# Introduction

In this chapter we are going to use all the major topics we have gone through in this book. Before proceeding to this chapter I kindly request you to please go through the previous chapters which will give you basic step by step understanding of the topics.

In the previous chapter we have seen what is Tensor-flow is and how to use this in creation of machine learning models and also how to use them in building your android apps which can make on mobile inference without any connection to any of the server.

And also, we have gone through the SciKit-Learn how to build effective models, how to test them and how to use them in IOS apps by converting them to Core-ML format.

Now we are about to put all together and making a small POC [Proof of concept] to demonstrate how we can leverage this skills and make better (Artificial Intelligence) AI enabled apps which runs on the small limited hardware of our mobiles.

# Learning Outcomes

In this chapter mostly we are going to use the previous chapter’s knowledge. Upon that, we will learn what TF-Core-ML is about? and how to use it to convert a tensorflow model to Core-ML model. And also how to create an image reorganization app in IOS which uses the Image classification model prepared in tensorflow.

# About POC [Proof of Concept]

The POC which we are undertaking here is to demonstrate how to create an image reorganization model using neural networks. And also how to make use of a model created through tensorflow in an IOS app.

This whole POC has been divided into the following tasks.

* Create Image classification model in Tensorflow
* Convert it to Core-ML format.
* Create a IOs app using the converted model.

## Creating Image Classification model in Tensorflow

### About Tensorflow

Tensorflow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google, often replacing its closed-source predecessor, DistBelief.

Tensorflow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open source license on November 9, 2015.

Tensorflow is cross-platform. It runs on nearly everything: GPUs and CPUs—including mobile and embedded platforms—and even tensor processing units (TPUs), which are specialized hardware to do tensor math on.

What does a tensorflow does?

To keep it simple. Just assume you want to add two numbers. Now if you want to write a program in a regular program languages like python.

a = 1

b = 2

print(a+b)

If you run the program you will see the output as 3.

Then see the same implementation on tensorflow.

import tensorflow as tf

x = tf.constant(35, name='x')

y = tf.Variable(x + 5, name='y')

model = tf.global\_variables\_initializer()

with tf.Session() as session:

session.run(model)

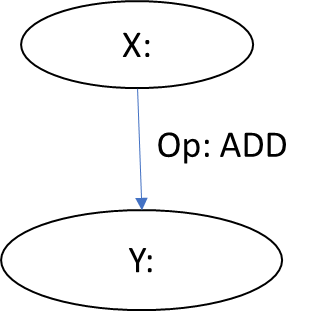
print(session.run(y))

Let me explain the above code. First we are creating an constant. With node name ‘x’ and add 5 to it and storing in another variable / node ‘y’.

If you see the output of console of y at this point you will find the definition of the node. But not the value as 40.

Because here you are defining the nodes of the graph and its corresponding operations. You can make use of graph once you initialize the variables and create and get session / instance of graph.

Please see the below representation to easily understand the concept.



In Tensorflow all the constants and placeholders, variables we will use to create the definition and the linkage between nodes which will create one graph.

Just like you Class concept in object oriented programing. Just assume graph as a class and nodes as data members.

And tf.globalvariableinitilizer() as just calling the static method to initialize the constants and variable. And session.run() as calling the constructor of a class.

That’s all a brief introduction about tensorflow.

Usually to create an image classifier we need to go through many things and do so much of coding. To keep it simple I am showing you how to create it using the Google provided code. The below content was taken from the google’s codelab tutorial.

This was made using CNN (convolutional neural networks). To explain it all this book is not sufficient. So, interested readers can go through some of the online resources available. One of such resource is <https://colah.github.io/posts/2014-07-Conv-Nets-Modular/>.

