Project Report for GNG 5299

For Partial Fulfillment of Project GNG 5299 Course Credits for Winter 2020

Submitted By

Rajshkehar Mukherjee
Student ID : 3000330835
M.Eng (Electrical and Computer Engineering)
University of Ottawa

Submitted To :

Prof Hussein Al Osman
University of Ottawa

Week of Feb 24th :

- Understanding and knowledge transfer for the project

After some research work, I have concluded some of the points that I found to be important in the progress:

Given it is a TCN model architecture, and involves convolutions so I am trying to use tensorflow lite to try to quantize or at least minimize the overall size of the model. That would help reduce the memory footprint of the model and hence help when trying to run on a mobile device with low memory capacity.

Also, the operations used in the model can be scheduled in such a way that it can run more efficiently while utilizing less compute power.

This can be approached with some techniques that are proven to work with deep learning models, and I am trying to find out and also which ones(s) might be most suitable for the model architecture that we use in the code. Some of the methods that I am working on are following:

- There is the strategy to produce fusion of operations - and hence do less number of multiply-accumulate cycles. So I am trying to extract and list out all the convolution, pooling, activation function layers utilized through the main code of the model, and then plot them in a graph, so that one can find which model operations can be potentially fused together.

- Also I am trying to delve further on the tflite conversion of the model - using a python API or a C++ API in the easiest way possible. But since code is already in python, thus making a python API for model conversion may be better idea.

Week of March 3rd

Learning about TensorRT and TFLite for Mobile Android and iOS platforms for the TCN model that we have to deploy on mobile device.
Also understanding of the TCN model graph itself would help us in unravelling the functions embedded in the model and how to optimize those functions, their order as well - to more efficiently process on mobile devices.

How to implement a simple ML model on Android

How to convert model from any (tensorflow, keras) format into tflite format

How to shrink model size and optimize model operations such as convolution2D and matrix multiplication, activation functions etc. for mobile platforms

1. · *Making repo of TFLite*
2. · *Understanding the code flow of any tflite model graph*
3. · *Documenting in form of power point bullets for easy understanding*
4. · *TFLite ppt link : Making a presentation about this project and it's development*
5. · *TO DO :*
6. · *Make and run a tflite Android graph int8*
7. · *Set up demo for 8 graphs (lenet to resnet50) both for fp32 and int8 quantized*

Simple Steps for compiling a Deep learning model in TFLite Android/ iOS development:

I researched a lot about the development and code workflow that needs to be followed for deploying any deep learning model on mobile or small memory footprint devices and came up with the following flow . I have also attached links that may be relevant for each steps to be more helpful.

Development workflow :

The workflow for using TensorFlow Lite involves the following steps:

i.Pick a model

Bring your own TensorFlow model, find a model online, or pick a model from our **Pre-trained models** to drop in or retrain.

ii.Convert the model

If you're using a custom model, use the **TensorFlow Lite converter** and a few lines of Python to convert it to the TensorFlow Lite format.

iii.Deploy to your device

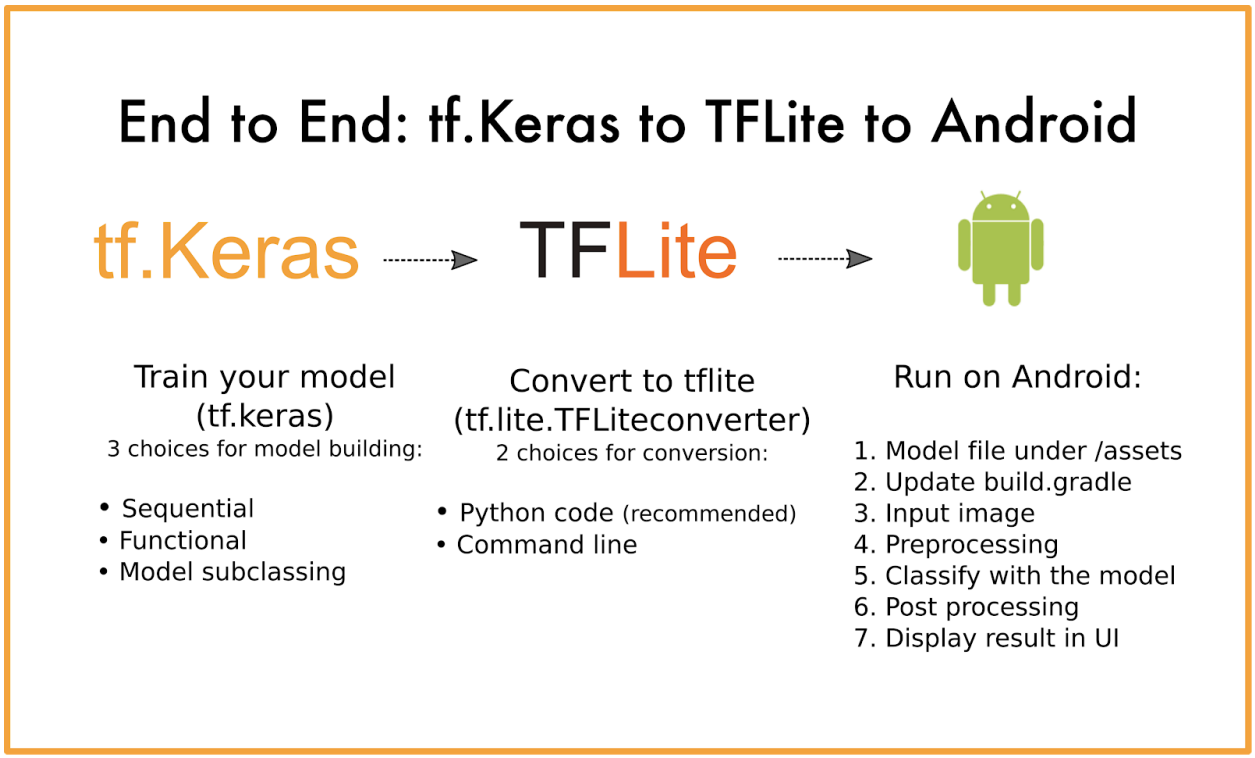
Run your model on-device with the **TensorFlow Lite interpreter**, with APIs in many languages.

iv. TensorflowLite will then execute the model with the input given by the interpreter.

v.Optimize your model

Use our **Model Optimization Toolkit** to reduce your model's size and increase its efficiency with minimal impact on accuracy.

From <https://www.tensorflow.org/lite/guide>



Week of March 9th:

To do :

- **Know more about the tensorflow and tflite flow**

- **Understanding the quantization flow for inputting a pre-trained model from frontend and then quantizing the weights and then compiling the model on TFLite . This uses post-training quantization of weight parameters. It does not use context aware quantization.**

- **Also an option to read and download raw models (not pre-trained models) and then compile then with context aware quantization - this can take care of parameters passed on to layer after layer and pack the weight parameters such that it is efficiently quantized**

Code flow for running a model on TFLite compiler :

· **I first consider a pre-tained model that has been already uploaded on the mobilenet model zoo**

Why pre-trained ?

