Compression of Neural Networks for Usage in Mobile Devices and Embedded Systems

M. Faizullin, R.Kiryanov, A.Yudnikov, M.Blumental

December 23, 2018

1 Abstract

Nowadays implementation of neural networks for mobile device applications is bounded by hardware properties, especially, available memory. Even the most powerful portable devices has no more than 256 Gb of disk space, which must be distributed between lots of applications. At the same time, neural networks can weigh a lot (for example, VGG16 has weight of more than 530 MB), and most of this memory is usually allocated for storing *weight matrices of fully-connected (dense) layers*. Neural networks adaptation for portable devices can be done by different techniques. One of them is a low-rank approximation of weight matrices, which can be done by *singular value decomposition (SVD)*.

In present work we applied this technique for approximation of 3 dense layers of a custom *convolutional neural network (CNN)*, trained for binary classification of input images. We measured decrease in memory, required for storing the network, inference time for a single prediction and classification accuracy via ROC AUC score on an Android mobile phone and Raspberry Pi device.

We demonstrated that allocated memory for neural network storage could be significantly reduced from **2.7 MB** to **36.1 KB** ($\approx$ 85 times smaller), while the classification accuracy slightly dropped from **0,994** to **0,992** both on the Android device and Raspberry Pi.

2 Theory

Singular-value decomposition (SVD) is a factorization of a real or complex matrix with size $m \times n$ in a form of $U\Sigma V^*$. Here U is an $m \times m$ real or complex unitary matrix, Σ is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and V is an $n \times n$ real or complex unitary matrix.

The Eckart–Young theorem states that in the case that the approximation is based on minimizing the Frobenius norm of the difference between M and $\tilde{\mathbf{M}}$ under the constraint that rank $\left(\tilde{\mathbf{M}}\right) = r$ it turns out that the solution is given by the SVD of M, namely $\tilde{\mathbf{M}} = \mathbf{U}\tilde{\mathbf{\Sigma}}\mathbf{V}^*$. Here $\tilde{\mathbf{\Sigma}}$ is the same matrix as Σ except that it contains only the r largest singular values (the other singular values are replaced by zero). It means that low-rank approximation built by SVD is the best possible approximation for a given rank.

Nowadays, there are a lot of different methods to compute singular value decomposition. Some of them are constructed to use unique properties of matrices (for example, SVD of symmetric or bidiagonal matrices) and their types (dense or sparse) while others just perform SVD in an efficient way. At this moment one of the most popular solution to do this is the LAPACK library written in Fortran. It provides such functions as implicit QR algorithm, computing SVD via a Jacobi iteration and divide and conquer algorithm. The last one is presented in Numpy (high-level mathematical Python library) by *_gesdd* LAPACK subroutine. As it was mentioned above, it implements divide and conquer algorithm whose idea is to transform initial matrix A to bidiagonal form

$$A = \begin{pmatrix} U_1 & U_2 \end{pmatrix} \begin{pmatrix} B \\ 0 \end{pmatrix} V^\top$$

and then recursively divide B by blocks and compute SVD of each of them. For more details look at original paper[1].

In this work we had two ways to compute SVD: by Numpy or by Tensorflow. The last one is used to construct the required neural network and provides its own singular value decomposition algorithm. The aim of this is to use all the performance which can be achieved by using GPU. We decided to make this decomposition manually with Numpy for learning purposes and measured processing time on CPU for both ways. We generated random square matrices with sizes varied from 250 to 2000 and have measured time required to compute Numpy SVD in one case and Tensorflow SVD with transformation the result to numpy array in another. It should be mentioned that the last operation must be performed inside Tensorflow session [2]. The results of this test are presented in **Table 1**.

Table 1: SVD computation time

library / matrix size	250×250	1000×1000	2000×2000
Tensorflow	4.3 s	9.7 s	58 s
Numpy	330 ms	500 ms	600 ms

So, computation SVD using Numpy takes much less time than one using Tensorflow and therefore it was decided to use Numpy function.

