DEGREE PROJECT IN TECHNOLOGY,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2018

Performance analysis of convolutional neural networks implemented with TensorFlow Lite and TensorFlow Mobile

Juan Luis Ruiz-Tagle Oriol

Oskar Henriksson *(Supervisor)*
Anne Håkansson *(ICT Examiner)*

KTH ROYAL INSTITUTE OF TECHNOLOGY
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Authors

Juan Luis Ruiz-Tagle <jlrto@kth.se>
Information and Communication Technology
KTH Royal Institute of Technology

Place for Project

Stockholm, Sweden

Examiner

Anne Håkansson Software and Computer Systems ICT
KTH Royal Institute of Technology

Supervisor

Oskar Henriksson
Head of Recruitment
Slagkryssaren

Abstract

TensorFlow is an open-source library developed by Google for machine learning applications. It can be run in different environments such as desktop computers, servers or even browsers. For running TensorFlow on mobile devices there exist TensorFlow Mobile and the recently launched TensorFlow Lite. The latter is an improved version of the former, which claims to have several performance advantages over its predecessor and is meant to substitute it on the long run. The goal of this study is to test and compare the performance of this two libraries when running a convolutional neural network trained to analyze pictures of credit cards. An application was programmed to run the same network with both versions of TensorFlow and installed on several devices to measure their performance. The collected empirical data showed that TensorFlow Lite's performance is still lower than its counterpart's when running convolutional neural networks. This is due to the early stage of development of the library, which is not optimized enough yet.

Keywords

Machine Learning, Tensorflow Lite, Convolutional neural networks, Performance

Contents

1 Introduction

1.1 Background

Machine Learning has been a research field in Computer Science since the 1980's [1], but it has experienced a considerable major breakthrough during the last 5 years. The emerging of novel machine learning techniques combined with the strength of the powerful processors present in modern computers has proven to give a lot of surprisingly positive results.

Google has become a pioneer company and an international reference in the field of machine learning. On November 2015 they released TensorFlow to the public , a software library specifically designed to build machine learning applications. TensorFlow is currently available in numerous platforms, like desktop computers, mobile devices and browsers [2]. To be able to run TensorFlow models on iOS and Android devices they initially developed TensorFlow Mobile. Nevertheless, a reimplementation of the TensorFlow Mobile framework produced TensorFlow Lite, which has several advantages over its predecessor: it has higher execution speed, needs fewer dependencies and it is lighter in terms of memory usage. TensorFlow Lite is still in developer preview, so it is not optimal yet and does not cover all use cases **??**.

1.2 Stakeholders

Slagskryssaren is a tech agency based in Stockholm which provides with software solutions to their clients by building digital services based on the most recent available technologies [3]. At Slagkryssaren there is a research project which aims to build an API that receives pictures of credit cards and returns the data printed on them (that is the card number, expiry date, etc). To extract the information they use a convolutional neural network running in TensorFlow, but they are interested in making the API available for mobile devices, so the natural step was to reimplement their model with the mobile versions of Tensorflow. I have been in contact with them regularly throughout the project and they have guided me and

solved my questions at all times.

1.3 Problem

When Google presented TensorFlow Lite in May 2017 [4], it drew my attention enormously for its claimed promising performance advantages. I thought that it would be interesting to make a performance comparison between this framework and its predecessor TensorFlow Mobile in the specific ambit of convolutional neural networks (the term convolutional neural network will be referred from now on as CNN). This would be achieved by developing an Android test application which would run the same input data through two versions of the same graph, one exported with TensorFlow Lite and the other with TensorFlow Mobile. The problem statement of this thesis is: How much does the performance of a CNN improve when executed as a TensorFlow Lite model compared to its execution as a TensorFlow Mobile model?

1.4 Purpose

The purpose of this report is to present some theoretical background and then describe the investigation process which has been followed during the bachelor thesis. This consists, firstly, in providing an accurate description of what CNN's are, as well as some general information about TensorFlow and its available implementations for mobile devices. Secondly, it will introduce the problem statement and will outline how the two models were compared to each other in terms of performance. Finally, it will illustrate the results of this comparison and will draw some conclusions.

1.5 Goals

This project has several goals. To begin with, its initial goal is to study and get acquainted with the whole TensorFlow framework, specifically with CNNs and the Lite and Mobile versions of the library. The next goal is to export the Tensorflow graph done at Slagkryssaren as both Tensorflow Lite TensorFlow Mobile model files. With these two models exported, the succeeding step is the development of

a deliverable test application in Android which runs the models over a sample of data images. Then the app is to be installed and tested on several devices with sets of pictures of different dimensions, and the data produced by each device is to be collected. Finally, the ultimate goal is to interpretate this data and to draw some conclusions according to the problem statement previously presented.

1.5.1 Benefits, Ethics and Sustainability

The project has been benefitial for both the developers at Slagkryssaren and myself since it has deepened our understanding of the TensorFlow framework and the possibilities it offers to mobile devices. The programmed app as a deliverable might be useful in the future to compare the performance of other CNN models running with TensorFlow Lite and TensorFlow Mobile since very few modifications in the code would need to be done to adapt the application to work with a new graph.
An ethical issue appears regarding the management of the sensitive information which the models could extract from the credit cards pictures. For now the CNN graph is trained to generate an output image where this sensitive information appears located and highlighted (a more thorough explanation will follow later on), but it is still only an image what is being produced, so there are not any apprehensions which one should worry about. Of course, if a further development of the project is done arriving to the point where the data is totally extracted from the image and rendered in form of text some measures should be taken to assure that the privacy of the users is not violated and there are not any misuses of such sensitive information.