Because since TFLite is relatively new and in poduction still, not all the operators of tensorflow are supported by Tflite compiler.
So using a pe-tained model is safe and less surprises will result.

Why mobilenet model zoo?

· **Mobilenet models are made as very efficient and low memory and processor footprint . This has been the main focus of their architecture and also while being compatible with TFLite operator library, they ae also possible to be quantized with the TFLite convertor (TOCO) library.**

· **Model zoo for mobilenet graphs is having a variety of graphs including object detection, recognition, etc.**

MobileNet versus Inception achitectures :

- **Faster in execution and lightweight**

- **Mobilenet have depthwise convolution implemented for performing convolutions instead of the normal conv2D. So they use depthwise_conv operator which is significantly faster as it saves number of operations and parameters. sums and multiplications to be performed to calculate the weights of layers is also reduced.**

· **Here is the link for mobilenet model zoo which can be compatible with TFLite compiler:**

Two files after downloading the tflite model file -

labels.txt file which has all the labels the model will need

Model.tflite file which is the tflite converted model (already in tflite format)

· **Found an easier way to run TFLite files - and test on various platforms and various models - Google Collaboratory**

Also about Quantization of models in TFLite format, some interesting facts and comparisons :

Optimization options

There are several post-training quantization options to choose from. Here is a summary table of the choices and the benefits they provide:

Learning some tricks to Convert and interpret Tensorflow models in TFLite format for efficient execution :

1.

Converting a Keras model

The following example shows how to convert a `tf.keras` model into a TensorFlow Lite `FlatBuffer`.

```
import tensorflow as tf


# Create a simple Keras model.
x = [-1, 0, 1, 2, 3, 4]
y = [-3, -1, 1, 3, 5, 7]
```

```
model = tf.keras.models.Sequential(
    [tf.keras.layers.Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')
model.fit(x, y, epochs=50)

# Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
```

Only one way to use TFLite on CPUs :

With Python APIs

So converting a model into TFLITe using Python API for executing the model

- **`TFLiteConverter.from_saved_model()`: Converts SavedModel directories.**
- **`TFLiteConverter.from_keras_model()`: Converts `tf.keras` models.**
- **`TFLiteConverter.from_concrete_functions()`: Converts concrete functions.**

From <https://www.tensorflow.org/lite/convert/python_api#python_api>

End-to-end MobileNet conversion

```
import numpy as np
import tensorflow as tf

# Load the MobileNet tf.keras model.
model = tf.keras.applications.MobileNetV2(
    weights="imagenet", input_shape=(224, 224, 3))

# Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Load TFLite model and allocate tensors.
interpreter = tf.lite.Interpreter(model_content=tflite_model)
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Test the TensorFlow Lite model on random input data.
```

```
input_shape = input_details[0]['shape']

input_data = np.array(np.random.random_sample(input_shape),
dtype=np.float32)

interpreter.set_tensor(input_details[0]['index'], input_data)


interpreter.invoke()


# The function `get_tensor()` returns a copy of the tensor data.

# Use `tensor()` in order to get a pointer to the tensor.

tflite_results = interpreter.get_tensor(output_details[0]['index'])


# Test the TensorFlow model on random input data.

tf_results = model(tf.constant(input_data))


# Compare the result.

for tf_result, tflite_result in zip(tf_results, tflite_results):

  np.testing.assert_almost_equal(tf_result, tflite_result, decimal=5)
```

Update of March 20th :

Currently working on the following directions :

- Still working on the tflite model conversion using various APIs
- Some research about java gradle deployment using Android studio
- How to do inference and training on Java based gradle in Android studio - while having python frontend for the model code
- Would be interesting to see how to call python APIs for the python code of TCN, but in the java environment based gradle in Android studio
- Size of model and size of code requirements to be checked with the Android studio limitations

To do :

- Download and try to make a sample .apk app in Android Studio

Update of March 27th :

First thinking of trying out TFLIte with various graphs.

added a page with a table listing all the graphs, and current status

Graphs that can be tried which are similar to the structure that TCN graph has :

TFLite Graph Status, using basic TFLite (e.g., not with flex delegate)

Graph	Location in Raj Zoo Repo	Float32 working	Int8 working	Notes
Mobilenet v1	TFLite /graphFP32/MobileNet v1/mobilenet_v1_0.25_160.tflite	Working	· Working , with less accuracy and less size	· Seems to be most versatile and adaptable to various deep learning applications ; due to depth wise convolutions and some other architecture hacks, it has faster execution and comparatively lightweight
Mobilenet v2	TFLite /graphFP32/MobileNet v2/mobilenet_v2_1.0_224.tflite	Working	· Working , and with less accuracy and size	· Python API may be good for conversion but , it seems for keras models it follows a different path to convert models

Inception v2	/TFLite /graph_quant/inception v2 quant/ inception_v1_224_quant.tflite	Working	Working now	· Able to convert it now, seems some problem with functions used but now is ok · Size of inception larger than 100 Mb , so to keep in the ca09 server
Resnet	/TFLite /graph_quant/inception v2 quant/ inception_v1_224_quant.tflite	working	Working now	· Resnet has huge size, and less accuracy, also after quantization and tflite conversion it can have 200% decrease in size · Size of resnet larger than 100 Mb , so to keep in the ca09 server
Squeezenet	TFLite /graphFP32/Squeezenet/squeezenet.tflite	working	Not yet able to convert	· Not yet tested with int8 version, but should work

- **Will keep filling the graph as I try to compress and run these graphs and check thier accuracies and wheather they work or not with TFLite format and quantization to smaller sizes**

- Now Distiller is also a very nice tool to be used for the quantization of models to be used on the mobile platform.

I did some research as to how to use it and the variations available and found the following :

- **Working on Distiller and any compiler such as TFLite flow - Integrate and try distiller output on various other frameworks and learn how it works**
- **Check if TFLite also has a flow where it consumes FP32 and converts to INT8 bit - how is that different from distiller flow**
- **Understanding overall TFLite + Relay IR flow and collecting documentation for various graphs and frameworks that can be consumed by TFLite**
- **Check with various framework model graphs -> try to quantize and tun on TFLite**
- **Check for Tensorflow quantization of model files -> feed into TFLite**
- **Find out about TFLite models -> useful for quantization -> large model zoo in repo**
- **Check : TFLite models running with android studio ? If yes, then possible solution for running quantized models on android studio**
- **Complete Training modules**

- Working on CNN toolkits like OpenVino, facebook tool for quantization

- **Kill the bits fb project using Pytorch**
- **Works on Resnet-18 and ResNet-50**

Model (non-compressed top-1)	**Compression**	**Size ratio**	**Model size**	**Top-1 (%)**

ResNet-18 (69.76%)	Small blocks Large blocks	29x 43x	1.54MB 1.03MB	65.81 61.18
ResNet-50 (76.15%)	Small blocks Large blocks	19x 31x	5.09MB 3.19MB	73.79 68.21