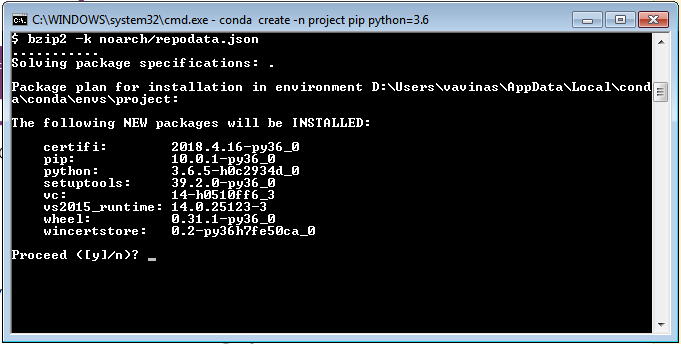
Let me show you how easily we can create an image classifier in tensorflow.

To get started in this. First we need to install anaconda into your machine.

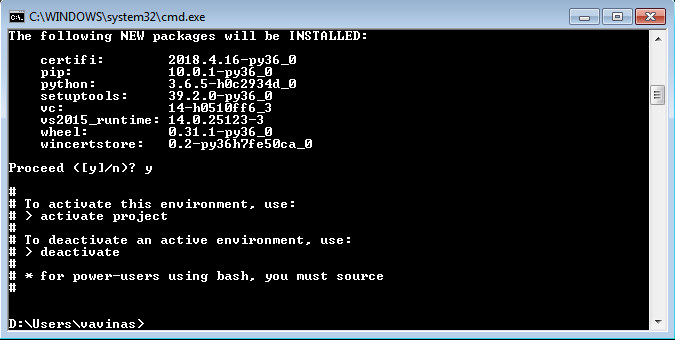
Then run the following commands.

**conda create -n tensorflow pip python=3.6**

Once you run the command it will prompt like this.



Then type y to proceed. Once the command got successfully executed. You will see the below screen.



Then type activate project.

Once the project got activated you will see the prompt like

**(project) D:\Users\vavinas>**

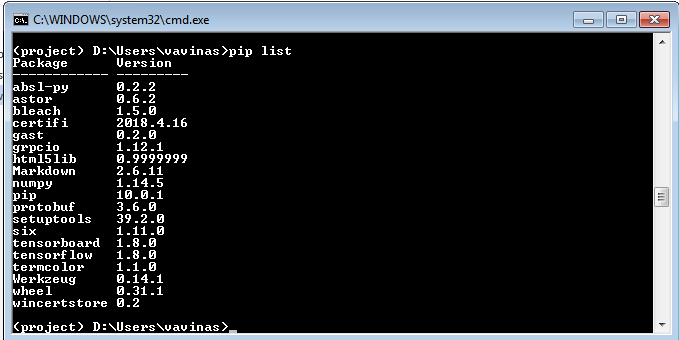
Then type the below commands.

**Pip install tensorflow**

Once it got installed successfully. You need to verify the installed packages. Using the below command.

**Pip list**

It has to produce the following result. If you didn’t see some of the Packages in your machine. Then reinstall them.

git

Now we have successfully installed tensorflow and its dependencies.

Now let us get the code from Google which will do the classification.

For this make sure you have installed **Git** in your machine. There are several ways to install it but the simple way is through npm.

To check the **Git** is properly installed. Type **git** in the opened command prompt. You will see all the options available for that command. If it is prompting as **invalid command** please try to install it correctly.

Now execute the command

**git clone** [**https://github.com/googlecodelabs/tensorflow-for-poets-2**](https://github.com/googlecodelabs/tensorflow-for-poets-2)

Once it got cloned. Go to the tensorflow-for-poets-2. BY executing below command.

**Cd tensorflow-for-poets-2**

This folder is having all the scripts required to train a model for image recognition.

In this if you see **tf\_file** folder. It is empty.

Here we are going to keep the training images and train the model using the scripts in the scripts folder.

To input the images you need to first download the images.