1.6 Methodology

The methodology chosen to perform a scientific research is a crucial matter since it guarantees the accuracy and reliability of the results and conclusions drew. S. Rajasekar provides an accurate theoretical explanation of the nature of methodology and research methods. ”Research methods are the various procedures, schemes, and algorithms used in research. All the methods used by a researcher during

a research study are termed as research methods. They are essentially planned, scientific and value-neutral. They include theoretical procedures, experimental studies, numerical schemes, statistical approaches, etc. Research methods help us collect samples, data and find a solution to a problem. Particularly, scientific research methods call for explanations based on collected facts, measurements and observations and not on reasoning alone. They accept only those explanations, which can be verified by experiments. Research methodology is a systematic way to solve a problem. It is a science of studying how research is to be carried out. Essentially, the procedures by which researchers go about their work of describing, explaining and predicting phenomena are called research methodology. It is also defined as the study of methods by which knowledge is gained. Its aim is to give the work plan of research." [5].

Quantitative methods have been chosen to compare the performance of the two TensorFlow frameworks for mobile devices. The measurement which will be collected is the inference time for the CNN models to run, given exactly the same input. This measurement will be done setting timers in the device's processor through the Java API before and after the execution of the CNN, to calculate the time interval subsequently. These timers are accurate enough to provide a realistic measure which lets a comparison between measurements be made. A deductive method will be utilized to reach these conclusions from the collected data.

1.7 Delimitations

The study has been delimited in a subsequent manner. To test the performance of the TensorFlow Lite framework in comparison with its former version TensorFlow Mobile, the specific context of convolutional neural networks has been chosen. No other types of networks or graphs will be considered in this study. As well, there exist delimitations in the number of devices selected to run the performance tests. Four Android devices have been used for testing purposes, namely a Samsung Galaxy S4, a Samsung Galaxy S5, a Samsung Galaxy S8 and a Motorola Moto G5 Plus. These telephones where chosen because they represent different class categories. Some of them are modern flagships while others belong to a middle-class

category.

1.8 Outline

The structure of the next sections of the report is the following: in the upcoming section, the theoretical background concerning convolutional neural networks and TensorFlow Lite and Mobile will be presented. Then the chosen methodologies for performing the research will be explained. After that, the whole research process will be accurately described, together with the results gotten. Finally, some conclusions will be drawn.

2 Theoretical Background

After the concise background description given in the previous section, a more elaborated explanation of the theoretical background of this thesis project will be furnished in the following paragraphs. First, artificial neural networks will be briefly introduced and then, more specifically, convolutional neural networks will be explained with a higher level of detail. Then the TensorFlow framework will be presented, as well as both of its mobile versions TensorFlow Lite and TensorFlow Mobile. These will be compared and their relation described.

2.1 Artificial Neural Networks

Artificial neural networks are an extremely important branch in the Artificial Intelligence and Machine Learning research fields. Neural networks are systems of interconnected nodes which are programmed to find patterns in data by themselves after having gone through a training process, which consists of feeding the network with a large number of samples. This process tries to mimic the way the brain operates, and that is where its name comes from: each node simulates a neuron which sends pulses to the other ones in the network. Neural networks have been a matter of research for more than 40 years now [6], therefore they exist in the form of countless types and classes. The main procedure followed by neural networks to make an output estimation for new input data is described below, and more specifically the functioning mechanism of CNN's will be presented in the following section more thoroughly.

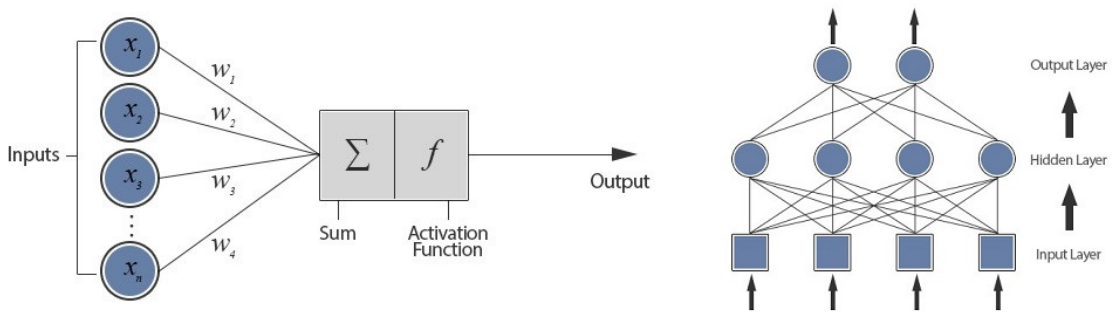


Figure 1: Image X. A perceptron and a neural network

In the images in Figure 1 we can see the basic structure of an ANN (artificial

neural network). The left image depicts a node, the basic unit of the network. Each node performs a dot product between the input and weight values, that is, it takes the input values ($x_1 ... x_n$), multiplies each one by its respective weight ($w_1 ... w_n$) and sums everything up. The result of the dot product is passed to the activation function. If the result of this mathematical function is above a certain threshold, the node will fire and send an output value. In the right image can be seen a set of nodes structured in layers, with 4 in the input layer, 4 in the hidden layer and 2 in the output layer. The input values are placed in the input nodes, which simply refer them to the nodes at the hidden layer. Each hidden node computes an output and forwards it to the output layer nodes, which make a final computation and generate a result. Training the network consists of finding the best values for all the weights which optimize the results for most given cases. The presented example is very simplistic, however, ANNs can have many different structures, with several hidden layers and other elements, like a bias value.

