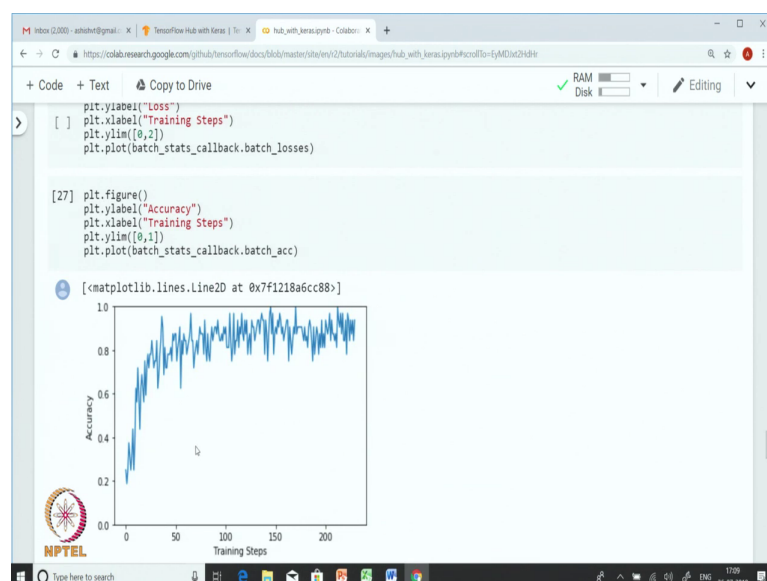
steps_per_epoch = np.ceil(image_data.samples/image_data.batch_size)
batch_stats_callback = CollectBatchStats()
history = model.fit(image_data, epochs=2,
                    steps_per_epoch=steps_per_epoch,
                    callbacks=[batch_stats_callback])
```

WARNING: Logging before flag parsing goes to stderr.
W0726 11:21:30.181313 139725930243968 deprecation.py:323] From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_ops.py:306: instructions_for_updating (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version. Use tf.where in 2.0, which has the same broadcast rule as np.where

Epoch 1/2
115/115 [=====] - 16s 137ms/step - loss: 0.6622 - acc: 0.9062
Epoch 2/2
115/115 [=====] - 16s 135ms/step - loss: 0.3302 - acc: 0.9375

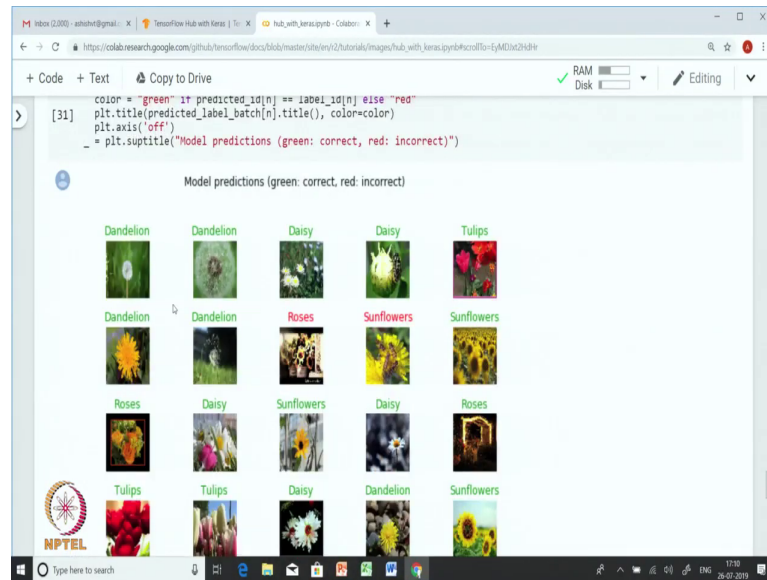
To visualize the training progress we use a custom call back to log the loss and the accuracy of each batch individually instead of epoch average. We also compute steps per epoch and define a CollectBatchStats() callback. We use the callback in the fit and the steps per epoch computed over here. You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

(Refer Slide Time: 09:55)



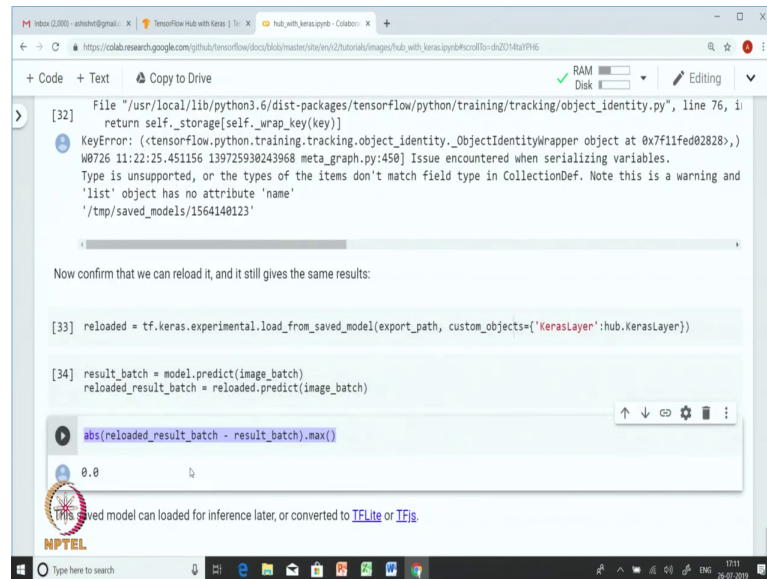
If you look at the training accuracy by the steps, we can see that it is increasing as we progress further in the training. Let us get the prediction for the image batch and plot the results.

(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)



The screenshot shows a Google Colab notebook interface. At the top, there are tabs for 'Index (2/2020)', 'TensorFlow Hub with Keras', and 'hub_with_keras.pyth - Colab'. The address bar shows a URL from colab.research.google.com. The code editor displays a file path and a `KeyError` message: `KeyError: (<tensorflow.python.training.tracking.object_identity.ObjectIdentityWrapper object at 0x7f11fed02828>,) W0726 11:22:25.451156 139725930243968 meta_graph.py:450] Issue encountered when serializing variables. Type is unsupported, or the types of the items don't match field type in CollectionDef. Note this is a warning and 'list' object has no attribute 'name' '/tmp/saved_models/1564140123'`. Below the error, a text prompt says 'Now confirm that we can reload it, and it still gives the same results:'. The code continues with `[33] reloaded = tf.keras.experimental.load_from_saved_model(export_path, custom_objects={'KerasLayer':hub.KerasLayer})` and `[34] result_batch = model.predict(image_batch); reloaded_result_batch = reloaded.predict(image_batch)`. A cell below shows the calculation `abs(reloaded_result_batch - result_batch).max()` with a play button icon. The output of this cell is `0.0`. At the bottom, there is a message: 'This saved model can be loaded for inference later, or converted to [TF Lite](#) or [TF.js](#).' and an 'MPTEL' logo.

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.