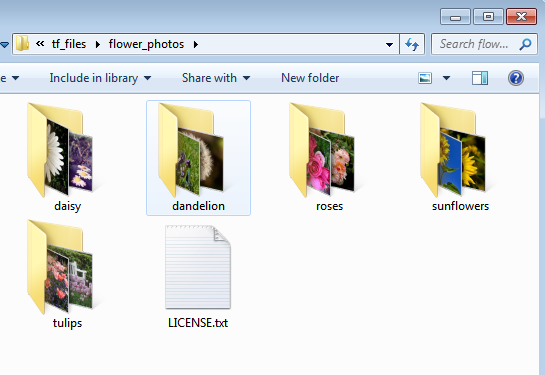
For sample we are using flowers images with 4 class labels.

To download that use the following command.

**curl http://download.tensorflow.org/example\_images/flower\_photos.tgz | tar xz -C tf\_files.**

If you are unable to execute the command open the folder in explorer. And download the file in the tensorflow-for-poets-2/tf\_files.

Extract the files in to flat files.



Now we are going to retrain the model using the following script provided by the tensorflow team. You need to run the following command

python -m scripts.retrain \

--bottleneck\_dir=tf\_files/bottlenecks \

--how\_many\_training\_steps=500 \

--model\_dir=tf\_files/models/ \

--summaries\_dir=tf\_files/training\_summaries/ mobilenet\_0.50\_224 \

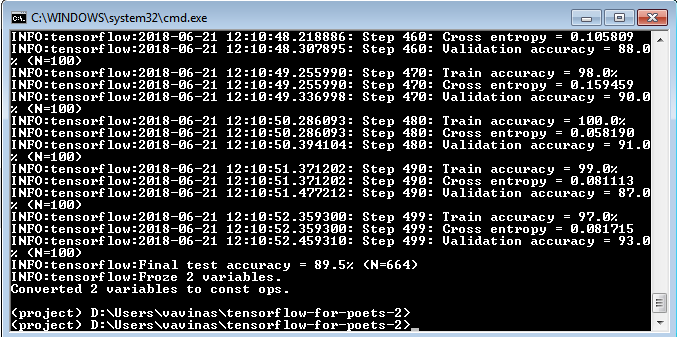
--output\_graph=tf\_files/retrained\_graph.pb \

--output\_labels=tf\_files/retrained\_labels.txt \

--architecture=mobilenet\_0.50\_224 \

--image\_dir=tf\_files/food\_photos

The script will support many attributes, but her using some of the important ones. Once you execute the above command you it will take a while to complete. Once it is completed you will see a similar screen.



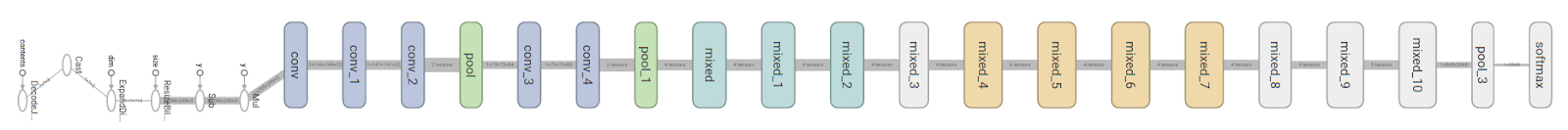
Let us know about the attributes and its values we are using.

Bottlenexk dir: In this we will give where to save the bottle next files.

