Deep Learning for Text Data on Mobile Devices

1st Jakub Sido
Dept. of Computer Science and Engineering
University of West Bohemia
Pilsen, Czech Republic
sidoj@kiv.zcu.cz

2nd Miloslav Konopík
Dept. of Computer Science and Engineering
University of West Bohemia
Pilsen, Czech Republic
konopik@kiv.zcu.cz

***Abstract*—With the rise of Artificial Intelligence (AI), it is becoming a significant phenomenon in our lives. As with many other powerful tools, AI brings many advantages but many risks as well. Predictions and automation can significantly help in our everyday lives. However, sending our data to servers for processing can severely hurt our privacy. In this paper, we describe experiments designed to find out whether we can enjoy the benefits of AI in the privacy of our mobile devices. We focus on text data since such data are easy to store in large quantities for mining by third parties. We measure the performance of deep learning methods in terms of accuracy (when compared to fully-fledged server models) and speed (number of text documents processed in a second). We conclude our paper with findings that with few relatively small modifications, mobile devices can process hundreds to thousands of documents while leveraging deep learning models.**

***Index Terms*—Deep learning, neural networks, mobile computing, CNN, LSTM.**

I. Introduction

This paper deals with topics of machine learning (part of AI) and privacy. We attempt to investigate whether the current technical means enable users to process their data within the privacy of their mobile devices. Current mobile devices are usually well protected with various biometric security authentication methods. If the mobile devices are sufficiently powerful and the software framework mature enough, there might be the possibility to execute machine learning algorithms directly on the devices. This way, users are not forced to transfer their data out of the security of their mobile devices.

We decided to focus on text data. There are already quite a few benchmarks for image processing [1] [1]. Text data are, however, entirely different. The sentences have variable lengths (images can be re-scaled). The models for text also require large matrices of word embeddings [2].

We design our experiments in the Tensorflow Lite, which is a mobile version of Tensorflow [2]. Tensorflow is an open-source machine learning tool focused mainly on artificial neural networks released by Google in 2015. Tensorflow supports conversion of models computed on powerful computers to mobile devices. However, Tensorflow Lite supports only a subset of operations and the models have to be simplified in some cases. A quantization process can further optimize the Tensorflow models (see section IV-C). It significantly reduces the computation time of the models at the costs of small accuracy degradation.

To summarize, in this paper, we attempt to reach the following goals:

- Benchmark the computational performance of the deep learning models on mobile devices.
- Measure the accuracy drop of quantization.
- Examine the support of complex models in the Tensorflow Lite tool on mobile devices. Discover the accuracy penalty of simpler models.

II. Selected Task

For our experiments, we opted for the task of Sentiment Analysis. The goal of the task is to determine the overall sentiment of a given document. For example, a sentence *“A good film all round.”* bears a positive sentiment and *“The film, to put it even more bluntly, is a total bore and would appeal to no one but perhaps those who made the film.”* expresses clearly a negative sentiment.

The task requires some degree of text understanding and the algorithm needs to deal with several NLP[3] challenges such as figurative language (irony, sarcasm, and metaphor), negation, ambiguity (some positive words can be used in a negative context) and others.

III. Training and Evaluation Datasets

A. CSFD Sentiment Dataset

The CSFD dataset consists of 91,381 movie reviews written in Czech from the Czech Movie Database [4]. The reviews are split into three categories according to their star rating (0-1 stars as negative, 2-3 stars as neutral, 4-5 stars as positive). The dataset contains 30,897 positive, 30,768 neutral, and 29,716 negative reviews. 82,244 reviews are used for training and 9,137 for testing. More details about the dataset can be found in [3].

[1] See e.g. http://ai-benchmark.com/.

[2] Word embeddings are in principle large dictionaries of words and their vector representations. The models require vocabularies of more than 10,000 words.

[3] Natural Language Processing

[4] https://www.csfd.cz/

ISBN 978-80-261-0812-2, © University of West Bohemia, 2019

B. *IMDb Sentiment Dataset*

The IMDb dataset [4] contains 25,000 training and 25,000 evaluation movie reviews written in English from IMDb [5]. Because some movies receive substantially more reviews than others, only at most 30 reviews from any movie are included in the collection. Ratings on IMDb are given as star values (1, 2, ..., 10), which are mapped to 0 and 1 (0: $stars \leq 5$, $1 : stars > 6$) to label sentiment categories.

IV. Deep Learning Models

For our experiments, we employ two popular deep learning architectures: Long Short-Term Memory (LSTM) – see section IV-A and Convolutional Neural Network (CNN) – see section IV-B.