2.2 Convolutional Neural Networks

Convolutional neural networks are a specific kind of ANN usually used in computer vision and they usually take an image as an input. The general idea is that in each layer of the network the image is shrunken and processed by the activation functions in the neurons, and a new one is outputted which has partially abstracted any patterns found. As the image goes from layer to layer, it becomes more abstract until it is finally tagged as belonging to some class.
Regular ANNs might work fine in small images, but don't scale well to bigger ones. "For example, an image of more respectable size, e.g. $200 \times 200 \times 3$ (3 stands for the RGB values of each pixel), would lead to neurons that have $200 \times 200 \times 3 = 120000$ weights. Moreover, we would almost certainly want to have several of such neurons, so the parameters would add up quickly. Clearly, this full connectivity is wasteful and the huge number of parameters would lead to overfitting." [7]
Since it is not computationally affordable to establish so many connections between nodes a new structure for the network has to be designed, with a new arrangement of the layers. Each layer has neurons disposed in 3 dimensions: width, height, and depth. Each neuron in a layer is only connected to a small group of

neurons in the previous layer, which refers to adjacent pixels in the original image. What each layer does is to transform the 3D input volume to a 3D output volume of neuron activations [7]. [Figure 2]

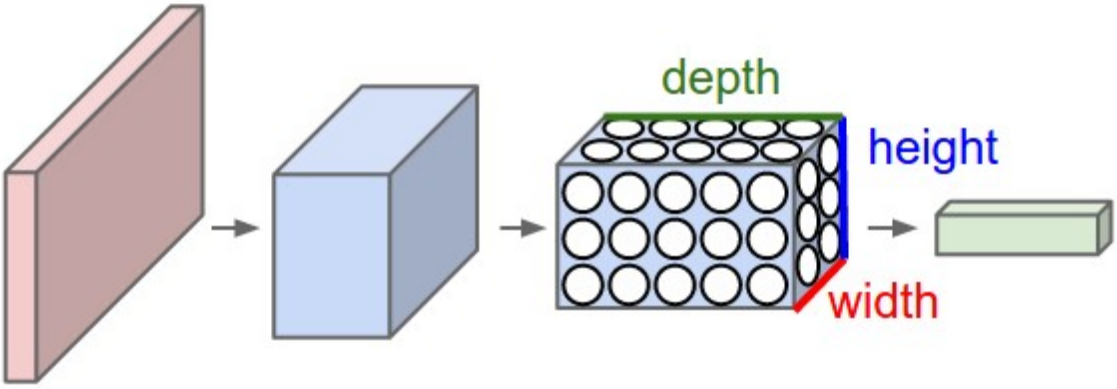


Figure 2: A CNN transforms the 3D input volume from layer to layer.

2.2.1 Types of layers

- **Convolutional layers**: These layers are the principal building blocks of the CNN. The way they operate is by applying a set of filters to the input image trying to recognize patterns. These filters are way smaller than the input image, (square matrices of 5×5 pixels, for example) so they have to slide or "convolve" from left to right and top to bottom along the whole image moving usually one pixel at a time (stride of 1). The dot product is computed between each filter and the portion of the image that is currently on focus, so the more similar they are to each other the higher the result will be and the greater influence it will have in the next layer's convolutions [7]. [Figure 3]

- **Pooling layers**: the function of the pooling layers is to gradually reduce the width and height of the input 3D volume as its depth keeps increasing due to the filtering process. The most common pooling method is applying a filter of 2×2 pixels with a stride of 2 using the *MAX* operation. This means that the image is divided into squared groups of 4 pixels and the maximum value of each group is taken, discarding the other three pixels, so its area is reduced by 3/4. [Figure 4]

- **Fully connected layer**: All the nodes in the last layer have connections to all the activation inputs in the previous one. At this point, the image's dimensions have diminished and it has become small enough through the pooling process to feasibly compute a connection from each node to each pixel [8].

