OPTIMIZING DEEP LEARNING INFERENCE: REPLICATION OF ADAPTIVE MODEL SELECTION ON IMAGE CLASSIFICATION [1]

Anel Mengdigali and Vladislav Yarovenko
Department of Computer Science, Nazarbayev University, Nur-Sultan, KZ

ABSTRACT Deep neural networks (DNNs) are leading machine learning techniques with many application domains. However, embedded systems experiencing long inference time and big resource requirements for DNN domains implementations. Compression algorithms are infeasible solutions due to reduced accuracy at a cost of reducing inference time.
This paper presents a replication work on laptops of the experiment about a new, alternative approach to enable efficient execution of DNNs on embedded devices. The examined approach dynamically determines which DNN to use for a given input, by considering the desired accuracy and inference time. We succeeded in making the DNN inference part of this experiment. We apply our investigation to the image classification domain of DNN domains. Unfortunately, we did not achieve a complete reproduction of the paper and were not able to implement the premodel part of the experiment.

1. INTRODUCTION

This paper is about reproducing an experiment of a research study on the topic of Optimizing Deep Learning Inference on Embedded Systems through Adaptive Model Selection [1]. Our main focus was to go through the same creative process, as authors of the original paper did, analyze their resulting source code, and optimize it for laptop usage.

2. MOTIVATION

The presented algorithm is a solution for optimal usage of CPU, memory, and power without affecting any accuracy.

While authors of the original paper established a remote connection between the Desktop Computer and NVIDIA Jetson TX2 module, our goal was to perform the inference using a less complicated setup, which is only one laptop. Additionally, the performance of the code could be improved by using newer versions on CNNs and libraries. Two introduced devices.

3. METHOD

This work consists of two main parts: DNN Inference and Premodel Algorithm:

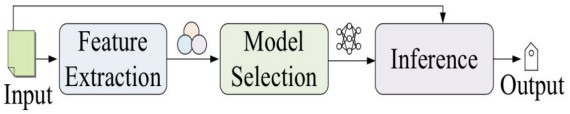

Figure1. Design Structure (Taylor et al., 2019 [1]).

According to Figure 1, the whole algorithm of the original paper can be described in several steps:

1. Three pre-trained Convolutional Neural Networks with different characteristics (low, medium, high accuracy, and inference time).
2. Feature extraction algorithm that analyzes the image's edges, hues, and brightness, and produces the result of each value in the range (0, 1).

Address: Anel Mengdigali, anel.mengdigali@nu.edu.kz; Vladislav Yarovenko, vladislav.yarovenko@nu.edu.kz

3. The premodel algorithm, which chooses the suiting CNN based on the results of the feature extractor. The more complex the image, the more complex CNN should be chosen.
4. The chosen CNN performs the image classification and gives results.

3.1. DNN Inference

The reproduction process required examining and testing several Deep Neural Networks, and particularly Convolutional Neural Networks, which are used for image classification. This includes: MobileNet_v1, ResNet_v1, and Inception_v4. These networks are later used by the premodel algorithm to perform the inference on a particular image dataset.

3.2. Premodel Algorithm

The predictive model is a predictor for the efficient Deep Neural Network (DNN) model at run time. Main measures for this algorithm: high accuracy and fast execution time.

4. METHODOLOGY

Main parts of the experiment to be examined:

1. DNN inference: examining used DNN models such as MobileNet, Inception, and Resnet; and choosing the most efficient networks and using them in the premodel.
2. Premodel algorithm: model selection using multiple KNN models for identifying the most optimal DNN model for the received input based on their inference time.

5. APPROACHES

Two approaches have examined in the reproduction process.

5.1 Top-down

Alongside the paper, authors have also published their working source code, which could be freely used. While they were using several remotely connected devices to establish the inference, such as stationary computers and Jetson TX2, our setup consisted of only one laptop, so the provided code had to be rewritten to fit the current setup. This process required knowledge of several libraries, such as TensorFlow [2] and Scikit-learn [3], which are used primarily for the machine learning; Pyro4, which is responsible for establishing the remote connection between devices; MatPlot [4] and Plotly, to observe results of the inference.

During the process of rewriting the code, Pyro4 library and all of its mentions in the code had to be removed completely, and all other parts had to be changed drastically, which requires a very deep understanding of the mentioned libraries. As we were working to optimize the code, many issues with it have started to appear. The major number of errors have been caused by version mismatching issues as authors of the original paper produced and shared their source code in 2018 and much have changed over the last years. Thus, we had to find ways to fix those nonmatching parts in the source code files to obtain results of the reproduction process after successful full execution. Many logical and syntactical errors that were encountered, but were possible to fix, however, two major issues remained unsolved. The first one was caused by an invalid imported library, and then another was about a used method that was defined neither in the source code files nor in the library. We identified several ways of solving those above-mentioned issues such as examining different versions of implemented high-level language and libraries, using other libraries and methods. However, the former two variants of the predicted solutions failed, which is why this approach was not successful. This was caused by our lack of experience with the above-mentioned libraries and could possibly be fixed with the

Address: Anel Mengdigali, anel.mengdigali@nu.edu.kz; Vladislav Yarovenko, vladislav.yarovenko@nu.edu.kz

right amount of studying and practicing in this field.

5.2. Bottom-up

As the first attempt failed, the approach had to be changed. The next chosen strategy required to start with a simple DNN implementation, and then add the main features of the source code, such as feature extraction, premodel implementation, etc.