Steps to install and run Distiller

Step 1 :
Distiller being installed in Ububtu 16.04 and python 3.6

- Without GPU

Step 0

There was an error in installing the distiller due to using the conda environment for installing some of the dependencies
Making sure no environment is activated while following the below step 1 is important
If any conda environment is activated , then the python of conda base will be used , instead of the usr/bin/python and the packages will be installed in that python
So make sure you do not install some in conda python and then try to run the distiller using normal python

Step 1
Clone distiller

$ git clone https://github.com/NervanaSystems/distiller.git

Step 2
Create venv

$ sudo apt-get install python3-venv

$ python3 -m venv env

As with virtualenv, this creates a directory called distiller/env.

```
$ source env/bin/activate
```

Step 3

Install the Distiller package

For Linux , pip install with no CUDA :
Run this command in terminal:

```
$ pip3 install https://download.pytorch.org/whl/cpu/torch-1.1.0-cp36-cp36m-linux_x86_64.whl
```

to install the compatible build of Pytorch 1.1 and torchvision 0.3, before installing the package.
Finally, install the Distiller package and its dependencies using pip3:

```
$ cd distiller
 $ pip3 install -e .
```

There seems to be an error in this command
Seems to not isntall the packages properly in the python3 usr bin module ,
as I chekced sometimes after using this command .
So using another workaround for this :
pip install -r requirements.txt

This seems to work but going into python3 command line and tying to import packages like
Import vision
Import numpy

Show errors s

Thus some error with packages while installing.

- Also, encountered a numpy version error
- corrected it by un installing numpy until it worked'

DO the following if import for any package does not work :

$ pip3 show numpy

- **Should show the bad version of numpy (recently installed)**

$ pip3 uninstall numpy

- **Should remove the bad version**

$ pip3 show numpy

- **Should now point to the previous stable version of numpy**

If uninstalling once, shows to the same numpy version, uninstall again till it points to the stable version

This problem again repapparwed while running the classificatrion python file Did the uninstall of numpy and install of numpy until it was showing the path of numpy to be : usr/bin/site-packages

This installs Distiller in "development mode", meaning any changes made in the code are reflected in the environment without re-running the install command (so no need to re-install after pulling changes from the Git repository)

Step 5
Three examples provided to be used with distiller

We focus on the quantization (model compression) exapmlpe

- **Training-only**
- **Getting parameter statistics of a sparsified model**
- **Post-training quantization**

Example :

1. **Tutorial: Post-Training Quantization of a Language Model [for ResNet20 model]**

2. https://github.com/NervanaSystems/distiller/blob/master/examples/word_language_model/quantize_lstm.ipynb [for RNN model]

We follow the first one .

Post-training quantization examples :

1. **ResNet20 on Cifar Dataset Example**

This example performs 8-bit quantization of ResNet20 for CIFAR10.
We've included in the git repository the checkpoint of a ResNet20 model that we've trained with 32-bit floats, so we'll take this model and quantize it:

```
$ python3 compress_classifier.py -a resnet20_cifar ../../../data.cifar10 --resume
../ssl/checkpoints/checkpoint_trained_dense.pth.tar --quantize-eval --evaluate
```

(if we want to try more examples for quantization and more modes/variations while quantization of a model : https://github.com/NervanaSystems/distiller/blob/master/examples/quantization/post_train_quant/command_line.md)

Also,
Command Line vs. YAML Configuration

Post-training quantization can either be configured straight from the command-line, or using a YAML configuration file. Using a YAML file allows fine-grained control over the quantization parameters of each layer, whereas command-line configuration is "monolithic". To apply a YAML configuration file, use the `--qe-config-file` argument.
Configure the YAML file.
Then run the py file :

```
# python compress_classifier.py -a resnet18 -p 10 -j 22 <path_to_imagenet_dataset>
--pretrained --evaluate --quantize-eval --qe-config-file
../quantization/post_train_quant/resnet18_imagenet_post_train.yaml
```

We can also create sample datasets (CIFAR 10 is automatically downloaded) but Imagenet can also be used

Running output of the file : compression_classifier.py with default cifar10 dataset on resnet20 layers

To run other examples :

Imagenet Dataset not being able to download issue

If that is doen , simply use this comand to generate the yaml file:

```
python compress_classifier.py -a resnet50 -p 10 -j 22 <path_to_imagenet_dataset>
--pretrained --qe-calibration 0.05
```

Or one can also use the given yaml file at:
distiller/examples/quantization/post_train_quant/resnot50_quant_stats.yaml

Then this command to modify and obtain more results on the resnet 50 model :

```
python compress_classifier.py -a resnet50 --pretrained <path_to_imagenet_dataset>
--evaluate --quantize-eval --qe-per-channel --qe-clip-acts avg --qe-no-clip-layers fc
--qe-stats-file ../quantization/post_train_quant/stats/resnet50_quant_stats.yaml
```

2. **LSTM quantization example :**

https://github.com/NervanaSystems/distiller/blob/master/examples/word_language_model/quantize_lstm.ipynb [for RNN model]

Step 1:

Go to folder : distiller/examples/word_language_model

Step 2 :

Install jupyter notebook for quantization example
Ensure that youa re in the same venv that you would lioke to work with

Then in terminal :

- $ pip3 install jupyter

Once it is installed , open the notebook (its generally an ssh connection that opens in your browser :

- $ jupyter notebook

Step 3 :

Open the *quantize_lstm.ipynb* in the jupyter notebook directory

Changed the code :

device = 'cpu:0'

Some points on Quantization :

1. **"Net-Aware" Quantization**

The term "net-aware" quantization, coined in this paper from Facebook (section 3.2.2), means we can achieve better quantization by considering sequences of operations instead of just quantizing each operation independently. This isn't exactly layer fusion - in Distiller we modify activation stats prior to setting quantization parameters, in to make sure that when a module is followed by certain activation functions, only the relevant ranges are quantized. We do this for:

- **ReLU - Clip all negative values**
- **Tanh / Sigmoid - Clip according to the (approximated) saturation values for these functions. We use [-4, 4] for tanh and [-6, 6] for sigmoid.**

2. **Static vs. Dynamic Quantization of Activations**

Distiller supports both "static" and "dynamic" post-training quantization of activations.

- Static Quantization: Pre-calculated tensor statistics are used to calculate the quantization parameters.
- Dynamic Quantization: Quantization parameters are re-calculated for each batch.

  Support for this mode is limited. It isn't as fully featured as static quantization, and the accuracy results obtained when using it are likely not as representative of real-world results.

  Specifically:

  - Only convolution, FC (aka Linear) and embedding layers are supported at this time. Non-supported layers are kept in FP32, and a warning is displayed.
  - "Net-aware" quantization, described above, isn't supported in dynamic mode.

3. Note 1: Accuracy Loss When Clipping Activations

Notice the degradation in accuracy in run (4) - ~2.6% compared to per-channel without clipping. Let's recall that the output of the final layer of the model holds the "score" of each class (which, since we're using softmax, can be interpreted as the un-normalized log probability of each class). So if we clip the outputs of this layer, we're in fact "cutting-off" the highest (and lowest) scores. If the highest scores for some sample are close enough, this can result in a wrong classification of that sample.
We can provide Distiller with a list of layers for which not to clip activations. In this case we just want to skip the last layer, which in the case of the ResNet-50 model is called `fc`. This is what we do in run (5), and we regain most of the accuracy back.