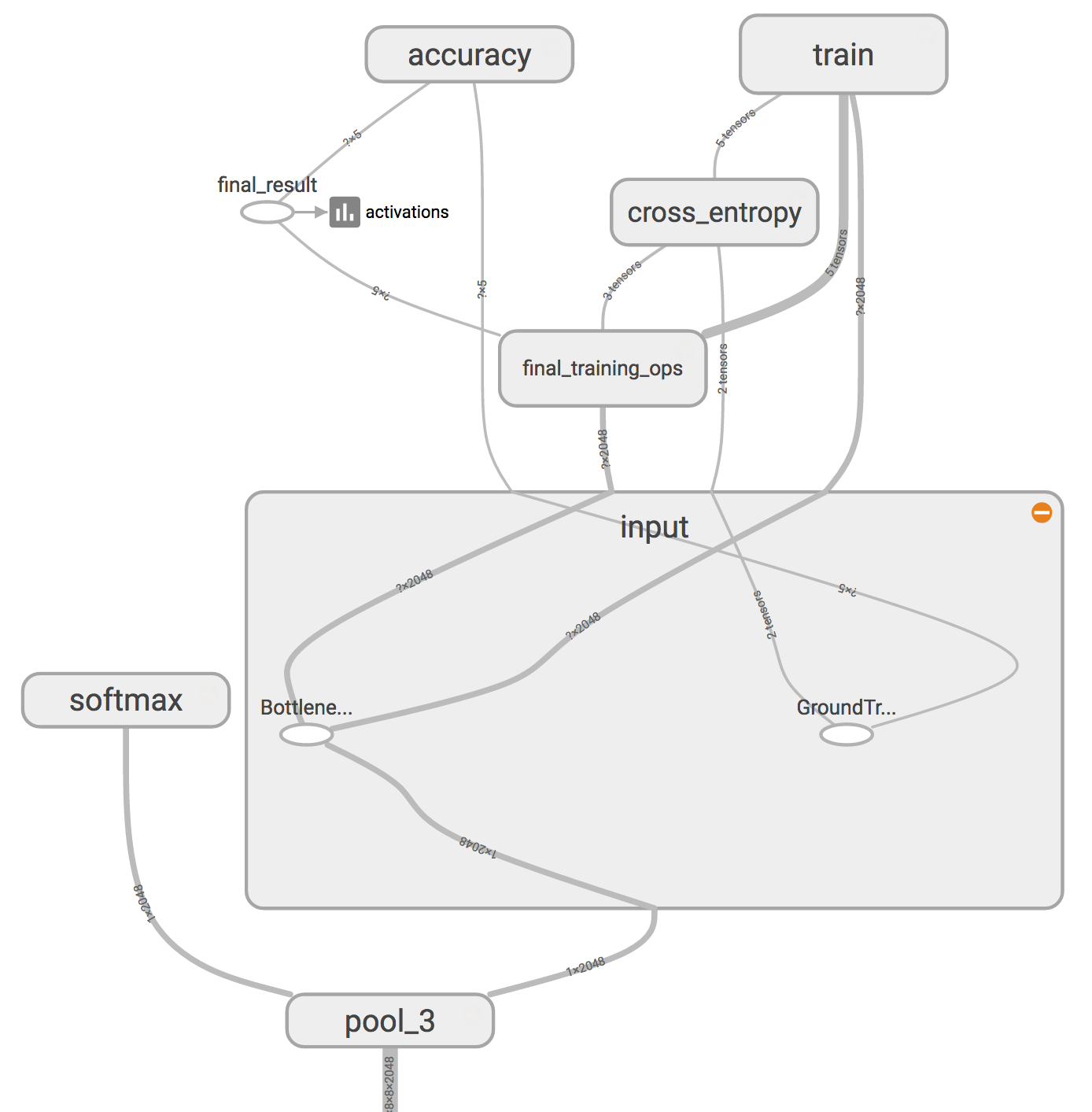
About Bottlenecks:

Now we will see some background on how this retraining process works.

The first phase analyzes all the images on disk and calculates the bottleneck values for each of them. What's a bottleneck?



These ImageNet models are made up of many layers stacked on top of each other, a simplified picture of Inception V3 from TensorBoard, is shown above (all the details are available in this paper, with a complete picture on page 6). These layers are pre-trained and are already very valuable at finding and summarizing information that will help classify most images. Here, you are training only the last layer (final\_training\_ops in the figure below). While all the previous layers retain their already-trained state.



In the above figure, the node labeled "softmax", on the left side, is the output layer of the original model. While all the nodes to the right of the "softmax" were added by the retraining script.

The above figure is a screenshot from tensorboard. You can open TensorBoard in your browser, to get a better look at it. You will find it in the "Graphs" tab.

