# PROJECT PROPOSAL FOR M.SC. ARTIFICIAL INTELLIGENCE THESIS RESEARCH

## 1. Information

### 1.1. Student personal information.

| | |
|---|---|
| Student name | : Erdi Calli |
| Telephone | : 0683 138 538 |
| Studentnumber | : s4600673 |
| E-mail address | : e.calli@student.ru.nl |

### 1.2. Supervisors.

| | |
|---|---|
| Internal Supervisor | : Luc Hendriks; Neurant |
| Second Supervisor | : Dr. Marcel van Gerven; Radboud University Nijmegen |

## 2. Project description

### 2.1. Title of research project.

*Optimization of Neural Networks for Real Life Applications*

### 2.2. Abstract. (104 words)

Inference on Deep Learning models can be done using mobile phones or cheap computing units. But these devices mostly have very low processing power, limited battery capacity and no cooling. Contrary to these facts, most deep learning models are over parameterized. It has been shown that various techniques can optimize the model structure to reduce the model complexity and required processing power while preserving the model accuracy. Using these optimization techniques we can partially overcome the limitations imposed by those devices. In this project we will apply various techniques to different models and measure the changes in speed and model performance on different devices.

## 2.3. **Project description.** (1606 words)

2.3.1. *Background.* Recent state of the art Deep Learning models are surpassing previously researched models in many fields such as computer vision, automatic speech recognition, natural language processing, speech recognition and bioinformatics. Most of these models are big network structures with many layers (such as 152 layers [HZRS15]) and lots of hidden nodes, therefore requiring massive number of operations for training and for inference. But recent studies show that, some models can be optimized to perform similar (*or better*) in terms of accuracy with huge speed and/or model size improvements. For example, researchers came up with a model called SqueezeNet [IMA+16], based on AlexNet [KSH12]. SqueezeNet is performing as well as AlexNet in object recognition but with 50 times less parameters. Another study [DLT16] optimized the network structure of image super resolution model [DLHT16] to perform 40 times faster with superior restoration quality. These examples give us a glimpse of how state of the art models are open for improvement.

2.3.2. *Aim of the Project.* These state of the art models offer great opportunities for improving lives. For example they can help visually impaired people to be more self reliant: by describing what is around them, by providing an alternative to their guide dogs, by providing social cues like facial expressions or body postures. Similar to these examples, we can come up with a lot of potential applications depending on real time inference. But inference is computationally heavy in most of the state of the art models, that's why it is hard or expensive to integrate them into our daily lives. By diminishing their computational cost, or by optimizing them, we could apply them to cheap computing units (i.e. Raspberry Pi's) or devices that are already a part of our lives (i.e. Mobile Phones) and make them accessible to everyone. This, in return, would create an environment that nourishes new applications of these models, hopefully leading to new startups, economical growth and a better world.

A good example to sketch the problem we are dealing with is the super resolution model [DLHT16]. This model tries to generate a high resolution image from a low resolution one. This model is composed of three phases. The *patch extraction* phase, collects information about the low resolution input and creates a feature map for each pixel. The *non-linear mapping* phase applies deconvolutions to this input and constructs a feature map representing the inferred features of the high resolution image. The *reconstruction* phase uses this information to infer the pixels of the high resolution image.

For example, when we increase the resolution of a $100 \times 100$ image by a factor of 4, it results with a $400 \times 400$ image. Therefore, we have a $400 \times 400$ feature map for the reconstruction phase. If we have 64 features per color channel(3), this gives us $400 \times 400 \times 64 \times 3$ distinct feature representations to work with. Assuming we are using kernels of size 9 in the reconstruction phase, for each resulting pixel we should process $64 \times 3$ feature representations with $9 \times 9$ convolutions, requiring $64 \times 3 \times 9 \times 9$ floating point operations. When we apply it to all pixels, we need approximately 2.5 Billion ($400 \times 400 \times 64 \times 3 \times 9 \times 9$) floating point operations only for the last (reconstruction) phase of this model.

When we have access to very expensive state of the art GPU's, such as Nvidia GeForce GTX 1080[1], 2.5 Billion floating point operations are easy to calculate. A single Nvidia GeForce GTX 1080 GPU is capable of running 8.8 trillion (single precision) floating point operations per second (TFlops)[2], meaning that it could execute such an operation about 3500 times per second. In contrast, when we look at a regular mobile phone CPU, this amount of floating point operations is overwhelming. In our previous experiments, we have observed that the reconstruction phase (with 64 features per channel and kernel size of 9) takes about 7 seconds on a mobile phone with Snapdragon 801[3] processor. This processor contains a 2.5 GHz quad-core CPU.

There is a lot of research related to optimizing similar networks. [AP16] proposes the use of linear convolutions. Their research suggests using 1D (1 by 9 or 9 by 1) convolutions, instead of 9 by 9. According to their experiments, this method speeds up the inference time of VGG model [SZ14] up to 4.3 times.

[WLF16] projected color channels into one channel by using an extra layer. When applied to GoogLeNet [SLJ+15], this technique required 3.4 times less computation while preserving the accuracy.

[TBCS16] proposes using information theory to determine the importance of weights. Based on this information they pruned unimportant weights. Also they have quantized the weights and used the importance of a weight to determine the number of bits to assign for it. Their study only focuses on reducing the model size, and they succeed in that by preserving the accuracy. Similar methods could be used to prune unimportant parameters, adding up to performance.

[WLW+15] proposes quantizing the weights, with the aim of running them on mobile devices. Doing so they speed up AlexNet [KSH12] 4 to 6 times.