4. Note 2: Under 8-bits

Runs (8) - (10) are examples of trying post-training quantization below 8-bits. Notice how with the most basic settings we get a massive accuracy loss of ~53%. Even with asymmetric quantization and all other optimizations enabled, we still get a non-trivial degradation of just under 2% vs. FP32. In many cases, quantizing with less than 8-bits requires quantization-aware training. However, if we allow some layers to remain in 8-bit, we can regain some of the accuracy. We can do this by using a YAML configuration file and

specifying overrides. As mentioned at the top of this document, check out the resnet18_imagenet_post_train.yaml file located in this directory for an example of how to do this.

Update of April 3rd and 10th : To be Done

- **TO Work on Inception module and put it on a mobile app for performing classification :**

Some Key Features :

●Stacked convolutional layers followed by max-pooling, fully connected layer

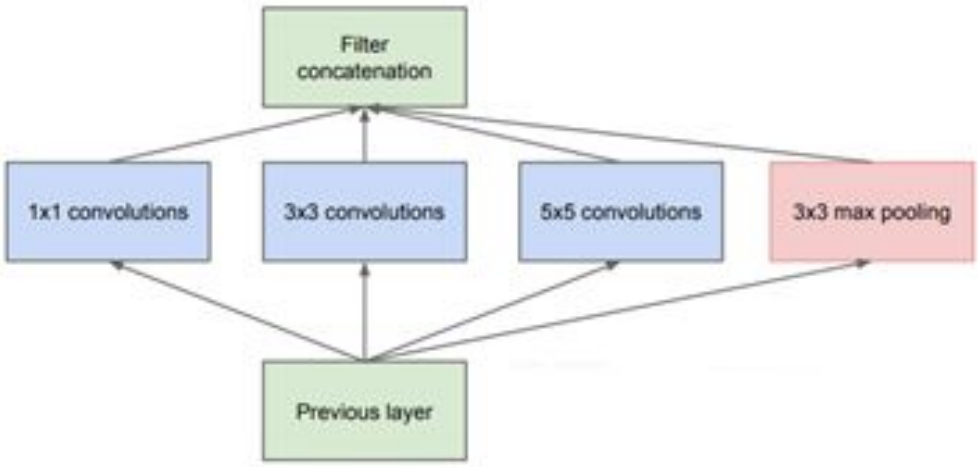
●Concept of drop-out introduced to avoid overfitting

●Inspired by neuroscience, Gabor filters of different sizes to handle multiple scales of features

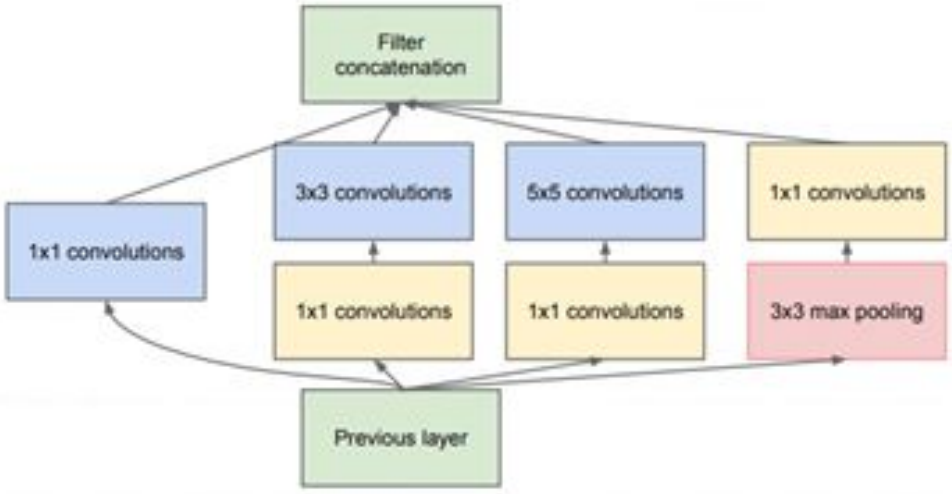
●All layers of Inception learned, contrary to fixed 2-layer deep model

●1x1 convolution filters help in dimensionality reduction, thus reduce complexity