The original paper worked with many Convolutional Neural Networks designed for image classification. Each one of them was trained and tested using the ImageNet 2012 validation set. Then, based on the accuracy, inference time, and the number of times the network was used, authors picked three the most efficient networks: MobileNet_v1_100 [5], Inception_v4 [6], and ResNet_v1_152 [7]. In our approach, we decided to choose the same networks, and changed only the TensorFlow version from 1.3 to 2.2, as it provides an easier way to implement these networks. They were then trained and tested using the "cats_vs_dogs" dataset, which serves the same purpose as the ImageNet dataset but has fewer classes which save a significant amount of time during the training process.

Even though we achieved considerably good progress in training and testing of the mentioned networks, we still struggled to implement more important features of a premodel, mainly because the first attempt has taken up about two-thirds of an internship time, which has left us with only two weeks for the second approach.

6. DISCUSSION

Unfortunately, we could not fully succeed in this reproduction work of the research paper-based experiment by implementing all needed code by ourselves. This was mainly caused by the lack of knowledge and experience with used libraries, technology issues, and an insufficient amount of time for such a vast and complicated project. Nonetheless, the reproduction is completely doable and could be finished in about two more months are given. In this period of time, the following steps could be done to finish the work by allocating for each step about two weeks of time:

1. Improve CNN inference implementation to achieve a higher accuracy rate. This could be done by rewriting the code and choosing newer versions of used CNNs. Since the DNN part of our code is almost completed, this step can be finished in the first week;
2. Add several classifiers, such as K-Nearest Neighbor and Decision Tree in order to choose the most efficient one. This part is fully implemented by the authors, so adapting it to our code should probably take about two weeks;
3. Implement the feature extraction algorithm with the same features as in the original paper (brightness, hues, and edge length). Since our team's code is different from the original one, it may take about two-three weeks to fully implement this algorithm.
4. Develop the premodel algorithm that chooses the CNN based on the numerical results of feature extractor. This is the main part of the project, so implementation should take at least three weeks in order to be done as precisely as possible.
5. (optional) Replace all networks and libraries with newer versions. This includes TensorFlow, MobileNet, ResNet, Inception, Scikit-learn. This step may improve the performance of the whole project, as the last versions of CNN's are faster than the previous ones.

Since the bottom-up approach failed mainly because of the lack of time, it is fair to say that with more time the replication can be finished. Additionally, more time could be spent on mastering used libraries in order to deepen our understanding of them.

Address: Anel Mengdigali, anel.mengdigali@nu.edu.kz; Vladislav Yarovenko, vladislav.yarovenko@nu.edu.kz

However, this work was not completely ineffectual, as we became more experienced in machine learning sphere through working directly with the practical part of it. Importantly, a clear understanding of several methods became clear through investigating their used purposes. For instance, we got theoretical and practical knowledge in different spheres, techniques, and algorithms of machine learning such as model selection, training model, feature selection, correlation-based feature selection, k-fold cross-validation, DNN inference, K-nearest neighbors (KNN). Additionally, we got user experience of Jyputer notebook, Google Collaboratory, production of Comma-separated values (CSV) files, Python scikit-learn package, Tensorflow platforms.

We became more educated on the topic of deep learning as a whole and mainly got a good understanding of our research topic. Now, we know more about inner details of the work done in the research study with used algorithms such as model selection, training the prediction model, feature selection, k-fold cross-validation and etc. We started to understand more about how the convolutional neural networks operate as this is the vital architecture behind the prediction model algorithm.

All needed implementations for completion of the experiment are identified for possible further work.

Generally, an experience obtained from this practical knowledge is vital as we investigated the machine learning field more deeply. We have a good practical experience in machine learning sphere through investigating source code and subsequently trying to fix errors and implementing the experiment from the beginning by ourselves based on our second strategy.

It was a very vital experience since the designed algorithm is applicable to all domains of deep learning along with image classification.

7. SUMMARY

Reproduction work was not fully completed: only the DNN inference part of the experiment was implemented.

All needed implementations for completion of the experiment are identified for possible further work.

Skills gained throughout this summer research studies will possibly contribute to our successful completion of the experiment.

The experience got from this replication process are vital: Theoretical knowledge in different spheres, techniques, and algorithms of machine learning such as model selection, training model, feature selection, correlation-based feature selection, k-fold cross-validation, DNN inference, K-nearest neighbors (KNN), and many machine learning libraries and APIs; practical experience with many machine learning methods (k-fold cross-validation, DNN inference, K-nearest neighbors) and libraries (Tensorflow, Scikit-learn); additional user experience with utility software, such as Jupyter Notebook, Google Collaboratory, production of Comma-separated values (CSV) files, etc.

REFERENCES

[1] Marco V., S., Taylor B., Wang Z., & Elkhatib Y. January 2019. Optimizing Deep Learning Inference on Embedded Systems Through Adaptive Model Selection.

[2] Dean, Jeff; Monga, Rajat; et al. (November 9, 2015). "TensorFlow: Large-scale machine learning on heterogeneous systems". TensorFlow.org. Google Research. Retrieved November 10, 2015.

Address: Anel Mengdigali, anel.mengdigali@nu.edu.kz; Vladislav Yarovenko, vladislav.yarovenko@nu.edu.kz

[3] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[4] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.

[5] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[6] Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017, February). Inception-v4, inception-resnet and the impact of residual connections on learning. In Thirty-first AAAI conference on artificial intelligence.

[7] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

Address: Anel Mengdigali, anel.mengdigali@nu.edu.kz; Vladislav Yarovenko, vladislav.yarovenko@nu.edu.kz