These generalized techniques and many others aside, researchers who came up with super resolution model revisited their work in [DLT16]. By investigating the phases, they came up with some modifications to speed up this model. First, they removed the interpolation phase and converted their deconvolution layer to output the high resolution image. Second, they have *shrank* the input feature dimensions of mapping layer, and *expanded back* afterwards. And third, they added more mapping layers but reduced the filter sizes. All those optimizations came up with a new model, that is 40 times faster than the original one and better than the previous version.

[1]As of September 27, 2016

[2]`http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_1080_Whitepaper_FINAL.pdf`

[3]`https://www.qualcomm.com/products/snapdragon/processors/801`

The room for improvement on state of the art models and the opportunities of their real life applications are raising some questions. In this study, we are interested in answering the following;

- Which state of the art models are not usable for real life applications because of computational cost?
- Which state of the art optimization techniques could be used to optimize those models?
- Why does or doesn't an optimization technique work on a model?
- How can we combine different optimization techniques to come up with a more efficient model?
- What are some optimized versions of these state of the art Deep Learning Models?
- Are those optimized models efficient enough to be used for real life applications?

To summarize, our research question is,

> *Can we come up with a generalized method to optimize a computationally heavy Deep Learning model for real life applications?*

Another way of applying a computationally heavy Deep Learning model to real life is relying on Moore's Law and waiting for required processing power to become available. But, as the model gets more popular, the requirement of high computational cost, or inefficiency, will lead to unnecessary electricity consumption and heat generation. In this era of accelerated global warming, as researchers, it is our duty to come up with less computationally heavy models.

2.3.3. *Research Plan.* In this research we're interested in optimizing the computational cost of some pre-trained state of the art Deep Learning models, to be able to apply them to real life. To do so we will divide the project into several phases.

The first phase will consist of collecting interesting state of the art models. The selection of these models will be based on

(1) the coverage of different operation or architecture types(such as convolutions, deconvolutions, LSTMs and etc.),
(2) availability of a pre-trained version,
(3) availability of the codebase,
(4) licensing that allows research,
(5) availability of the dataset[4], and

[4]Important to measure the change in accuracy and some techniques may require re-training.

(6) performance on related platforms.

At the end of phase one, the second phase will begin. In this phase, we will

(1) come up with a technique or,
(2) cherry-pick an optimization technique based on type(e.g. pruning, factorization, operational optimizations, topological optimizations and etc.) and applicability to selected models,
(3) build an understanding of why and how that technique works,
(4) implement the technique on appropriate models,
(5) evaluate the changes, and
(6) go back to 2.

We will keep iterating these steps until the scheduled end of phase two. That way we will be building a basic understanding of various state of the art optimization techniques and implementing them for future use. After this phase, we will begin phase three. In this phase, we will try to optimize the selected models by using meaningful combinations of techniques. We will

(1) Select a model,
    (a) create a combination of techniques that could optimize the model in theory,
    (b) apply that combination to the model,
    (c) evaluate the changes, and
    (d) if the results are satisfying for real life applications, go back to 1, otherwise go back to 1a.

2.3.4. *Evaluation.* To evaluate the effects of an optimization technique or a combination of them, we will measure the changes in

- accuracy,
- number of "floating point" operations, and
- required storage.

To evaluate the applicability on cheap computing units and mobile devices, for each device and optimized model we will measure

- inference time,
- battery usage,
- changes in core temperature, and

- memory usage.

2.3.5. *Tools.* To be able to evaluate these techniques on mobile devices and cheap computing units, we are going to use TensorFlow. TensorFlow provides boilerplate codes to run Neural Network models on different type of devices[5]. Underneath the hood, TensorFlow converts a model to a series of operations and tries to execute them as fast as possible by exploiting low level instructions of a given architecture[6].

We will run models on **Raspberry Pi**(2), **Android** and **iOS** platforms to evaluate their real life applicability.

To implement the optimization techniques and other required code bases, we will use **Python 2.7**. To run TensorFlow models on Android and iOS environments, we will be dealing with a native bridge written in **C** between compiled TensorFlow libraries and the related environment. For Android specific code development, we will use **Java**. For iOS specific code development we will use **Swift**. For Raspberry Pi specific code development, we will use **Python 2.7**. To deploy applications to specific environments, we will use **Bazel**.

## 2.4. Schedule.

| *Start Date* | *Phase* | *Duration* |
|---|---|---|
| *Start* | Phase 1 | 7 weeks |
| *Start* + 7 weeks | Phase 2 | 12 weeks |
| *Start* + 19 weeks | Phase 3 | 12 weeks |
| *Start* + 31 weeks | Phase 4 | 12 weeks |
| *Start* + 43 weeks | Finish | |

[5]https://www.tensorflow.org/mobile.html

[6]Currently TensorFlow is unable to execute operations on mobile GPU's. The lack of OpenCL support is causing this. For more information, see https://github.com/tensorflow/tensorflow/issues/22

## References

[AP16] Jose Alvarez and Lars Petersson. Decomposeme: Simplifying convnets for end-to-end learning. *arXiv preprint arXiv:1606.05426*, 2016.

[DLHT16] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2016.

[DLT16] Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network. *arXiv preprint arXiv:1608.00367*, 2016.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[IMA$^+$16] Forrest N Iandola, Matthew W Moskewicz, Khalid Ashraf, Song Han, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 1mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[SLJ$^+$15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[TBCS16] Ming Tu, Visar Berisha, Yu Cao, and Jae-sun Seo. Reducing the model order of deep neural networks using information theory. *arXiv preprint arXiv:1605.04859*, 2016.

[WLF16] Min Wang, Baoyuan Liu, and Hassan Foroosh. Factorized convolutional neural networks. *arXiv preprint arXiv:1608.04337*, 2016.

[WLW$^+$15] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. *arXiv preprint arXiv:1512.06473*, 2015.