●Allows more width and depth in network with less computational penalty

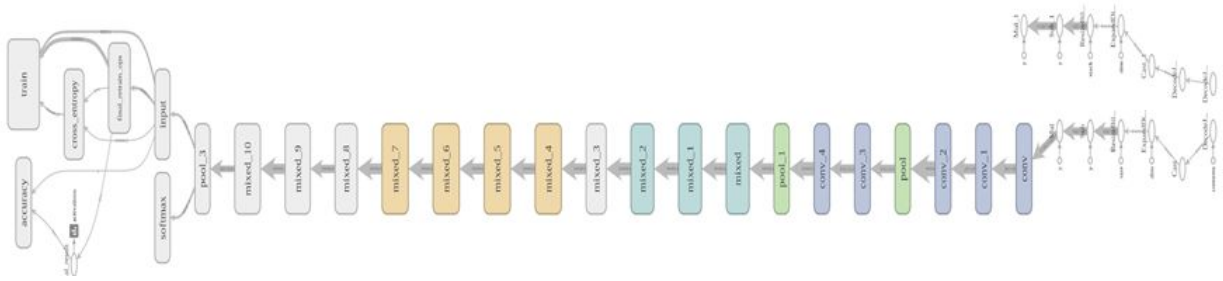


(a) Inception module, naïve version

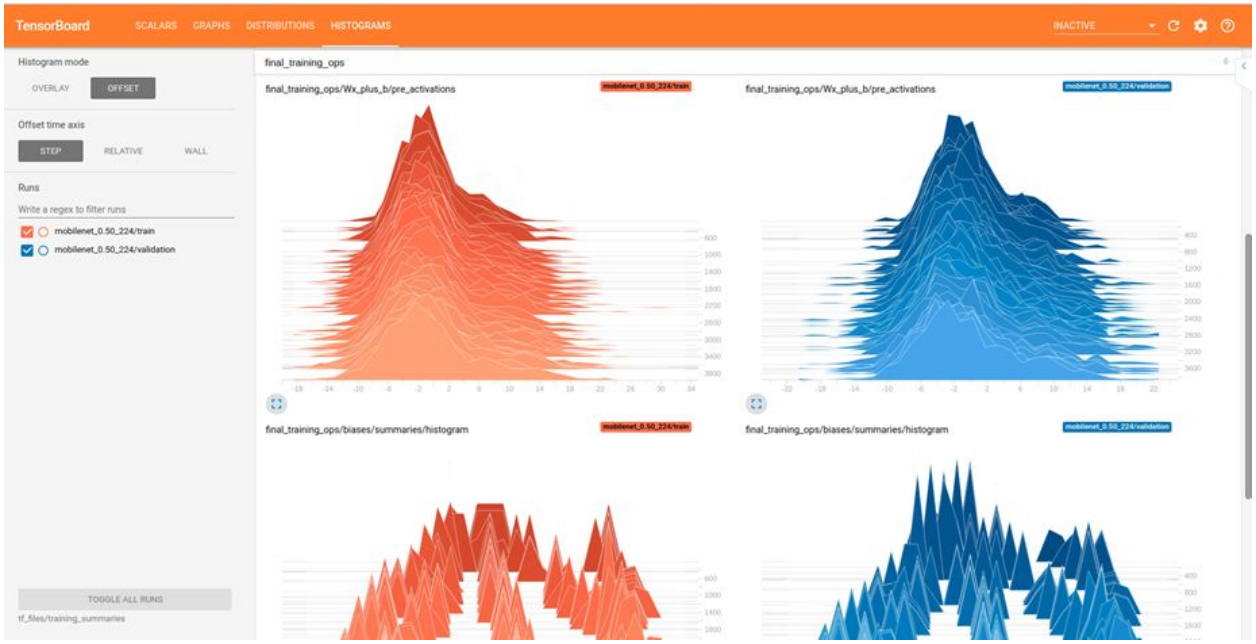


(b) Inception module with dimensionality reduction

Inception V3 training Model architecture

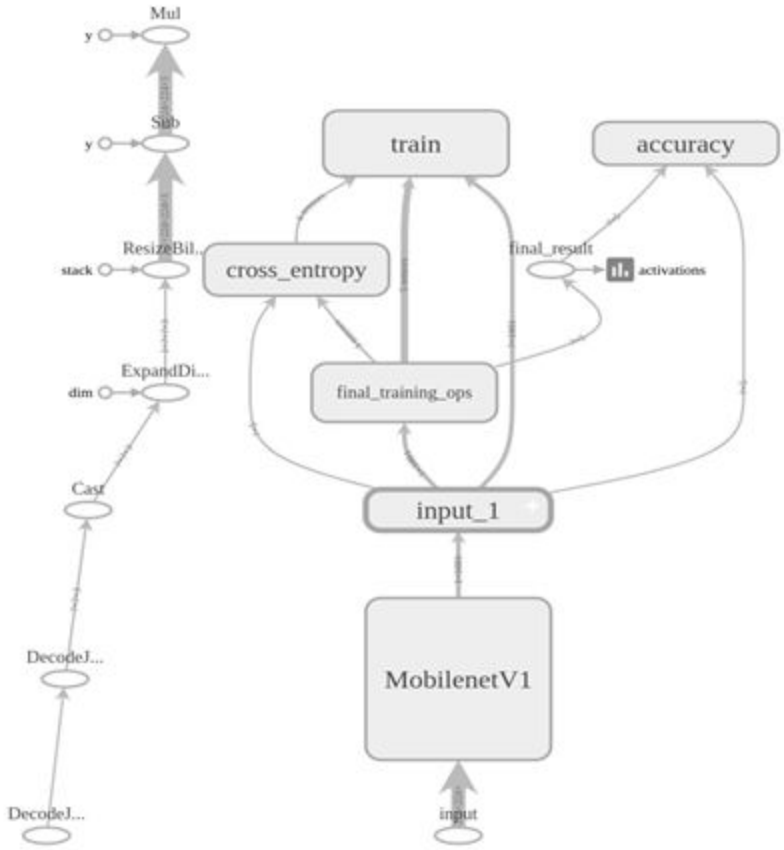


Pre-activation and biases graphs

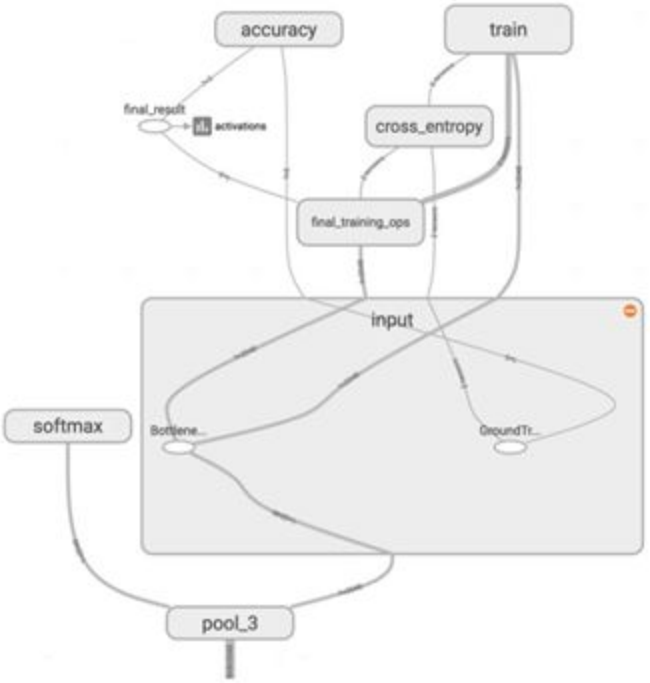


Cross validation graphs [red-training , blue-validation]

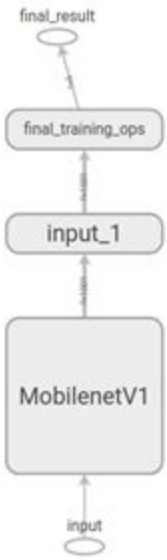
Data-Flow graph of our model : MobilenetV1 training using multiple pooling, cross validation and activation