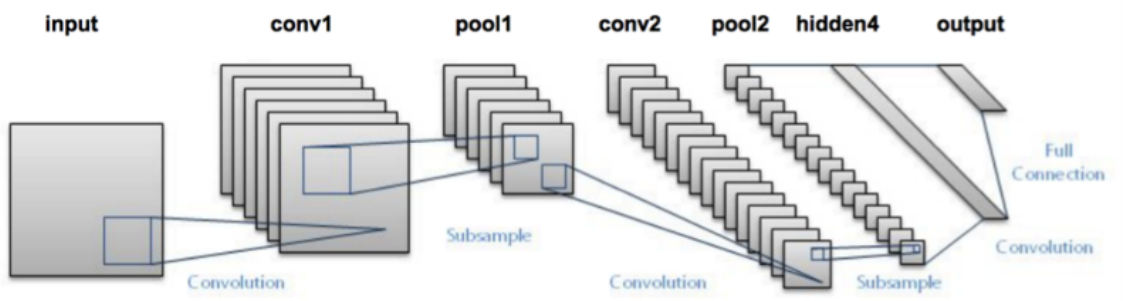


Figure 3: Filter sampling at convolutional layers

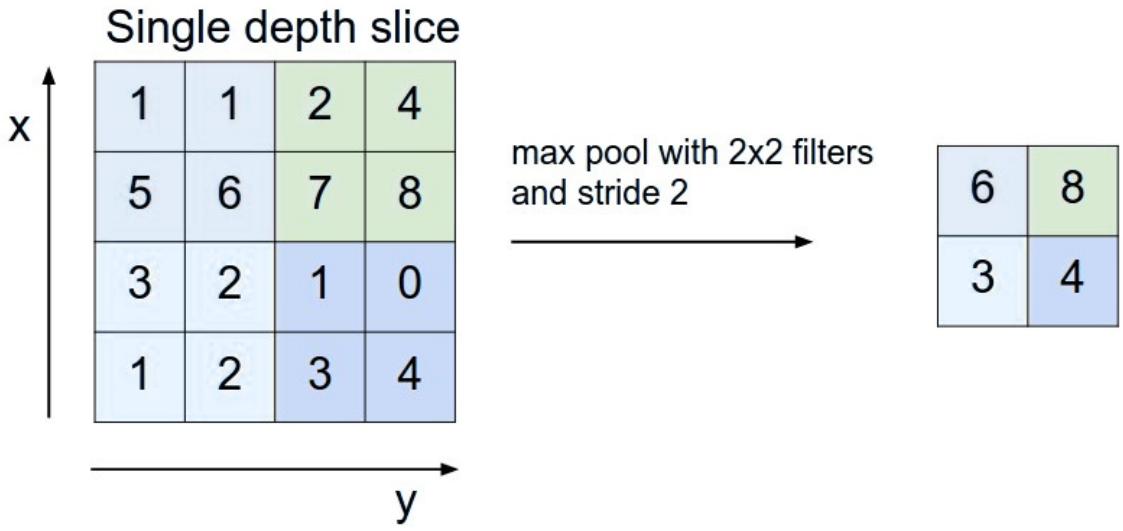


Figure 4: *MAX* operation usage at pooling layer.

2.3 TensorFlow

TensorFlow is a machine learning library developed by Google to satisfy their needs of systems capable of building and training neural networks [9]. Today it is used both for investigation purposes as well as in many of Google's own products. TensorFlow has been continuously expanding since it's very beginnings. It was

open sourced in 2015 and now the library is implemented in several programming languages and optimized for its use in different domains. To run TensorFlow applications in mobile devices Google released first TensorFlow Mobile, which later evolved into TensorFlow Lite. These two frameworks will be described and contrasted, since the main goal of the project is to compare their performance at the task of running CNN's.

2.3.1 TensorFlow Mobile

Running ANNs is a very computationally demanding task which has always been entrusted to powerful computers, but Google planned to take a step further and make the less powerful mobile devices capable of running them as well, since this would open a door with lots of potential. TensorFlow Mobile was developed to be able to run TensorFlow graphs on iOS and Android devices **.** When a TensorFlow graph is trained and ready for deployment it can be exported as a TensorFlow Mobile model with the protobuffer file format. A protobuffer is a file which contains all the relevant information about a graph [10]. TensorFlow graphs are defined by their overall structure and the values of each of the weights. This two parts of the graph are usually stored in different files for convenience reasons, but they need to be put together into a protobuffer file to be usable by TensorFlow Mobile on a mobile device. There are two types of protobuffer files: text format (*file.pbtxt*) which is a human-readable form, and binary format (*file.pb*) which is not readable but much smaller file.

2.3.2 TensorFlow Lite

TensorFlow Lite is a natural evolution of TensorFlow Mobile, meant to substitute it in the long run. It was first released in November 2017, and it is still in a development stage. It presents many advantages over his predecessor: it generally provides better performance and a smaller binary file, but it doesn't cover all use cases yet. "TensorFlow Lite uses many techniques for achieving low latency such as optimizing the kernels for mobile apps, pre-fused activations, and quantized kernels that allow smaller and faster models" [11]. TensorFlow Lite's file format is the FlatBuffer (*file.lite*), an "open-sourced, efficient cross-platform serialization

library. It is similar to protocol buffers, but the primary difference is that FlatBuffers does not need an unpacking step to a secondary representation before you can access data, often coupled with per-object memory allocation." [11]. It has also the capability of interacting with the Android Neural Networks API, designed specifically to boost machine learning computations.

3 Research and probing

3.1 Starting point: research and familiarization

The first natural step in the actual work was to get familiarized with the theory and concepts behind CNN's as well as with the software tools I would be working with, namely TensorFlow. I spent quite some time reading about the functioning mechanism behind CNN's and also TensorFlow Mobile's and Lite's documentation. As well, I had to get acquainted with the current state of the project at Slagkryssaren.

The project in which Slagkryssaren was working on consisted, as aforementioned in the introduction, in developing a TensorFlow graph which could analyze a picture of a credit card and return the information printed on it. This project was still in an early phase when it was first shared with me. The TensorFlow graph which they had written at the time consisted in a CNN which could remove the background noise and return a black and white image where only the credit card number could be seen. To train this graph they had programmed a credit card image generator which produced fairly realistic pictures. The graph had been trained with it and the results were accurate enough. This convolutional network already served for my purposes, so we decided to export it as both a TensorFlow Lite model and TensorFlow Mobile, to compare their performance afterward. Figure 5 shows the structure of the CNN being used. It has the typical structure used in CNN's, with convolutional layers followed by pooling layers iteratively. It hasn't got a fully connected layer in the end though since the output of the graph is not yet meant to make a classification, but still generates a picture.

