Samenvatting

Deep learning is used more and more for vision applications like image recognition and object detection. With neural networks we can extract more and beter features from images. Unfotunately a lot of these neural networks require a lot of memory and computing power to extract those features. Consequently, development of deep learning applications always requires an internet connection.

Another problem is that the operations of an existing neural network are often not compatibel with the mobile environment where the application is executed. A coding language often used for designing and training neural networks is Python. Thus these neural networks are designed to be executed in a Python environment while Android studio applications are executed in a Java environment. The purpose of this thesis is to research the compatibility between operations of an existing neural network in its original environment and a mobile enviroment so we can implement complex neural networks on a mobile device with a minimal effect on its accuracy. We will study this for recognition systems and detection systems.

Primarily we study recognition systems wich are the least complex. A recognition system is an application where a neural network predicts the identity of an image. The main part of a recognition system is the convolutional neural network. The main building block is the convolution layer which can extract features from a previous layer or image. The purpose of these convolutional layers is to extract more high level features with each new convolutional layer. These high level features can later be classified by a fully connected layer. As recognition system we will be studying the ResNet50 architecture which is an architecture existing of 50 convolutional layers and is frequently used for image classification problems.

Secondly, we consider detection systems which are more complex than the recognition systems. The purpose of detection systems is to localize and detect objects in an image. There are two approaches for detection systems: the two-stage detector and the one-stage detector. Faster-RCNN is a two-stage detector where first regions of interest are extracted from the image. These regions of interest are possible locations where an object can be located. The second part is object classification and bounding box regression for the regions of interest. YOLO is a one-stage detector where the object detection happens in a single time. The image is devided in a grid, where for each grid cell one classification and multiple bounding boxes are predicted. Two-stage detectors are known to be more accurate but slower while one-stage detectors are known to be fast but less accurate.

Subsequently we consider existing models of the TensorFlow and PyTorch framework. Both fra-

meworks are able to convert a neural network to a network that is compatible with a mobile environment using its own library of operations. Each framework has it's own optimalisation methods which are executed while converting to a mobile compatible neural network. Another possibility is to convert the existing model to another framework with ONNX. This method is usefull when the existing model is developed in a framework which doesn't support conversion for a mobile environment. In this way we can export the neural network to ONNX and import the ONNX model to a framework that supports a conversion method. ONNX also has his own framework to implement an ONNX neural network in a mobile environment.

Finally we study the compatibility of operations for different neural network architectures developed in PyTorch and TensorFlow. The network architectures we will study are: ResNet50, Faster-RCNN and YOLO. ResNet50 is a classification network that is also the least complex among the three architectures. The ResNet50 architecture is fully supported to be converted to TensorFlow Lite, PyTorch Mobile and ONNX. TensorFlow gives the possibility to add metadata to a TensorFlow neural network that is converted to TensorFlow Lite. If we add metadata to this TensorFlow Lite model, Android studio will generate the code to implement the model. However the ResNet50 model from PyTorch will be executed faster on a mobile device.

The conversion of the Faster-RCNN model to a mobile model will be more complex. Prior to converting the model to TensorFlow Lite we have to specify the input shape. Otherwise the converter will set the input shape to a width and height of 1. To execute the TensorFlow Lite model of Faster-RCNN in Android studio we first have to specify the shapes of the output buffers, because the TensorFlow Lite converter also changes the output shapes to a width and height of 1. The PyTorch model can be converted to a language independent model but can't be optimized for mobile use because not all operations are supported. We solved this by implementing the not supported operations as an Android studio project. This method of implementing the not supported operations has a negative effect on the execution speed of the model. The operations of both frameworks are fully supported by ONNX with a minimum opset version of 11. For the Android studio implementation we see that there is a maximum file size for ONNX.

We can't find a YOLO model that is pretrained by PyTorch or TensorFlow, but we can find the pretrained YOLO weights. Those weights can be loaded in a model that we configured in TensorFlow or PyTorch. Like the Faster-RCNN model the input shape must be confined in TensorFlow before we convert to TFLite, otherwise de input shapes wil be set to 1. This model can be converted to TFLite without any problems. The output shapes of the converted model will be correct and the model can be implemented in Android studio without additional configurations for the outputbuffers. The PyTorch model couldn't be converted because of the way the weight are loaded. When the YOLO PyTorch model is converted to a model compatibel with a mobile environment the output is always the same. Because of the file size of YOLO model the ONNX model is to large to be implemented in Android studio.

We conlude that TensorFlow supports the most operations for mobile use, but PyTorch mobile is still in its beta phase. We can expect that more operations wil be added to PyTorch. It is obvious that PyTorch wants these operations for android studio because there is already a library for Android

studio. However, this library doesn't combine well with the other PyTorch imports for Android studio. Conversion to TensorFlow Lite and Pytorch mobile has little to no effect on the accuracy. ONNX is compatibel with all the operations we encountered. Although most of the operations are supported since opset version 1, a part of them are only fully supported in later opset versions. The file sizes of the models are still a problem. This is observed for the ONNX Android implementation for Faster-RCNN, where we get an error that there is not enough memory for execution. A possible solution for this problem can be the quantisation and weight sharing method which should be subject to further studies.