Training the final output layer

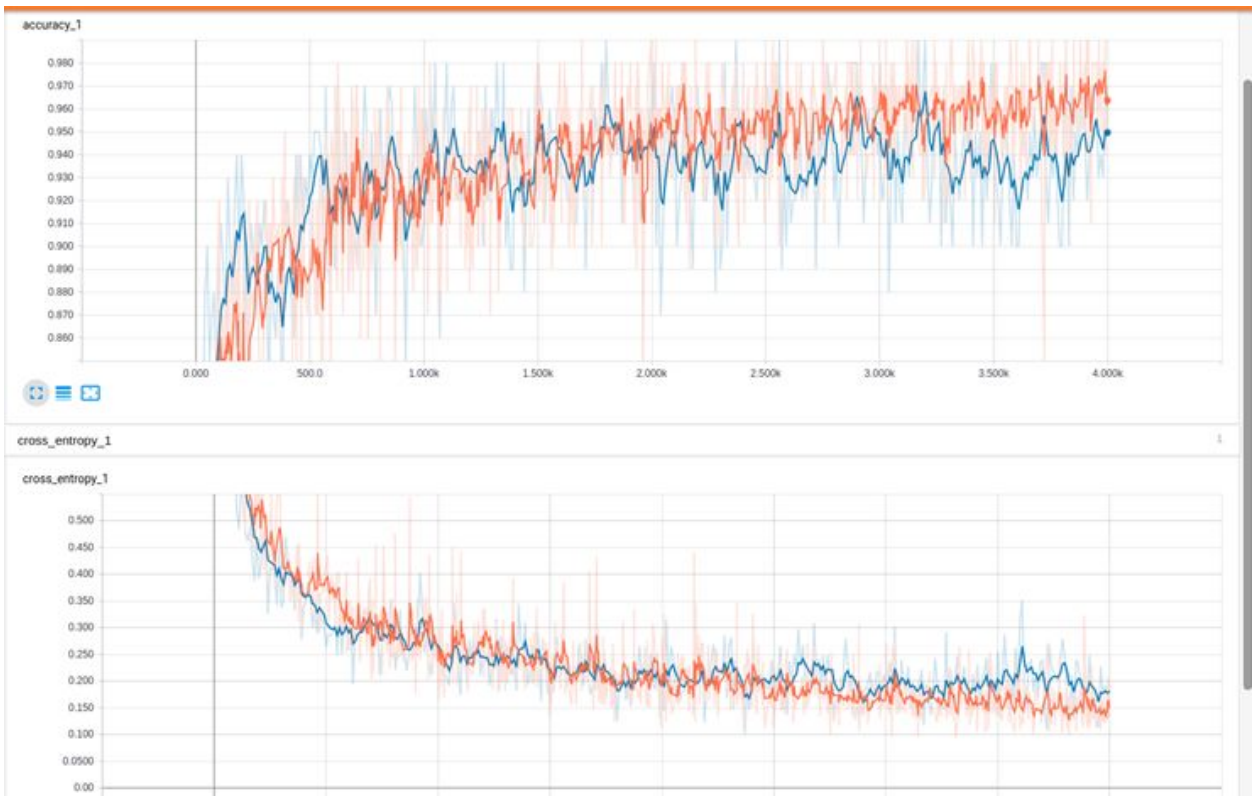


For the mobile platform : An optimized version of reduced complexity

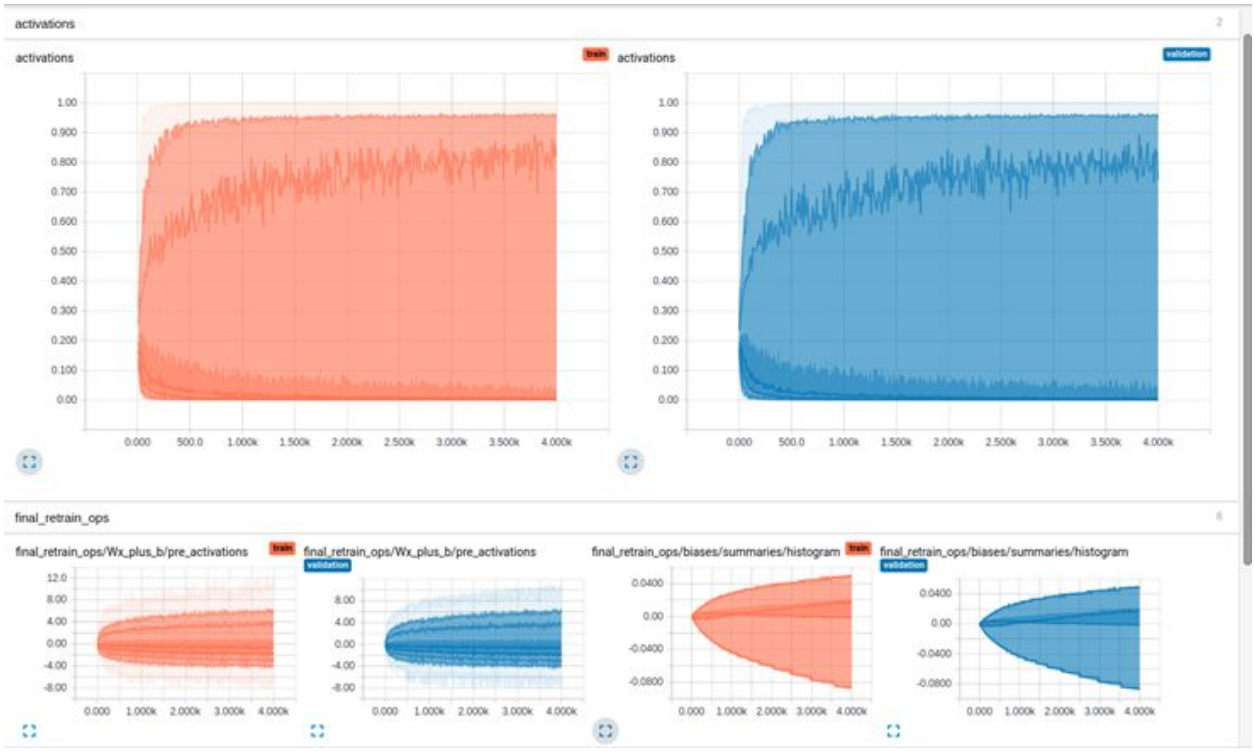


Training epochs on MobileNet model (GooGLe net)

