Computer Vision Project Report

Submitted by:

Hafeez Olafisayo Jimoh

h.jimoh@innopolis.university

Furqan haider

f.haider@innopolis.university

Submitted to:

Dr. Muhammad Fahim

Marcus Ebner

1 Object Detection with Android Using Yolo and TensorFlow Lite

Mobile deep learning is not really common due to limited computation power of mobile devices due to GPU not really common on mobile phones.

The idea of this project is to implement and deploy YOLO for object detection on mobile device specifically an android device. Previously, the a frozen model is ran on tensorflow on android device but this is computationally expensive and it takes much more time to detect an object. This project work seeks to implement YOLO trained model on Android using tensorflow lite.

The purpose of using the Tensor flow is that its lite version has a mobile optimized interpreter which keeps the apps efficient and The interpreter uses a static graph ordering and a custom memory allocator to ensure minimal load initialization and execution latency.

Since there are different YOLO architectures that can be implemented for this task, the authors of YOLO described a full YOLO version which achieved optimal accuracy and a more compact YOLO called tiny-yolo that run faster but isn't as accurate. Tiny-yolo would be important to our project because it would allow us get reasonable results when deployed due to the limited hardware or computational power of a mobile device.

1.1 Introduction

Inspired from the vision system of Humans and how they can perform visionary tasks so efficiently with great accuracy i.e driving vehicles which in case of computers require strong and complex algorithms to carry out such tasks to detect and distinguish between different objects, This work is an application of object vision and detection with YOLO on android mobile device.Android is a package for mobile devices, including operating system and core applications. The android SDK provides the tools and APIs necessary to develop applications on the android platform using the Java Native Programming language. YOLO is selected for this job based on its fast responses. YOLO sees the entire image during the training and testing procedure thus this gives and edge to this technique having less background errors.

1.2 Related Works

One of the works reviewed on object detection on Android using YOLO has ability to detect 20 classes using the PASCAL VOC dataset: aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, potted plant, sheep, sofa, train and tv/monitor. The network only outputs one predicted bounding box at a time for now.Similar works like this are heavy and required huge computation power because of the use of TensorFlow Mobile which is now deprecated and a previous version of tensorflow lite. Some other related work used YOLO Net on iOS.

2 Dataset

Since we are making use of the YOLOV2 for object detection, the dataset the model is trained on is the COCO dataset with 80 different classes. The pretrained model to be used is the YOLO tiny V2 which is very practical and constrained for mobile environments. The tensorflow model would be converted from from darknet format (.weights) to TensorFlow Protocol Buffers format.(lite). Another similar implementation uses the YOLO tiny v2 and developed an android object detection application using Flutter. This process was done in the CLI and the instructions are provided in appendix section of this report.

3 Idea Implementation

Tensorflow provides an API; a ready for implementing and deploying Machine learning models on mobile and IoT devices. The implementation of this work is with Java for Android Studio using the TensorFlow ML API.

3.1 Preparing the Model

We have used tiny Yolov2 model for the implementation of this work. As mentioned earlier, the tiny model is a reduced model constrained for low computing power environments like mobile To use YOLO v2 for mobile, we need to convert the model from darket format (.weights) to tensorflow protocol buffers format. Protocol Buffers (protobuf) are Google's language-neutral, platform- neutral, extensible mechanism for serializing strcutured data.

3.2 Converting the Tensorflow Model to Tensorflow Lite

The TensorFlow Lite is a set of tools to help develpers run Tensorflow models on mobile, embedded and IoT devices. The TensorFlow Lite Android Support Library makes it easier to integrate models into your application. It provides high-level APIs that help transform raw input data into the form required by the model, and interpret the model's output, reducing the amount of boilerplate code required. It supports common data formats for inputs and outputs, including images and arrays. It also provides pre- and post-processing units that perform tasks such as image resizing and cropping. The process of converting from a frozen tensorflow format(pb) to TensorFlow lite is also shown in appendix section The conversion is done in CLI by a tool known as TensorFlow Lite converter. It takes a tensorflow model and generates a TensorFlow