A. *LSTM*

LSTM [5] belongs in the recurrent neural network (RNN) class of artificial neural networks. RNNs process input data in sequences and encode them into a hidden vector (memory). LSTMs improves vanilla RNNs by adding an ability to control storing and deleting information from memory. With more precise control of the memory, it can store substantially longer sequences.

In our architecture (see Figure 1), we first transform words into their vector representations using an embedding layer. For the CSFD dataset, we use randomly initialized vectors with a dimension of 60. Since the training part of IMDb dataset is almost three times smaller, we use pre-trained vectors from the FastText tool [6] in this case.

After the embedding layer, we attach two stacked LSTM layers and one fully connected softmax layer.

During the implementation phase of our experiments, we struggled heavily with the conversion to Tensorflow Lite models. Currently, the RNNs implementation is highly experimental in Tensorflow Lite and it seems that it does not support variable sequence lengths. Support of variable sequences is, however, crucial for text data. If we pad sequences, e.g. with an end-of-sequence token, we get approximately 10% drop of accuracy. To solve this issue, we have first opened an issue on GitHub and contacted the community. However, they have confirmed our findings that Tesorflow Lite does not support processing sequences with variable length. We have solved this issue by computing LSTMs for the whole padded sequences. Then we select appropriate end states of the LSTM by an advanced indexing tensor operation (*gather-nd*). Our solution is available on GitHub[6].

We use the maximum length of sequences set to 150 words and the dimension of hidden states is set to 64.

B. *CNN*

Convolutional Neural Networks apply a convolution operation to the input followed by a pooling operation (computing maximum or average on a window of convolution output) [7].

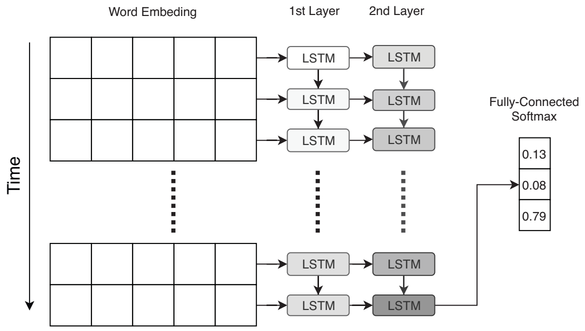

Fig. 1. LSTM architecture.

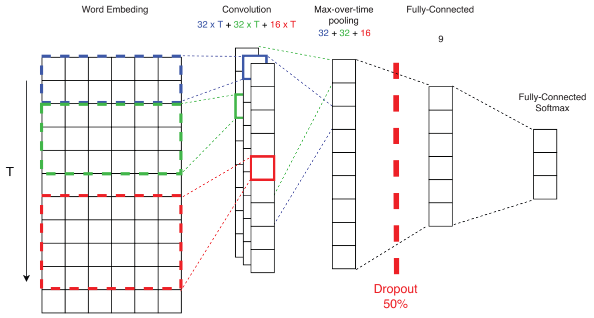

Fig. 2. CNN architecture.

In our architecture, we use the same embedding layer as in the LSTM architecture and add only one convolution layer followed by max pooling. This approach is quite standard for text processing. We use 2D convolution kernels where one dimension is set to the embedding dimension and the other one is set to 2, 3 and 4 respectively. This way the network looks for patterns in 2, 3 and 4 consequent words. We apply 32 2-word kernels, 32 3-word kernels and 16 4-word kernels.

In the end, we connect two fully connected layers, one with Sigmoid activation, the other one with softmax (see Figure 2).

C. *Post-training quantization*

Tensorflow stores the weights in a 32-bit floating point format by default. In order to increase the computation speed, Tensorflow allows converting (quantize) the weights into 8-bit fixed point format. In the post-training quantization approach, Tensorflow uses hybrid operators, which compute the activations in an 8-bit fixed point format. However, the outputs are still stored using floating point.

The post-training quantization process increases model computation speed, decreases inference latency and model size. The costs of quantization consist of reduced accuracy of the model. We measure both computation speed difference and accuracy difference in our experiments.

V. Experiments and Discussion

We conduct experiments where we measure the computation speed of the models (in terms of a number

[5] https://www.IMDb.com/
[6] https://github.com/konopik/tflite-lstm-text

of processed reviews per second), the CPU load of model execution and the accuracy of the models. In our experiments, we distinguish between original and quantized models. We measure both proposed architectures, LSTM based and convolutional neural networks. In Figures 3 – 8, we show the computation speed in solid lines and the CPU load in dotted lines.

For the CSFD dataset, we measure the computation speed on two devices, Huawei p9 lite and Google Pixel 3XL. For the IMDb dataset, we show the results only on Huawei p9 lite.