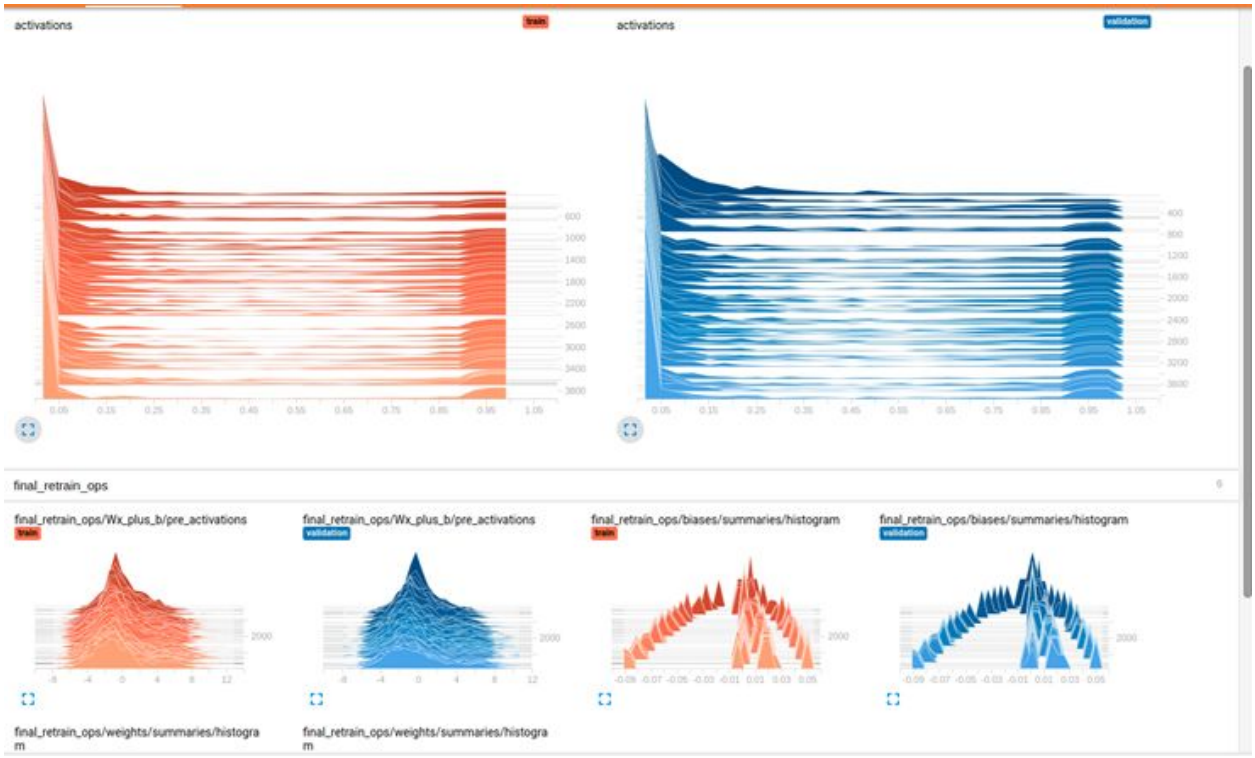
Training epochs on Inception V3 model



Cross validation graphs [red-training , blue-validation] : Inception V3



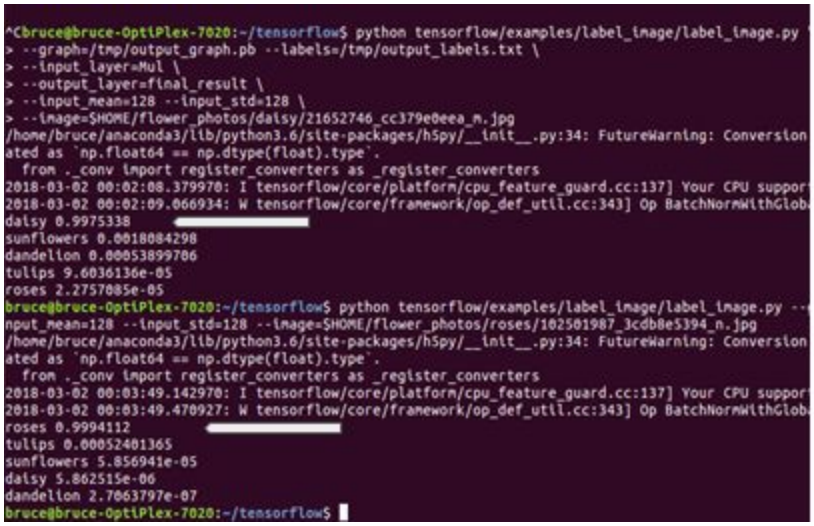
Pre-activation and bias graphs : Inception V3


activations
train
activations
validation
final_retrain_ops
final_retrain_ops/Wx_plus_b/pre_activations
train
final_retrain_ops/Wx_plus_b/pre_activations
validation
final_retrain_ops/biases/summaries/histogram
train
final_retrain_ops/biases/summaries/histogram
validation
final_retrain_ops/weights/summaries/histogram
final_retrain_ops/weights/summaries/histogram

Results of classification of Flowers : MobileNet

```
>     --graph=tf_files/retrained_graph.pb  \
>     --image=tf_files/flower_photos/daisy/21652746_cc379e0eea_n.jpg
/home/bruce/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:34: FutureWarning:
ated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
2018-03-01 01:45:34.749444: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your

Evaluation time (1-image): 0.326s

daisy 0.99874085
dandelion 0.00110975
sunflowers 0.00014903292
roses 4.073964e-07
tulips 5.3334177e-09
bruce@bruce-OptiPlex-7020:~/tensorflow-for-poets-2$ python -m scripts.label_image \
>     --graph=tf_files/retrained_graph.pb  \
>     --image=tf_files/flower_photos/roses/2414954629_3708a1a04d.jpg
/home/bruce/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:34: FutureWarning:
ated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
2018-03-01 01:48:15.724353: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your

Evaluation time (1-image): 0.118s

roses 0.977218
tulips 0.022781836
dandelion 2.2188662e-07
sunflowers 3.50745e-08
daisy 5.150656e-09
```

Results of classification of Flowers : Inception V3



Scaling down the model on a brain like processor and energy intensive platform : Deployment

Optimize the model architecture into a simpler version. Drop in accuracy <1%. (Refer earlier slide for figure

The retrained model was still 100 MB in size. This large size of data might be a limiting factor for any algorithm being processed in the brain. Normal zip compression algorithm : Only **8% compression obtained**

Without any changes to the structure of the network, all **floating point weights were quantized** with the constants in place. Successfully reached **70% compression ratio** with a **drop in accuracy of only 1-2%.**

Update of April 17th :

So in this case ::

I will comapre between various models , as described below :

Mobilenet architecture with 224 input size with

Mobilenet v1 quantized

Mobilenet v1 floating point

Mobilenet v2 quantized

Mobilenet v2 floating point

Taking the link for model from here :
https://www.tensorflow.org/lite/guide/hosted_models

And substituting it in the line : (from this link :
<https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/examples/python/>)

```
# Get photo
curl
https://raw.githubusercontent.com/tensorflow/tensorflow/master/tensorflow/lite/examples/label_image/testdata/grace_hopper.bmp > /tmp/grace_hopper.bmp


# Get model   # Substitute the link for your model here
curl
https://storage.googleapis.com/download.tensorflow.org/models/mobilenet_v1_2018_02_22/mobilenet_v1_1.0_224.tgz | tar xzv -C /tmp

# Get labels
curl https://storage.googleapis.com/download.tensorflow.org/models/mobilenet_v1_1.0_224_frozen.tgz  |
tar xzv -C /tmp  mobilenet_v1_1.0_224/labels.txt
```

mv /tmp/mobilenet_v1_1.0_224/labels.txt /tmp/

Then Run the model using label_image.py file in the directory : /tensorflow/tensorflow/lite/examples/python

```
python3 label_image.py \
 --model_file /tmp/mobilenet_v1_1.0_224.tflite \
 --label_file /tmp/labels.txt \
 --image /tmp/grace_hopper.bmp
```

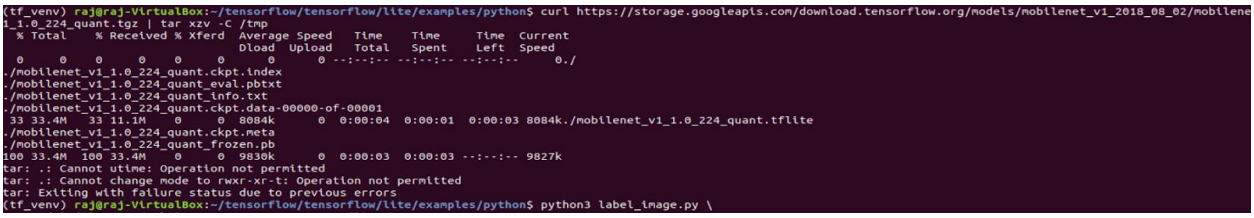
From <https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/examples/python/>

While substituting model name to your model's name.