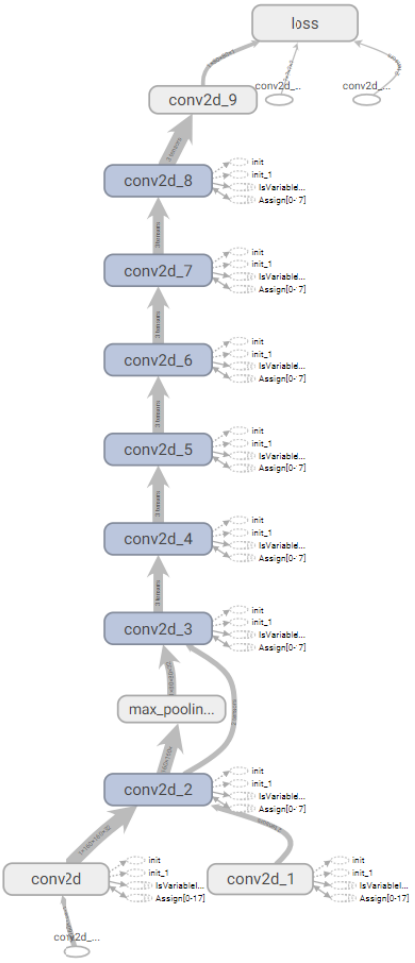


Figure 5: Graph of the CNN used on the credit card images

3.2 Exporting the graph

After the research done it was time to export the CNN as both a TensorFlow Mobile model and a TensorFlow Lite model to be able to use them in an Android application. The graph had to be ”frozen” first, that is, all the variable values for the weights had to be converted into inline constants [12]. The function *freeze_session()* contained in *freeze_graph.py* in the TensorFlow library allows us to do this. Then it is time to write the frozen graph into a file, and this will be done in different ways depending on the TensorFlow version which is going to run it and on the output format desired for the file.

To get a model runnable by TensorFlow Mobile there were different possible output formats for the file, but I opted for the raw protobuffer format (with extension *.pb*). The function *write_graph()* is invoked, specifying the path and the output format of the model, and it generates a file. For the TensorFlow Lite model, the process followed is slightly different, since there is an intermediate step between freezing and writing the graph to a file. The frozen graph has to be converted first to a TensorFlow Lite graph using TOCO (TensorFlow Lite Optimizing Converter). This is done by invoking the function *toco_convert()*. The resulting graph is then written to a file with flatbuffer format (*.lite*).

3.3 Development of testing application

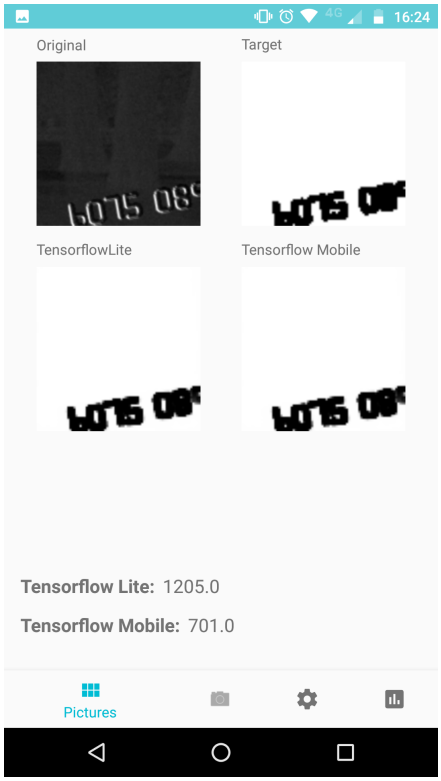
Once the two models had been retrieved the development of a test application could start. The application’s purpose is to be able to quickly test the relative performance of both models by running them over a set of images on different devices. The time which takes to run the models for each image is stored and displayed in a nice graph to get an overview of the performance of each model. All this inference data can be easily exported from the app to a text file for posterior analysis. The development of the Android application took a considerable portion of the work time used for this project. My personal experience with the Android environment

was somehow limited, and the TensorFlow Lite and Mobile dependency libraries for Android were totally new to me. The app was divided into different activities:

- **MainActivity**: runs the models and displays a grid with the pictures to be analyzed. When any of the pictures are clicked, the models are run on that picture and the result is displayed in DisplayActivity. [Figure 6b]
- **DisplayActivity**: after running the models it plots the picture, the target (expected result), and both of the predictions of the TFLite and TFMobile models, together with their respective inference times. [Figure 6a]
- **StatsActivity**: plots a chart where it is represented the inference run time of both models for each picture. Each time a model is run on a picture, a new point is added to the chart. [Figure 6d]
- **SettingsActivity**: used to configure the settings for the test, like the number of images selected to run the model on, which model or models to use and the usage of the Neural Networks API if available. The data on the chart can be exported from here to a text file, or totally erased as well. [Figure 6c]
- **CameraActivity**: lets the user take a picture with the device camera and run the models on it.

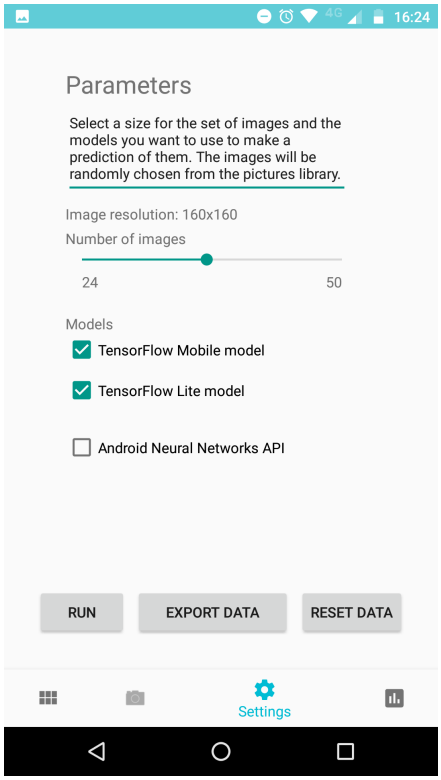
A parent Model class was also implemented with some abstract methods, from which specific TfLiteModel and TfMobileModel classes were extended. An Adapter class for the picture grid and a BaseActivity where implemented as well.