Note that this will only work after the retrain script finished generating the "bottleneck" files.

A bottleneck is an informal term we often use for the layer just before the final output layer that actually does the classification. "Bottleneck" is not used to imply that the layer is slowing down the network. We use the term bottleneck because near the output, the representation is much more compact than in the main body of the network.

Every image is reused multiple times during training. Calculating the layers behind the bottleneck for each image takes a significant amount of time. Since these lower layers of the network are not being modified their outputs can be cached and reused.

So the script is running the constant part of the network, everything below the node labeled Bottleneck. above, and caching the results.

The command you ran saves these files to the bottlenecks/ directory. If you rerun the script, they'll be reused, so you don't have to wait for this part again.

**how\_many\_training\_steps**

This will a number below 4000. Greater number will give your greater accurate model and takes much time to build and also the model file also becomes large..

**model\_dir**

This tells where to save the model

**summaries\_dir**

It will have training summaries.

**output\_graph**

Where to save the output graph. This is the resultand model which will use in mobiles.

**output\_labels**

This is the file having the class labels. Usually the class label for an image is the folder name.

**architecture**

This tells which architecture to use. Here we are using mobilenet model wilth 0.50 elative size of model and 244 image size.

**image\_dir**

Input images directory. In this case flower photos.

Now you are having the tensorflow retrained model in your hand.

How to test it?

You can do it using the below command

python -m scripts.label\_image \

--graph=tf\_files/retrained\_graph.pb \

--image=tf\_files/flower\_photos/daisy/21652746\_cc379e0eea\_m.jpg

It will give you which class the flower image belongs to.

Now let us go to the next task, converting the tensorflow model to Core-ML format.

# About TF-Core-ML

Tensorflow team has developed a package which is used to convert the models created in Tensorflow into Core-ML which in through used in IOS apps.

In order to use this, you much install need a MAC operating system installed with python 3.6 and all the tensorflow.

Using this we can convert the tensorflow model file (.pb) to Core-Ml format (.mlmodel).

For this first you need to execute the following command.

**Pip install tfcoreml**

Once you install this.

Write the following code in your in one python file name it as **inspect.py** and save it.

import tensorflow as tf  
from tensorflow.core.framework import graph\_pb2  
import time  
import operator  
import sys  
  
def inspect(model\_pb, output\_txt\_file):  
 graph\_def = graph\_pb2.GraphDef()  
 with open(model\_pb, "rb") as f:  
 graph\_def.ParseFromString(f.read())  
  
 tf.import\_graph\_def(graph\_def)  
  
 sess = tf.Session()  
 OPS = sess.graph.get\_operations()  
  
 ops\_dict = {}  
  
 sys.stdout = open(output\_txt\_file, 'w')  
 for i, op in enumerate(OPS):  
 print('---------------------------------------------------------------------------------------------------------------------------------------------')  
 print("{}: op name = {}, op type = ( {} ), inputs = {}, outputs = {}".format(i, op.name, op.type, ", ".join([x.name for x in op.inputs]), ", ".join([x.name for x in op.outputs])))  
 print('@input shapes:')  
 for x in op.inputs:  
 print("name = {} : {}".format(x.name, x.get\_shape()))  
 print('@output shapes:')  
 for x in op.outputs:  
 print("name = {} : {}".format(x.name, x.get\_shape()))  
 if op.type in ops\_dict:  
 ops\_dict[op.type] += 1  
 else:  
 ops\_dict[op.type] = 1  
  
 print('---------------------------------------------------------------------------------------------------------------------------------------------')  
 sorted\_ops\_count = sorted(ops\_dict.items(), key=operator.itemgetter(1))  
 print('OPS counts:')  
 for i in sorted\_ops\_count:  
 print("{} : {}".format(i[0], i[1]))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 """  
 Write a summary of the frozen TF graph to a text file.  
 Summary includes op name, type, input and output names and shapes.  
  