We can draw several conclusions from the obtained results. First, the CNN networks in Tensorflow Lite are by one order of magnitude faster then LSTMs. Second, the quantization has signification effect of reducing computation costs of the models. In CNNs, this shows in CPU load where the quantized models require roughly half of the CPU allocation when compared to original models. However, it does not transfer to the increased speed of computation, which is comparable for both quantized and original models. In LSTMs, the CPU load is almost identical, but the quantized models are approximately twice as fast. Third, a high-performance mobile device (such as the Google Pixel 3XML we tested) is capable of processing hundreds to thousands of reviews per second. More than enough for handling users' data. Taking into account the rate of technological progress, we can safely expect that processor in today's high-performance models will be a norm soon.

In Table I, we compare the accuracies of predicting the correct sentiment for both CSFD and IMDb datasets and both LSTM and CNN architectures. To compare with state of the art, we add the best results published so far. For the CSFD, we use results from the original dataset publication [3]. There is a newer publication available, however, the authors use external information (association of the reviews to particular movies) and they obtain only a minor improvement (+1.5%). For this dataset, we can reach the accuracy of the state-of-the-art models.

For the IMDb dataset, the best-published results [8] obtain astonishing accuracy of 95%. In this case, we reach significantly lower accuracies. However, we believe that with careful fine-tuning, we would be able to decrease the gap slightly. Nevertheless, we think that the obtained accuracy is sufficient enough for the end users.

TABLE I
Model Accuracies

Dataset	CSFD		IMDb	
Model	Original	Quantized	Original	Quantized
LSTM	80	79	86	86
CNN	77	76	83	81
SoTA	81	–	95	–

VI. Conclusion

Our experiments show that even an average mobile phone can process tens to hundreds of text documents per second. To put these numbers into a context, a desktop computer (6-core Intel CPU) with a GPU accelerator (Nvidia GTX 1080Ti) can process up to 50 times more requests per second than a powerful mobile device (such as Google Pixel 3XL). We estimate that with more complex models, the difference between a mobile device and a server may increase even more.