3 Experiment

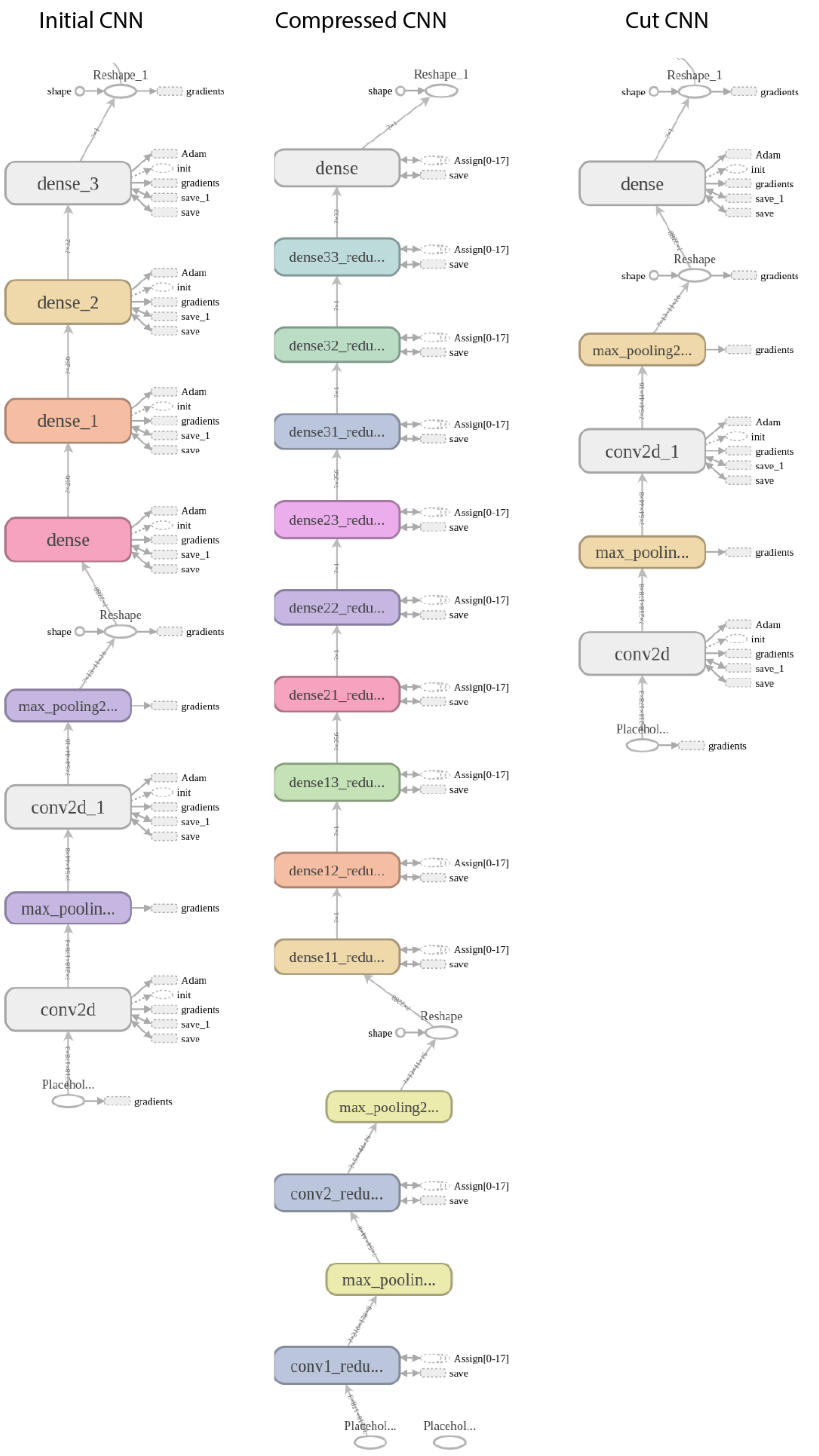
As a model experiment we chose a task of **gender classification** on CelebA dataset[3], containing $202,559$ face images with size 178×218 pixels.

Pic.1. Examples of face images in CelebA dataset.

For neural network construction we used **TensorFlow** package, because of its ease of converting to *tflite* version for mobile devies on Android operating system. We trained simple CNN with 2 convolutional and 3 subsequent fully-connected layers (pic.2). Next, we extracted weight matrices from *tf.Graph* structure of trained network and perform singular-value decomposition via module *numpy.linalg.svd*. Truncated matrices U_r^j, Σ_r^j and V_r^{*j} for each initial dense layer D_j were set as weight matrices to respectively $D_{j,3}, D_{j,2}$ and $D_{j,1}$ dense layers of Compressed CNN (pic.2). Also we constructed *cut* version of model with no dense layers between convolution and prediction layers (pic.2).