TfMobileModel and TfLiteModel classes have similar implementations of the *predictImage()* method, which is used to actually run the model on an image. This function converts the image in Bitmap format to a float array, starts then a time counter, feeds the float array to the model, stops the timer when the computation has finalized and reconstructs a Bitmap image from the output float array. Both implementations follow.

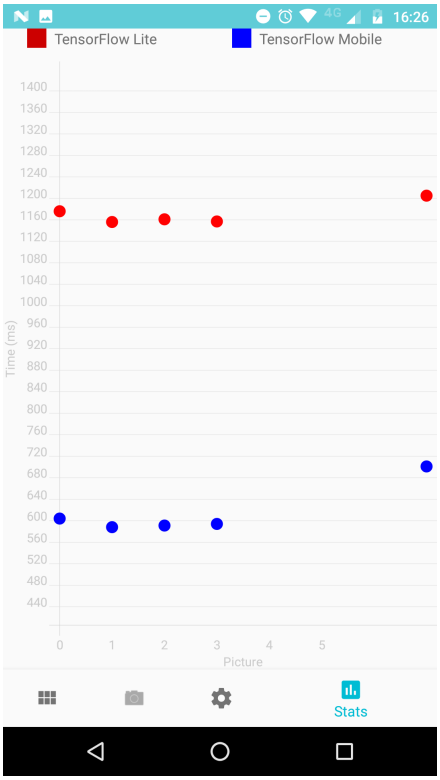


(a) DisplayActivity

(b) MainActivity



(c) SettingsActivity



(d) StatsActivity

Figure 6: Principal activities in test application

Listing 1: code in *predictImage()* used for TfMobileModel

```
inputData = convertBitmapToFloatArray(bitmap);
long startTime = SystemClock.uptimeMillis();
infInterface.feed(inputName, (float[]) inputData, 1, IMAGE_DIM_X, IMAGE_DIM_Y, 1);
infInterface.run(new String[]{outputName}, true);
infInterface.fetch(outputName, (float[]) outputData);
long endTime = SystemClock.uptimeMillis();
Bitmap outputImage = convertFloatArrayToBitmap(outputData);
```

Listing 2: code in *predictImage()* used for TfLiteModel

```
inputData = convertBitmapToFloatArray(bitmap);
tflite.setUseNNAPI(neuralAPI);
long startTime = SystemClock.uptimeMillis();
tflite.run(inputData, outputData);
long endTime = SystemClock.uptimeMillis();
Bitmap outputImage = convertFloatArrayToBitmap(outputData);
```

The principal difference between both approaches is that the TensorFlow Lite execution of the model (listing 2) is atomical and happens within one function call while TensorFlow Mobile (listing 1) subdivides the whole operation in feeding the model with data, running it, and fetching the results. The input vector differs between both implementations, being a one dimensional float array (float[]) for TensorFlow Mobile and a four-dimensional float array (float [][][][]) for TensorFlow Lite. When the Android application was completed and fully operational several tests were run in a handful of devices. The results of this tests are exposed in the following section.

4 Results

4.1 Presentation of collected data

Three sets of 50 images, with dimensions 160×160, 240×240, and 320×320 pixels were used to perform the tests. They were created with the credit card image generator that Slagkryssaren provided me. The application was compiled and run with each set of images on four different Android devices, namely a Samsung Galaxy S8, a Samsung Galaxy S5, a Samsung Galaxy S4 and a Motorola Moto G5 Plus. The reason to test the models with several image sizes was to check how the inference time grows in relation to the volume of pixels that have to be processed. The processors on the selected devices to run the application represent appropriately the spectrum of processing power. Some of them are high class whereas others are more outdated. They run as well different versions of the Android operating system.

Table 1 presents the average time taken for each model to run over the images of different sizes for each telephone. Surprisingly, the time taken for TensorFlow Lite to run is much longer than TensorFlow Lite. In table 2 the proportion between both model's inference time is calculated. In the devices running more recent versions of Android, the proportion stays around 2.0, while in the ones with an older operating system it can go up to 2.9.

	160x160 px		**240x240 px**		**320x320 px**	
	TfMobile	TfLite	TfMobile	TfLite	TfMobile	TfLite
Moto G 5+	577,66	1148,74	1276,94	2562,28	2259,64	4542,26
Samsung S4	516,9	1397,58	1307,1	3562,82	3360,74	9679,18
Samsung S5	462,9	1006,38	931,6	2212,34	2587,76	5777,78
Samsung S8	182,44	371,52	410,12	814,16	757,34	1494,28

Table 1: Average inference time on each device for the three sets of images

	Android version	Proportion 160px	Proportion 240px	Proportion 320px
Moto G 5+	*7.0*	1,99	2,01	2,01
Samsung S4	*4.4.2*	2,70	2,73	2,88
Samsung S5	*5.0.1*	2,17	2,37	2,23
Samsung S8	*8.0*	2,04	1,99	1,97

Table 2: Proportion between TensorFlow Lite and TensorFlow Mobile average inference times.