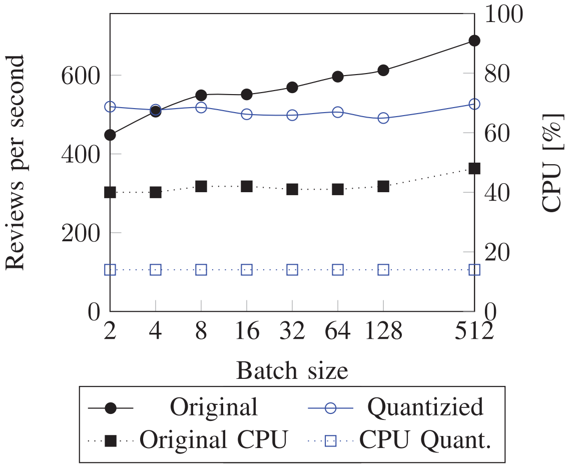

Fig. 3. CNN – CSFD – Huawei p9 lite

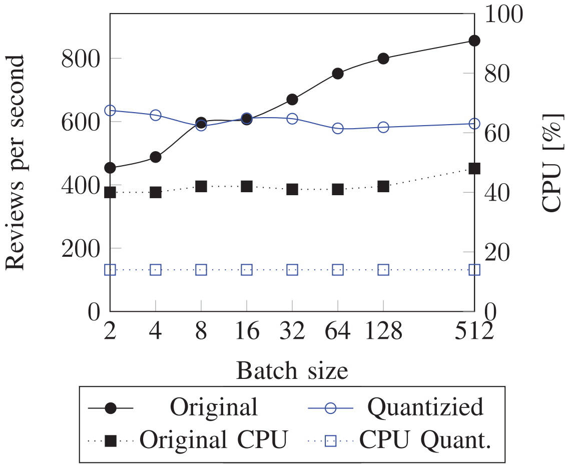

Fig. 4. CNN – IMDB – Huawei p9 lite

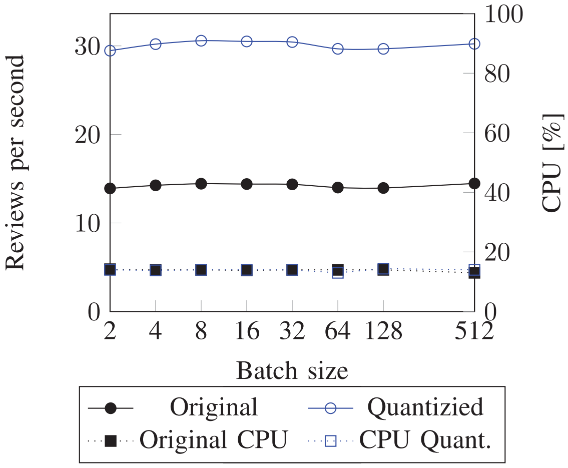

Fig. 5. LSTM – CSFD – Huawei p9 lite

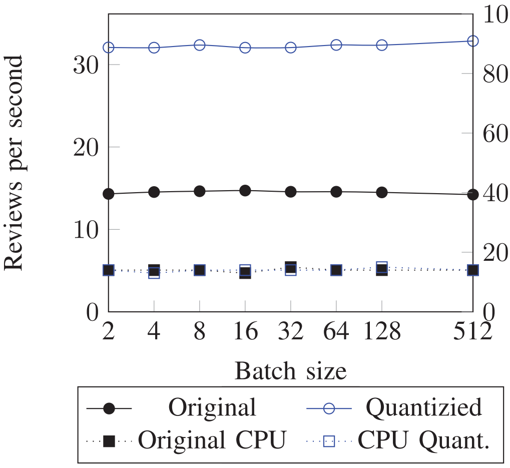

Fig. 6. LSTM – IMDb – Huawei p9 lite

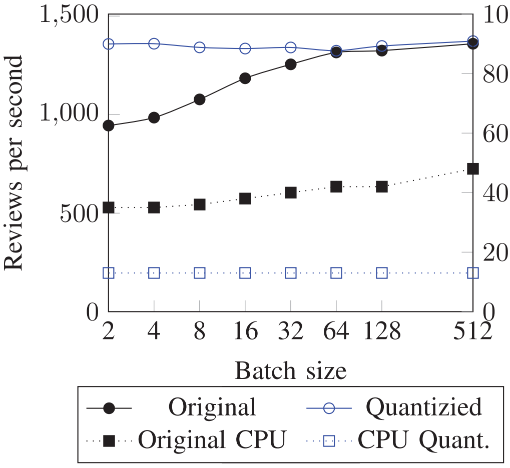

Fig. 7. CNN – CSFD – PIXEL 3XL

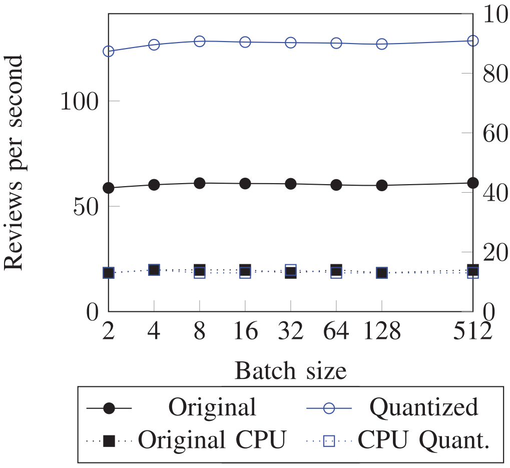

Fig. 8. LSTM – CSFD – PIXEL 3XL

Our results indicate that the state-of-the-art models deliver higher accuracy than our simplified ones. However, the accuracy of simplified models is still sufficient according to our opinion. Therefore, we conclude that it is technically possible to process user data directly on mobile devices. Users can enjoy the possibilities to predict a match of their interests, analyze post on their social network walls, extract information from their e-mails and more without the necessity to share their data with third parties.

VII. Acknowledgement

This work has been supported by Grant No. SGS-2019-018 Processing of heterogeneous data and its specialized applications, and was partly supported from ERDF “Research and Development of Intelligent Components of Advanced Technologies for the Pilsen Metropolitan Area (InteCom)” and by the project LO1506 of the Czech Ministry of Education, Youth and Sports. Computational resources were supplied by the Ministry of Education, Youth and Sports of the Czech Republic under the Projects CESNET (Project No. LM2015042) and CERIT-Scientific Cloud (Project No. LM2015085) provided within the program “Projects of Large Research, Development and Innovations Infrastructures”.

References

[1] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. AI benchmark: Running deep neural networks on android smartphones. *CoRR*, abs/1810.01109, 2018.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Ivan Habernal, Tomáš Ptácek, and Josef Steinberger. Sentiment analysis in czech social media using supervised machine learning. In *Proceedings of the 4th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 65–74, 2013.

[4] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 142–150, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. In *Advances in neural information processing systems*, pages 473–479, 1997.

[6] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[7] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Dec 1989.

[8] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia, July 2018. Association for Computational Linguistics.