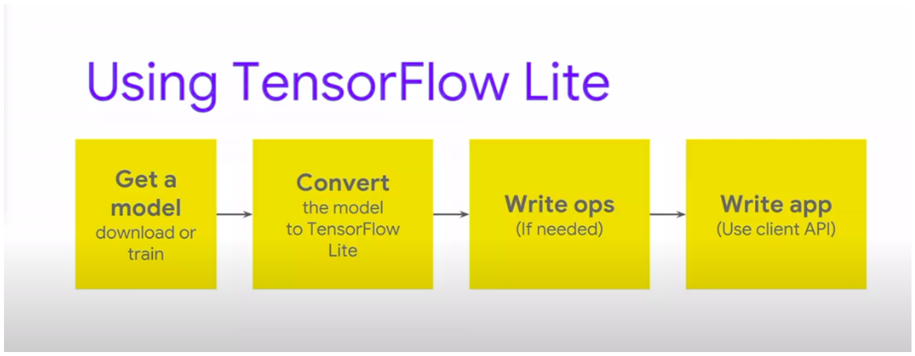

Figure 1: Process of using ML Model for Android AppSource: Google I/O 2018

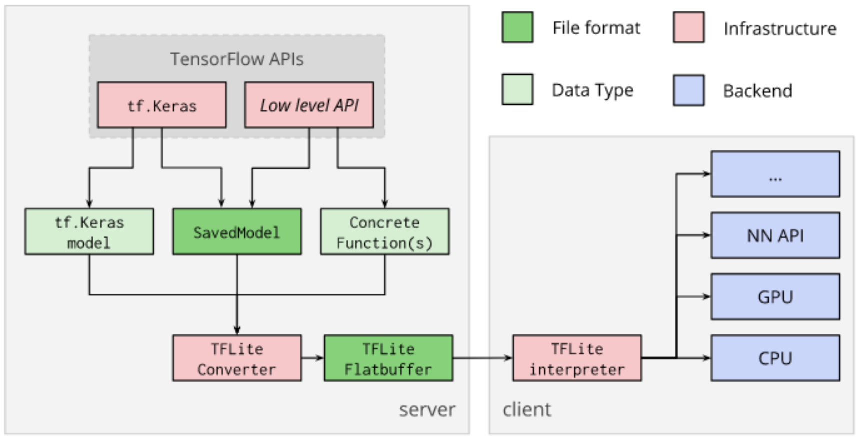

Figure 2: Conversion process to tflite

lite FlatBuffer file(.tflite). The converter supports SavedModel directories, tf.keras models, and concrete functions.

3.3 Integrating Trained Model with Android using Android Studio

The trained model and corresponding labels in txt files is uploaded into the asset folder in our android studio project. The goal is to make use of a camera to capture an image in real time and the image goes through the model and a corresponding prediction or detection of images from the camera, its boundary box coordinates and confidence for each object detection. The boundary box co0rrdinates would be used to draw a rectangle around each object detected from the android app. Tensorflow has an existing API that allows for each programming of an app that makes use of an ML model. This API was equally used in this work to develop the app.

The android part of this work consists if the classifier part that classifies the object detected from the camera. We also have the camera activity and detector activity java files. The camera activity ensures that the camera is started by default and that images from it are sent to the detector for detection which is making use of the tensorflow lite file located in the asset folder. The label file consists of list of all classes which are strings parsed and mapped after detection.

3.4 Future Work and Improvements

We were unable to download the COCO dataset on colab and use for some modification of the end classifier part of the YOLO model. This is due to the large size and colab could not allow to make use of 36gb of size of the data. Given much infrastructure, we would be able to do that and make improvement. We Also, on the android part, instead of outputting the object detected as texts on device screen, a text to speech activity module can be programmed give output as speech. This would be particularly be useful for the visually impaired persons who cannot see the text being displayed on screen

4 Timelines and Responsibilities

Mid-Project

1. Training of YOLO Model on Coco Dataset: **Furqan Haider**
2. Model Conversion: **Hafeez Jimoh**

Final Submission

1. Deploy and Program trained Model on Mobile Phone with Android Studio: **Hafeez Jimoh and Furqan Haider**
2. Report Writing **Hafeez Jimoh and Furqan Haider**

5 References

1. Video Demo on mobile device
2. GITHUB Link to Android Studio project.
3. Real-Time Object Detection with Flutter, TensorFlow Lite and Yolo[Available On Click]
4. YOLO Net on iOS [Available On Click]
5. Android YOLO with TensorFlow Mobile [Available On Click]
6. Huang, Rachel Pedoeem, Jonathan Chen, Cuixian. (2018). YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers. 2503-2510. 10.1109/BigData.2018.8621865.

6 Appendix

6.1 Preparing Model

```
git clone https://github.com/thtrieu/darkflow.git
cd darkflow
sed -i -e 's/self.offset = 16/self.offset = 20/g' darkflow/utils/loader.py
curl https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/yolov2-tiny.cfg -o
    yolov2-tiny.cfg
curl https://raw.githubusercontent.com/pjreddie/darknet/master/data/coco.names -o label.txt
flow --model yolov2-tiny.cfg --load yolov2-tiny.weights --savepb
```

6.2 Converting Model into tflite

```
tflite_convert \
  --graph_def_file=built_graph/yolov2-tiny.pb \
  --output_file=built_graph/yolov2_graph.lite \
  --input_format=TENSORFLOW_GRAPHDEF \
  --output_format=TFLITE \
  --input_shape=1,416,416,3 \
  --input_array=input \
  --output_array=output \
  --inference_type=FLOAT \
  --input_data_type=FLOAT
```