4.2 Interpretation of results

The most outstanding result probably is that the TensorFlow Lite model is considerably slower than its counterpart on TensorFlow Mobile. Theoretically, TensorFlow Lite is claimed to be an improved version of its predecessor, built from the ground to be faster and more efficient. However, it has been empirically tested that this is not the case, at least for convolutional neural networks. As it has been explained, the TensorFlow library is designed for a wide variety of machine learning graphs, and not only CNN's, so this lack of performance doesn't necessarily apply to other TensorFlow Lite operations.

To assert that the inference time stamps were being correctly measured I used the Android Profiler available in Android Studio. This tool monitors in real-time the activity of an application, and it helped to reassure that the *tflite.run(inputData, outputData)* function which can be seen in listing 2 was really being executed for an unexpected long time. After some research, I arrived at the conclusion that the TensorFlow Lite library is in an early development stage. It was first released in November 2017 [4], and even though the engineers at TensorFlow are making great efforts to match the already existing TensorFlow Mobile the library is far from being completed and optimized. When running the app with various releases of TensorFlow Lite which have taken place during the first semester of 2018 I have found substantial differences in its execution time, so the efficiency problems present in the last TensorFlow Lite stable release will probably be solved in the near future.

Other interesting results come when contrasting the inference time between devices. Figure 7 and figure 8 plot the inference time of each device in relation to the amount of work (number of processed pixels) in TensorFlow Lite and Mobile

respectively.

Firstly, the Samsung Galaxy S8, the device with the most powerful processor, totally overthrows the rest, taking as little as 182 ms to run the TensorFlow Mobile model. A straightforward and obvious conclusion can be drawn: the more processing power a telephone has the faster it will run a TensorFlow model. But not everything depends exclusively on the hardware, the device's software has also an important impact to be considered. This can be noticed by comparing the inference time taken by the Samsung Galaxy S5 and the Moto G5 Plus. The former has a better processor with a higher clock speed (2,5 GHz with 4 cores) but runs an older operating system (Android 4.4). With the latter happens exactly the opposite, its processor is not that fast (2,0 GHz with 8 cores) but has Android 7.0 installed. It can be appreciated in figures 7 and 8 how with a smaller workload the Samsung S5 accomplishes lower inference time, but when the size of the image increases the Moto G5 Plus performs better since its software is more prepared.

In general, the devices with a more modern operating system suited for this kind of tasks (Samsung S8 and Moto G5+) show to have a linear evolution in terms of performance when the image size increases. In contrast, the older devices with a more outdated operating system show to have an exponential evolution of the inference time. They are incapable to manage big workloads.