You should see results like this:

0.728693: military uniform
0.116163: Windsor tie
0.035517: bow tie
0.014874: mortarboard
0.011758: bolo tie

With Mobilenet v1 quantized



```
(tf_venv) raj@raj-VirtualBox:~/tensorflow/tensorflow/lite/examples/python$ python3 label_image.py \
> --model_file /tmp/mobilenet_v1_1.0_224_quant.tflite \
>   --label_file /tmp/labels.txt \
>    --image /tmp/grace_hopper.bmp
0.658824: 653:military uniform
0.149020: 907:Windsor tie
0.039216: 458:bow tie, bow-tie, bowtie
0.027451: 668:mortarboard
0.019608: 466:bulletproof vest
(tf_venv) raj@raj-VirtualBox:~/tensorflow/tensorflow/lite/examples/python$
```

With Mobilenet v2 quantized

```
(tf_venv) raj@raj-VirtualBox:~/tensorflow/tensorflow/lite/examples/python$ curl https://storage.googleapis.com/download.tensorflow.org/models/tflite_11_05_08/mobilenet_v2_1.0_22
4_quant.tgz | tar xzv -C /tmp
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0mobilenet_v2_1.0_224_quant.ckpt.data-00000-of-00001
 36 41.4M   36 14.9M    0     0  9183k      0  0:00:04  0:00:01  0:00:03 9177kmobilenet_v2_1.0_224_quant.ckpt.index
mobilenet_v2_1.0_224_quant.ckpt.meta
mobilenet_v2_1.0_224_quant_eval.pbtxt
mobilenet_v2_1.0_224_quant_frozen.pb
 90 41.4M   90 37.2M    0     0  10.1M      0  0:00:04  0:00:03  0:00:01 10.1Mmobilenet_v2_1.0_224_quant_info.txt
mobilenet_v2_1.0_224_quant.tflite
100 41.4M  100 41.4M    0     0  10.2M      0  0:00:04  0:00:04 --:--:-- 10.2M
```

```
(tf_venv) raj@raj-VirtualBox:~/tensorflow/tensorflow/lite/examples/python$ python3 label_image.py \
> --model_file /tmp/mobilenet_v2_1.0_224_quant.tflite \
> --label_file /tmp/labels.txt \
> --image /tmp/grace_hopper.bmp
0.686275: 653:military uniform
0.568627: 458:bow tie, bow-tie, bowtie
0.549020: 835:suit, suit of clothes
0.513725: 907:Windsor tie
0.490196: 753:racket, racquet
```

With Inception v1 quantized :

```
(tf_venv) raj@raj-VirtualBox:~/tensorflow/tensorflow/lite/examples/python$ curl https://storage.googleapis.com/download.tensorflow.org/models/inception_v1_224_quant_20181026.tgz
 | tar xzv -C /tmp
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0graph.pbtxt
inception_v1_224_quant.ckpt.data-00000-of-00001
 61 98.9M   61 61.0M    0     0  10.7M      0  0:00:09  0:00:05  0:00:04 11.4Minception_v1_224_quant.ckpt.index
inception_v1_224_quant.ckpt.meta
inception_v1_224_quant_frozen.pb
 84 98.9M   84 83.5M    0     0  10.8M      0  0:00:09  0:00:07  0:00:02 11.2Minception_v1_224_quant_info.txt
inception_v1_224_quant.tflite
100 98.9M  100 98.9M    0     0  10.9M      0  0:00:09  0:00:09 --:--:-- 11.3M
```

```
(tf_venv) raj@raj-VirtualBox:~/tensorflow/tensorflow/lite/examples/python$ python3 label_image.py \
> --model_file /tmp/inception_v1_224_quant.tflite \
> --label_file /tmp/labels.txt \
> --image /tmp/grace_hopper.bmp
0.690196: 653:military uniform
0.050980: 835:suit, suit of clothes
0.047059: 458:bow tie, bow-tie, bowtie
0.019608: 907:Windsor tie
0.015686: 668:mortarboard
```

Quantized

Also the input image is being converted to the inout dimensions of the type of model in the label_image.py code
For example inception v2 or Resnet v2 are taking in different dimensions for the test data image (not 224x224)

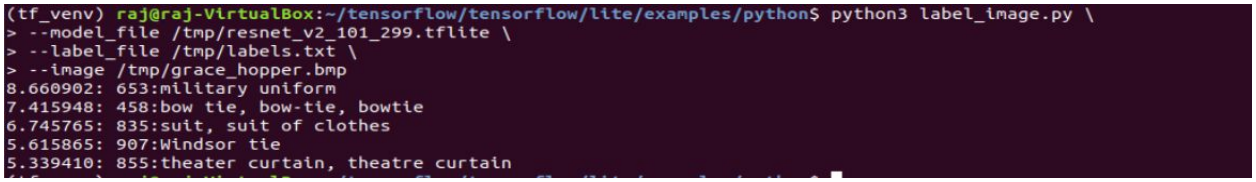
label_image.py code :

```
# check the type of the input
tensor
                floating_model = input_details[0]['dtype'] ==
                np.float32

                # NxHxWxC, H:1, W:2

                height = input_details[0]['shape'][1]

                width = input_details[0]['shape'][2]

                img = Image.open(args.image).resize((width,
                height))

                # add N dim

                input_data = np.expand_dims(img, axis=0)

                i
```

From <https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/examples/python/label_image.py>

So we can try to write the code for various models and test if the models run and prepare a github repo for the same.

With ReseNet v2 :



Able to do the same for more models :

Inception v2	/TFLite /graph_quant/inception v2 quant/ inception_v1_224_quant.tflite	Working	Working now	· Able to convert it now, seems some problem with functions used but now is ok · Size of inception larger than 100 Mb , so to keep in the ca09 server
Resnet	/TFLite /graph_quant/inception v2 quant/ inception_v1_224_quant.tflite	working	Working now	· Resnet has huge size, and less accuracy, also after quantization and tflite conversion it can have 200% decrease in size · Size of resnet larger than 100 Mb , so to keep in the ca09 server

Graph	Location in Raj Zoo Repo	Float32 working	Int8 working	Notes
Mobilenet v1	TFLite /graphFP32/MobileNet v1/mobilenet_v1_0.25_160.tflite	Working	· Working , with less accuracy and less size	· Seems to be most versatile and adaptable to various deep learning applications ; due to depth wise convolutions and some other architecture hacks, it has faster execution and comparatively lightweight
Mobilenet v2	TFLite /graphFP32/MobileNet v2/mobilenet_v2_1.0_224.tflite	Working	· Working , and with less accuracy and size	· Python API may be good for conversion but , it seems for keras models it follows a different path to convert models
Inception v2	/TFLite /graph_quant/inception v2 quant/ inception_v1_224_quant.tflite	Working	Working now	· Able to convert it now, seems some problem with functions used but now is ok · Size of inception larger than 100 Mb , so to keep in the ca09 server

Resnet	/TFLite /graph_quant/inception v2 quant/ inception_v1_224_quant.tflite	working	Working now	· Resnet has huge size, and less accuracy, also after quantization and tflite conversion it can have 200% decrease in size · Size of resnet larger than 100 Mb , so to keep in the ca09 server
Squeezenet	TFLite /graphFP32/Squeezenet/squeezenet.tflite	working	Not yet able to convert	· Not yet tested with int8 version, but should work