 Arguments  
 ----------  
 - path to the frozen .pb graph  
 - path to the output .txt file where the summary is written  
  
 Usage  
 ----------  
 python inspect\_pb.py frozen.pb text\_file.txt  
  
 """  
 if len(sys.argv) != 3:  
 raise ValueError("Script expects two arguments. " +  
 "Usage: python inspect\_pb.py /path/to/the/frozen.pb /path/to/the/output/text/file.txt")  
 inspect(sys.argv[1], sys.argv[2])

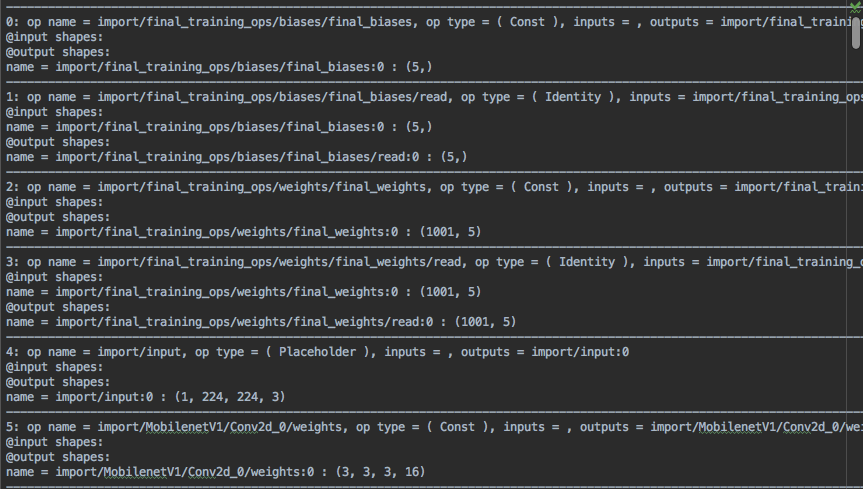
The above code will take the model file as input argument and save all the operations and input / output node names with description in a text file which we supply as input.

To run this you can enter the command as

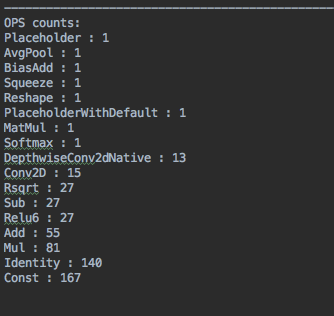
**Python inspect.py retrained\_graph.pb summeries.txt**

In this command you are executing inspect.py code you have saved before. And input the graph file obtained from the previous section and path of a text file where you want to save the summaries.

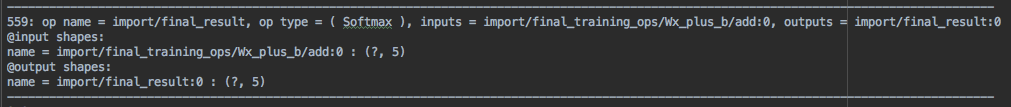
Once you execute this command summeries.txt will be created with all the summaries as shown below will be added in to that file.



In this file you can see all the operation and input and output names and there shapes. Along with this you can also see the overall operators used in this.

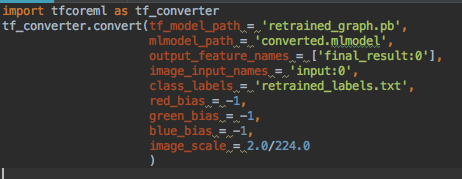


Now scroll down to the bottom of the file. You will see the definition of the end node. In this case it will be like this.



Here you can see that, the end node operation type is **Softmax**, and the output that it will produce will be stored in the name **final\_result:0**.

Now see the below code which is used to generate corresponding coreml model for this.



In the above code you can see that we have imported the tfcoreml package in the first line.

Then used its **convert** function. For this the following are the arguments.

Tf\_model\_path: the (.pb) file path which you have generated in the previous section.

Mlmodel\_path: Output model file path where you want to generate the model.