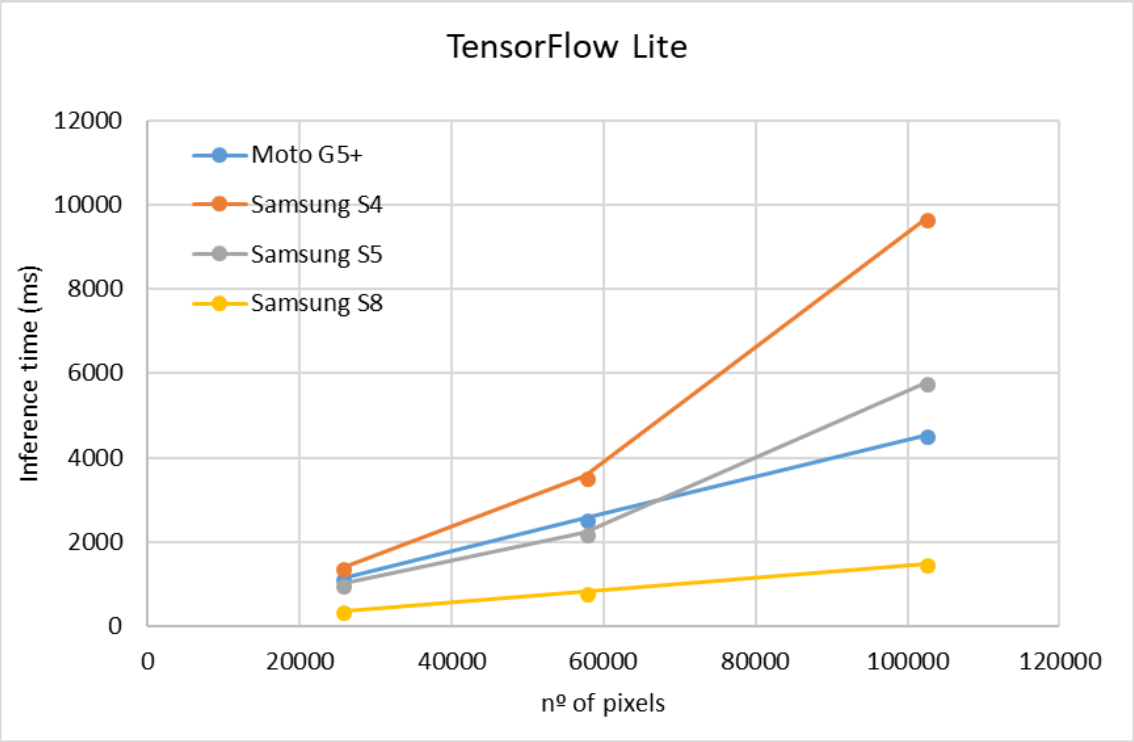


Figure 7: Performance comparison of TF Lite across devices

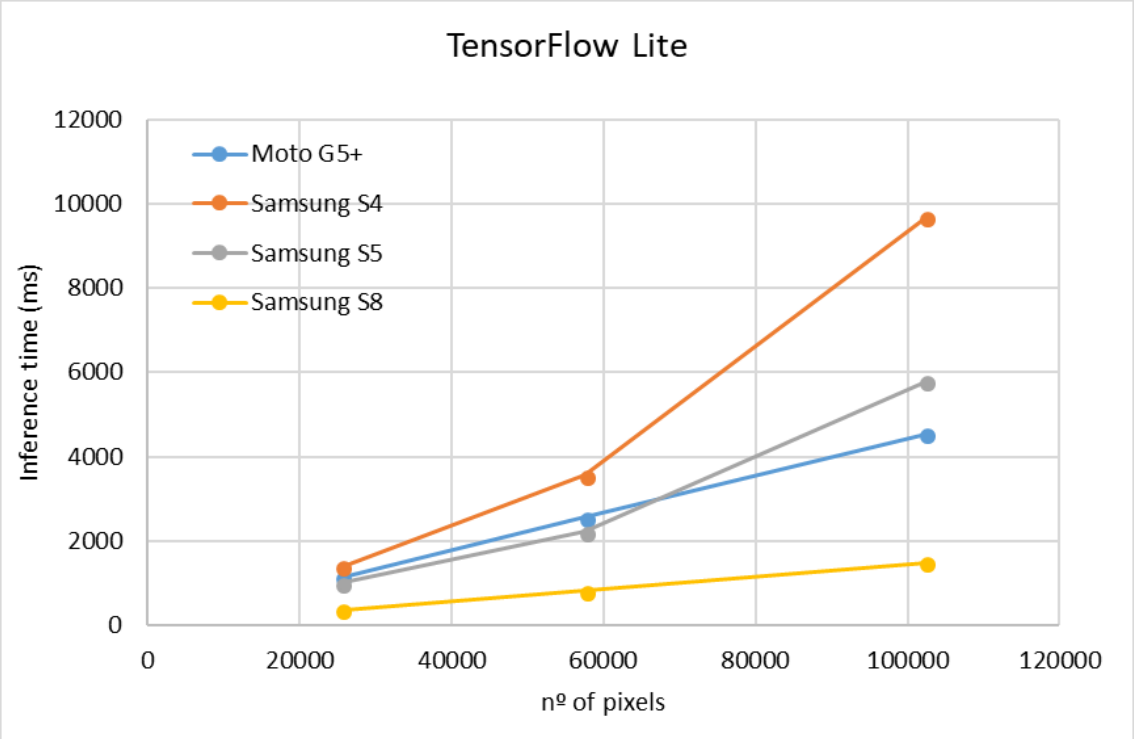


Figure 8: Performance comparison of TF Mobile across devices

5 Conclusions

5.1 Summary of conclusions

After obtaining and analyzing the results of this study in the form of empirical data some conclusions are ready to be drawn. First of all, as aforementioned, convolutional neural networks are not yet optimized for TensorFlow Lite, which makes them run slower than their equivalent in TensorFlow Mobile. This is due to the early development stage in which TensorFlow Lite is at the moment. The second conclusion is that both hardware and software affect the performance of the CNN execution. A more powerful processor will drop the inference time of the network, and a more recent version of Android will handle greater amounts of data with ease, making the execution time grow linearly with the network's workload. In contrast, the inference time increases exponentially with the workload in devices with an older operating system.

5.2 Future work

Subsequent future work could be done on this matter, by waiting until a stable and efficient version of TensorFlow Lite is released and making the tests again to confirm that it has performance improvements. During this research, an attempt was made of performing the tests in a Pixel 2 phone, but the tests failed due to technical problems. Tests on this device were of special interest for its capability of using TensorFlow Lite with NNAPI (Android neural networks API) which is designed to run machine learning models. This chip would enormously boost the performance of the CNN running in TensorFlow Lite, and it would be fascinating to test how far can the NNAPI go.

References

[1] S. Harnad, "The annotation game: On turing (1950) on computing, machinery, and intelligence (published version bowdlerized)," 2008.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning.," in *OSDI*, vol. 16, 2016, pp. 265–283.

[3] (2018). Slagkryssaren, [Online]. Available: https://slagkryssaren.com/ (visited on 05/30/2018).

[4] L. Matney, "Google shares developer preview of tensorflow lite," *Google Open Source Blog, Available at least as early as Jul*, [Online]. Available: https://techcrunch.com/2017/11/14/google-releases-developer-preview-of-tensorflow-lite/?guccounter=1 (visited on 11/15/2017).

[5] S Rajasekar, P Philominathan, and V Chinnathambi, "Research methodology," *arXiv preprint physics/0601009*, 2006.

[6] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[7] A. Karpathy. (2018). CS231n Convolutional Neural Networks for Visual Recognition course notes, [Online]. Available: http://cs231n.github.io/convolutional-networks/ (visited on 04/05/2018).

[8] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th annual international conference on machine learning*, ACM, 2009, pp. 609–616.

[9] T. team. (2018). About tensorflow, [Online]. Available: https://www.tensorflow.org (visited on 05/25/2018).

[10] K. Varda, "Protocol buffers: Google's data interchange format," *Google Open Source Blog, Available at least as early as Jul*, vol. 72, 2008.

[11] T. team. (2018). Introduction to tensorflow lite, [Online]. Available: https://www.tensorflow.org/mobile/tflite/ (visited on 05/25/2018).

[12] ——, (2018). Preparing models for mobile deployment, [Online]. Available: https://www.tensorflow.org/mobile/prepare_models (visited on 05/25/2018).