Deep Learning on Android Devices

TensorFlow is a library for different data manipulations, but it is mostly used for neural networks. TensorFlow Lite is its light-weight counterpart suitable for running on Android devices.

In this lab we will first see how to ship a neural network with a mobile app and use it for real-time object recognition, and then we will see how to improve the neural network, so that previously unknown objects can be recognized.

Real-time object recognition on Android

Within the TensorFlow codebase you will find an Android app that uses TensorFlow Lite for object recognition. Clone the examples repository with `git clone git@github.com:tensorflow/examples.git` and open a new project in Android Studio from the existing code located at `lite/examples/image_classification/android`

You might have to adjust the Gradle files (e.g. adjust the API version to whatever you have installed on your machine) to get the app to compile.

Note: a few other very useful Android examples can be found in the repository you have just cloned.

Testing the app

We will now run the app in the emulator or on the real phone and see how it classifies objects in real time. If using the emulator you can use the built-in virtual space or pass your webcam image to the emulator (via device settings in AVD).

Once running, the app shows you the top three classes predicted for the given camera image. In addition, the app allows you to choose between a floating point or a quantized model, MobileNet or EfficientNet, select the thread count, and decide whether to run on CPU, GPU, or via NNAPI.

Code inspection

Let’s look at the code. `CameraActivity` and `ClassifierActivity` extended from it are used to capture individual images and provide the user interfaces. The core of the inference happens in `tflite` package classes. Open the `Classifier` file. See how its `create()` method allows for either of the four classifiers to be created: the full (floating point) version or the quantized version of either MobileNet or EfficientNet. Furthermore, the `Classifier` class allows for a `Device` to be set, e.g. should we wish to run the classifier on the GPU. Finally, you can also set the number of threads that the classifier will use. The main part of the pipeline is the `Interpreter`, which performs inferences on the loaded model.

One way this app makes execution faster is by memory mapping the pre-built model. Check `loadMappedFile` function to see how this is done. Note that a different model file is loaded in case of the four neural network versions (inspect `ClassifierQuantizedMobileNet`, `ClassifierFloatMobileNet` and so on) . The files are shipped in the app’s APK package, and you should find them in the “assets” directory of the “models” module.

Before the inference, an image data is stored in a `TensorImage` object called `inputImageBuffer`. See how the object is created and filled in both the floating point as well as the quantized version of the neural network. For the floating point version, the data is normalized before the inference: the mean pixel values are subtracted from each pixel and the values are further divided by the standard deviation.

The inference is finally done when `tflite.run()` is called.

Transfer learning

The network architecture you are using is called MobileNet and is a great fit for on-device learning. The file that ships with the default app is trained on 1001 different categories of objects (labels). However, you might want to recognize some objects that are not there in the original dataset. Rather than training the whole network from scratch, you can use the first few layers of the original network, as they recognize the generic properties (e.g. horizontal/vertical edges, etc.) and add the new higher-layer content trained to fit your training data.

A classifier recognizing different coins would be quite useful. We could use it for automatic payment detection, or to help the visually impaired in certain situations. We don't have labels for different coin denominations in the original network, so we will retrain it to recognize different euro/cent coins. We will perform transfer learning using a Jupyter Notebook.

Prerequisites: All the prerequisites from Lab 2 and 3 + tensorflow python library (install it with `pip install tensorflow`)

Download the dataset of training images from https://bitbucket.org/veljkop/lab-4-transfer-learning and unzip it. It contains a bunch of photos of one euro coins, fifty cent coins, and five cent coins. This is far from an ideal dataset - it was created using videos deconstructed into individual frames with the ffmpeg tool (`ffmpeg -i euro.mp4 euro/euro_%04d.jpg` and so on).

Then, download the Jupyter Notebook from https://bitbucket.org/veljkop/lab-4-transfer-learning and follow the instructions there.

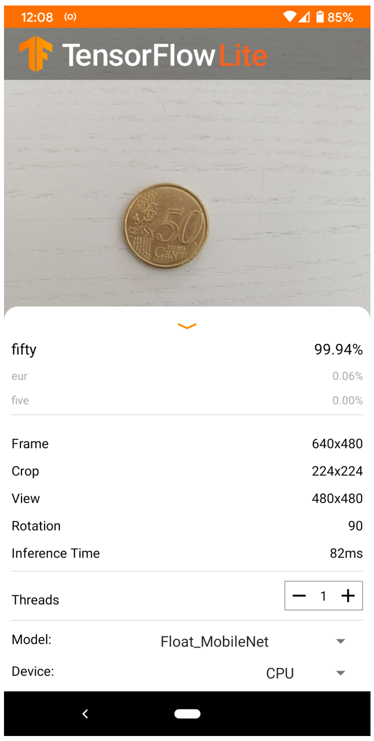
Loading the new model

Copy the new model file `euronet.tflite` to the assets directory. We will use this model whenever a user selects the floating point classifier. The original model uses 1001 labels, but the Euro coins are not among them.

Rename the original `labels.txt` file to `labels_old.txt` and set the `getLabelPath()` of the quantized mobile net classifier to point to these old labels. We will use this classifier for comparison with our new classifier. Copy the new `labels.txt` created during transfer learning to assets. Set the floating point classifier so that it uses the new model file and the new labels.

Compile and test whether the new model recognizes different coins.

The figure on the right shows a moment when the model is working really well - you will find that this is often not the case. Can you explain why this might happen?



```
fifty             99.94%
eur                0.06%
five               0.00%

Frame            640x480
Crop             224x224
View             480x480
Rotation              90
Inference Time      82ms

Threads          − 1 +

Model:   Float_MobileNet
Device:  CPU
```

The solutions should be committed to a private Bitbucket repository named **FRIMS2021-LAB-4** and a user **pbdfrita** (**pbdfrita@gmail.com**) should be added as a read-only member. The solutions will be pulled from your repository on **Sunday, March 27, 23:59**.

Happy coding!

Note: this lab is based on “Recognize Flowers with TensorFlow on Android” TensorFlow tutorial.