We converted TensorFlow networks to *tflite* versions for adaptation for mobile devises, based on Android operating system and uploaded networks on mobile phone and Raspberry Pi platform. We used custom software for testing total classification accuracy and mean inference time. All collected data and scripts are available on GitHub repository[4].

Raspberry Pi was powered by latest version of Raspbian (Version: November 2018) without any optimization as it is. Swap storage of 1GB was used on USB-flash card. Control of Raspberry Pi 3 Model B was from laptop via ssh connection. Pure TensorFlow was installed rather than TensorFlow Lite. The embedded system tested pretrained NN-models just like Android device. Some relation between quality of power supply and classification time was noticed. Using more powerful power supply decreased mean classification time almost two times.



Pic.2. Graph representation of initial (left), compressed (center) and cut(right) versions of custom CNN

4 Results

Result of tests are collected in **Table 2**. We managed to compress significantly weight matrices of fully-connected layers of custom CNN (from 2.7 MB to 36.1 KB), while the accuracy dropped really slightly (from 99,4% to 99,2%). In comparison to *cut* version of CNN without dense layers *compressed* CNN demonstrated higher results. Also we showed, that inference time did not decrease in both *compressed* and *cut* versions, what is also a noticeable fact.

Table 2: Test Results

Tested parameters / Network	Initial	Compressed	Cut
Memory: size of tflite	2.7 MB	36.1 KB	26.6 KB
Prediction (roc_auc score): Android	0.994	0.992	0.989
Prediction (roc_auc score): Raspberry Pi	0.994	0.992	0.989
Mean inference time, ms: Android	69 ± 9	73 ± 10	73 ± 11
Mean inference time, ms: Raspberry Pi	73 ± 14	77 ± 26	73 ± 23

You can find all trained models and results on Google Drive [5].

5 Discussion

The experiments suggest that SVD allows to considerably reduce the size of neural networks at the cost of little prediction quality drop. Inference time does not improve, because the most time is spent on convolution of the input image. This explanation is consistent with the time measured for cut model.

To make the results even more prominent, one could choose a problem, where the role of dense layers would be bigger. A good candidate here is multiclass classification. In this case, the difference in prediction quality between compressed and cut versions would be larger. Also, for MNIST classification one could use a network with only dense layers, in which case inference time for compressed model could be better, than for initial.

We see several directions in which the research of neural networks compression could be continued.

First, there are other matrix decompositions which could be applied to this problem. Tensor train seems to be one of the most interesting variants here. Another approach could be this: turn a weight matrix into a sparse low-rank form with variational dropout and use skeleton decomposition.

Second, similar works [6] and [7] suggest to fine-tune the model restructured with SVD in order to eliminate the accuracy drop, which is another good idea to try.

6 Member Contribution

Maxim Blumental
- TensorFlow model adaptation for Android device
- Programming custom test software for Android device
- GitHub Repository

Marsel Faizullin
- TensorFlow model adaptation for Raspberri Pi device
- Programming custom test software for Raspberri Pi device
- Report

Alexander Yudnikov
- SVD implementation for network graph rebuild
- SVD comparison tests
- Report

Roman Kiryanov
- TensorFlow model compression, network graph rebuild
- Presentation and report

References

[1] Efficient Computation of the Singular Value Decomposition with Applications to Least Squares Problems; Ming Gu, James Demmely, Inderjit Dhillonz; 1994; www.netlib.org

[2] TensorFlow Documentation about session concept; www.tensorflow.org

[3] Large-scale CelebFaces Attributes (CelebA) Dataset; www.mmlab.ie.cuhk.edu

[4] Project GitHub Repository; www.github.com/maxblumental/network-compression

[5] Trained models and experiment results; https://drive.google.com/

[6] Restructuring of Deep Neural Network Acoustic Models with Singular Value Decomposition; Jian Xue, Jinyu Li, and Yifan Gong; 2013; www.microsoft.com

[7] Fast Learning of Deep Neural Networksvia Singular Value Decomposition; Cai1, Dengfeng Ke, YanyanXu, and Kaile Su; 2014; www.researchgate.net