Output\_feature\_names: In this we will get the output variable name which you obtained in the from the previous text file generated by our model inspection code.

Image\_input\_names: name you want to give for the image input. In coreml / IOS this will be the image buffer.

Class\_labels: this is the file you will get in the training step.

Once you run the above code you will see the generated converted.mlmodel file in your directory.

This one you can import in to your X-code project and make use of it.

# Creating IOS APP

In this section, we are going to see how to make use of the image recognition model that we created in the earlier sections to predict the images using your IOS mobile camera by creating an app.

To start with the section you need a MAC PC running X-code 9+ version.

First you have to download the source code (x-code project) in the git repository. <http://git.com/project>

Open the image recognization.xcodeproj in the X-code.

Now you will find the project structure as follows.

***//project structure image comes here.***

In this the main file we are going to see is view controller.swift. It will have the following code.

***import UIKit***

***class ViewController: UIViewController {***

***@IBOutlet weak var pictureImageView :UIImageView!***

***@IBOutlet weak var titleLabel :UILabel!***

These are the outlets for the image-view control and title label control in the main story board.

***private var model : Mymodel = Mymodel()***

This the instance of the model which generated when we added the core-ml file generated in the previous section.

***var content : [ String : String ] = [***

***"cheeseburger" : "A cheeseburger is a hamburger topped with cheese. Traditionally, the slice of cheese is placed on top of the meat patty, but the burger can include many variations in structure, ingredients, and composition.\nIt has 303 calories per 100 grams.",***

***"carbonara" : "Carbonara is an Italian pasta dish from Rome made with egg, hard cheese, guanciale, and pepper. The recipe is not fixed by a specific type of hard cheese or pasta. The cheese is usually Pecorino Romano.",***

***"meat loaf" : "Meatloaf is a dish of ground meat mixed with other ingredients and formed into a loaf shape, then baked or smoked. The shape is created by either cooking it in a loaf pan, or forming it by hand on a flat pan.\nIt has 149 calories / 100 grams",***

***"pizza" : "Pizza is a traditional Italian dish consisting of a yeasted flatbread typically topped with tomato sauce and cheese and baked in an oven. It can also be topped with additional vegetables, meats, and condiments, and can be made without cheese.\nIt has 285 calories / 100 grams"***

***]***

***We hard coded the content to display in the title label for the corresponding class label we have trained.***

***let images = ["burger.jpg","pizza.png", "pasta.jpg","meatloaf.png"]***

Thses are the images we have added to the project and also serve as input for our prediction app.

***var index = 0***

***override func viewDidLoad() {***

***super.viewDidLoad()***

***nextImage()***

***}***

***@IBAction func nextButtonPressed() {***

***nextImage()***

***}***

***func nextImage() {***

***defer { index = index < images.count - 1 ? index + 1 : 0 }***

***let filename = images[index]***

***guard let img = UIImage(named: filename) else {***

***self.titleLabel.text = "Failed to load image \(filename)"***

***return***

***}***

***self.pictureImageView.image = img***

***let resizedImage = img.resizeTo(size: CGSize(width: 224, height: 224))***

***guard let buffer = resizedImage.toBuffer() else {***

***self.titleLabel.text = "Failed to make buffer from image \(filename)"***

***return***

***}***

As we trained our model with 224px images we are also resizing the images of the input to the same size and also converting it to image buffer which we want to give to prediction method.

***do {***

***let prediction = try self.model.prediction(input: MymodelInput(input\_\_0: buffer))***

Here we are inputting the image and getting the prediction results.

***if content.keys.contains(prediction.classLabel) {***

***self.titleLabel.text = content[prediction.classLabel]***

***}***

***else***

***{***

***self.titleLabel.text = prediction.classLabel;***

***}***

In the above depending on the class label we are displaying the content to the user.

***} catch let error {***

***self.titleLabel.text = error.localizedDescription***

***}***

***}***

***}***

The mainstory board design will look like this.

//Image will come here

Once you run the app it will the output like this.

