NLA Project 2018

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

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

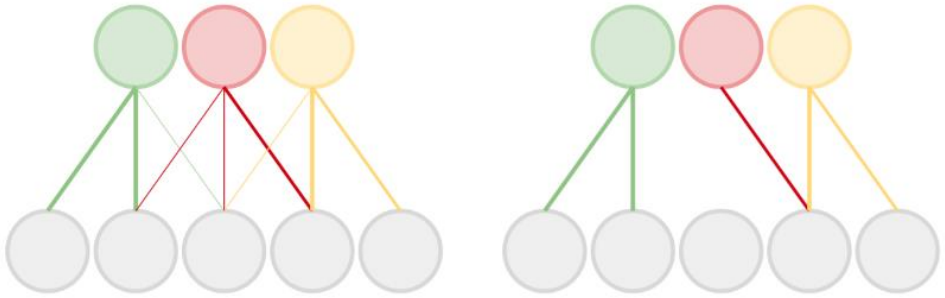
# Implementation of neural networks on mobile devices requires memory reducing techniques

## Current devices are bounded by:

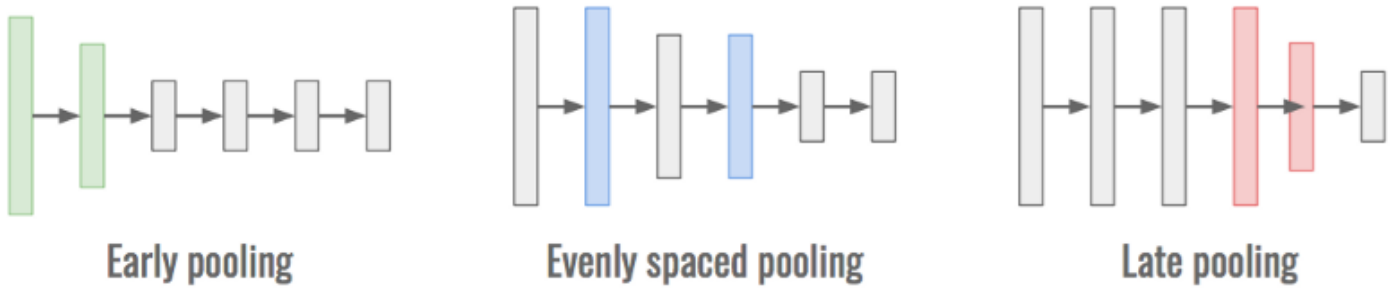
- Much less available disc space (256 Gb maximum)
- Computing power (currently only CPUs)

## Some tricks for neural networks adaptation:

### Weights pruning




### Optimizing the downsampling



# SVD decomposition can be used for low-rank approximation of weight matrices of fully-connected layers

Problem:

Storing weight matrices of dense layers requires a lot of space

- At the same time, dense layers perform simple matrix-by-vector multiplication

Idea:

Replace one fully connected layer with 3 consequent layers

- Memory decrease from $(N \times M) \to (N \times K) + (K \times K) + (K \times M)$ for one layer

Experiment:

Will decomposition influence:

- Prediction accuracy of model?
- Inference time (from $O(n^2)$ to $O(nr)$ matvec operations)?

# Methodology: NumPy SVD and TensorFlow network graph

## Implementation of neural networks in Python: TensorFlow

- Flexibility and control
- Ease of model converting to *tflite* version for mobile devices

## SVD implementations: NumPy

- NumPy SVD, using ‘divide and conquer’ algorithms (LAPACK subroutine)
- TensorFlow SVD, based on GPU (slower then NumPy on CPU)

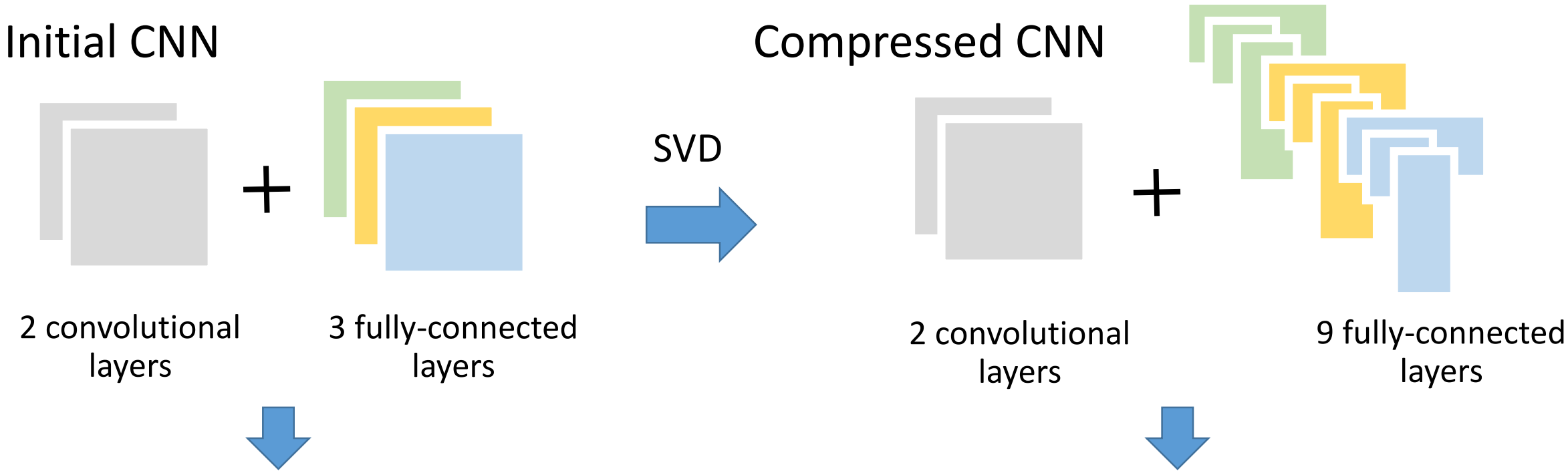
Comparison of SVD implementation runtime:

| SVD \ Matrix size | 250 x 250 | 1000 x 1000 | 2000 x 2000 |
|---|---|---|---|
| TensorFlow SVD | 4.3 s | 9.7 s | 58 s |
| NumPy SVD | 330 ms | 500 ms | 600 ms |

# Methodology: Experiment

## Neural network compression:

- Extraction of weight matrices (kernels) from trained network tf.Graph structure
- Create new network and substitute decomposed weight matrices




Convert to *tflite* model and tests for memory occupation, inference time and prediction accuracy on Android and Raspberry Pi devices

# Estimation of network compression

Our network is really dense

| Layer type | Number of weights |
|---|---|
| Convolution | 3824 |
| Dense | 660000 |

It can be significantly compressed!

| Approximation | Number of weights |
|---|---|
| None | 660000 |
| 5-rank | 41909 |

Formula for counting a convolution layer's weights:

$$N_w = n_{\text{in filters}} n^2_{\text{kernel size}} n_{\text{out filters}} + n_{\text{out filters}}$$

In our case:

$$N_{conv_1} = 3 * 5^2 * 8 + 8 = 608$$
$$N_{conv_2} = 8 * 5^2 * 16 + 16 = 3216$$
$$N_{conv} = N_{conv_1} + N_{conv_2} = 3824$$

For dense layers (excluding the last one):

$$N_w = n_{in} n_{out} + n_{out}$$
$$N_{dense_1} = 2288 * 256 + 256 = 585984$$
$$N_{dense_2} = 256 * 256 + 256 = 65792$$
$$N_{dense_3} = 256 * 32 + 32 = 8224$$
$$N_{dense} = N_{dense_1} + N_{dense_2} + N_{dense_3} = 660000$$

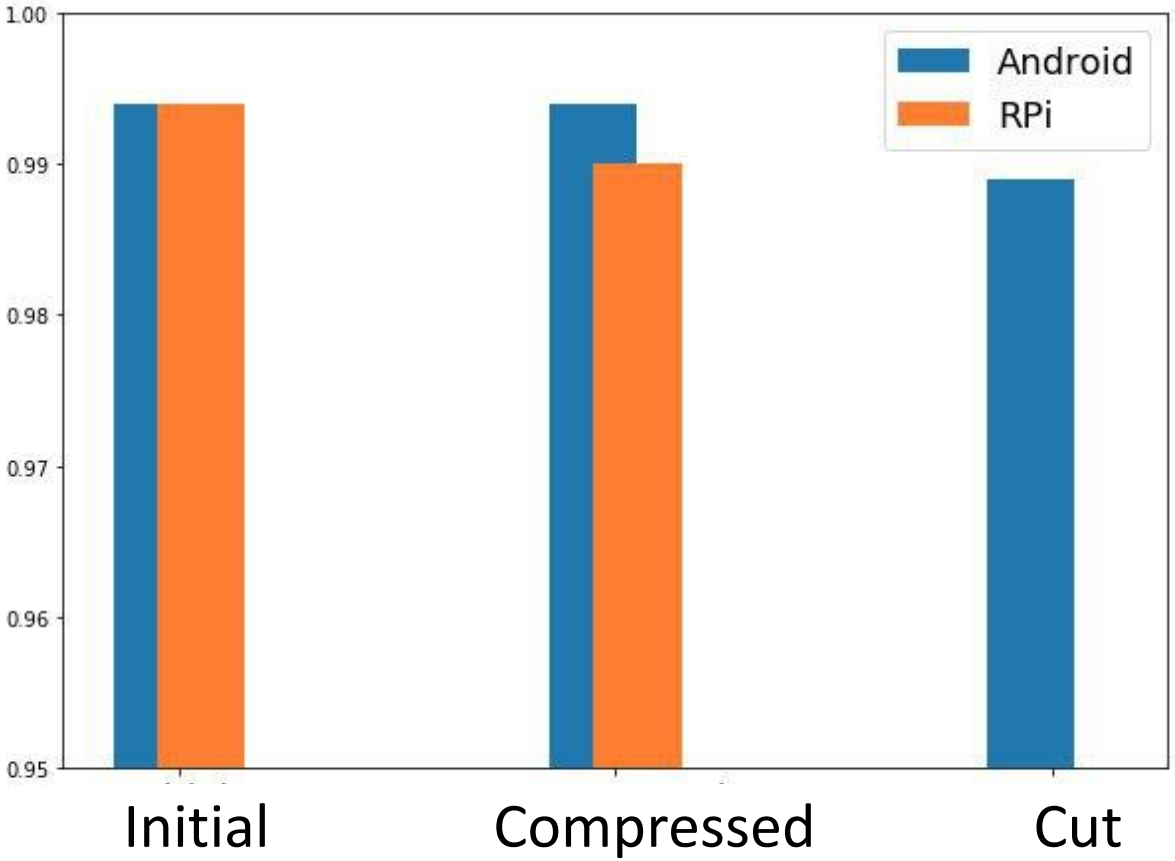
For dense layers after 5-rank approximation:

$$N_w = n_{in} n_{out} + n_{out}$$
$$N_{dense_1} = 2288 * 5 + 5 * 5 + 5 * 256 + 256 = 13001$$
$$N_{dense_2} = 256 * 5 + 5 * 5 + 5 * 256 + 256 = 2841$$
$$N_{dense_3} = 256 * 5 + 5 * 5 + 5 * 32 + 32 = 1497$$
$$N_{dense} = N_{dense_1} + N_{dense_2} + N_{dense_3} = 41909$$

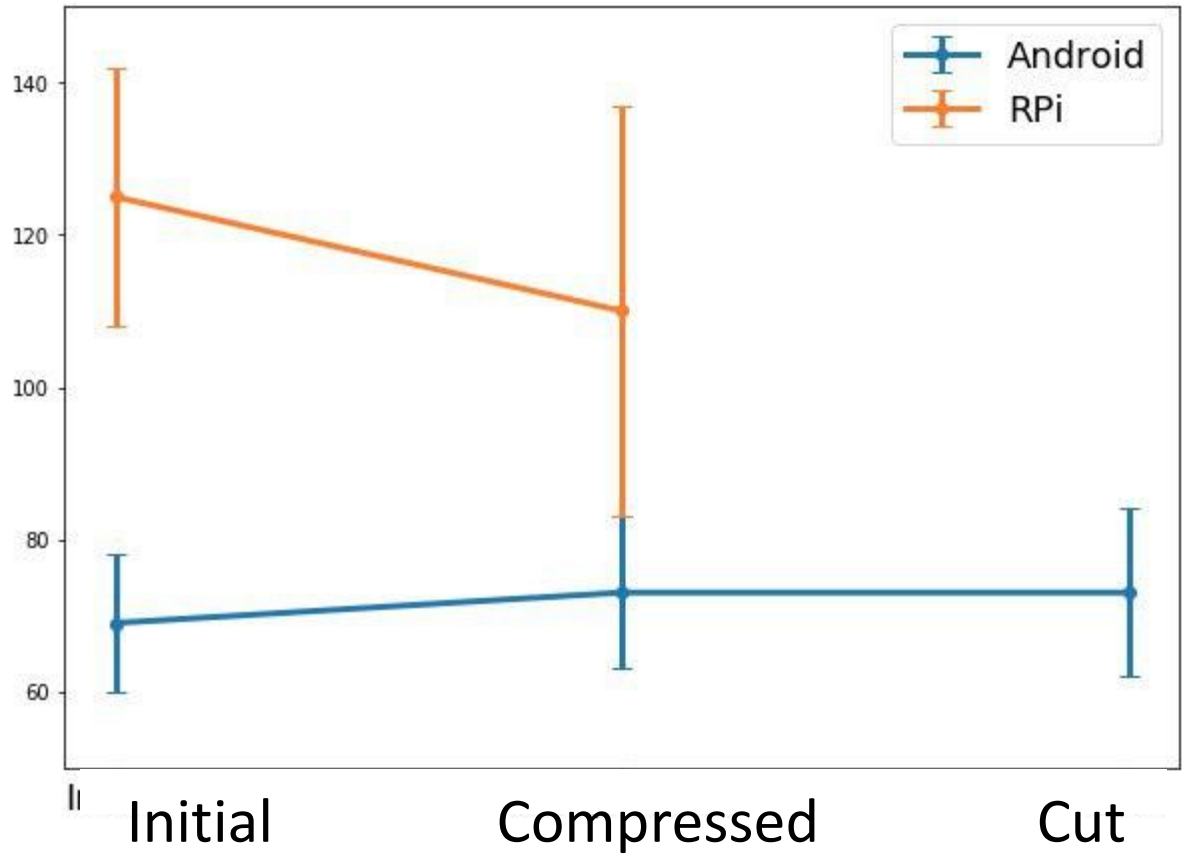
# Tests and Results: gender classification task

Model experiment: Gender classification task on CelebA dataset (202,059 face images)

Prediction score (roc_auc)




Mean inference time, ms



# Tests and Results: gender classification task

## Observed results:

| | Initial CNN | Compressed CNN | CNN without dense layers |
|---|---|---|---|
| Memory: size of tflite, | 2.7 MB | 36.1 KB | 26.6 KB |
| Inference time: Android, ms | 69 ± 9 | 73 ± 10 | 73 ± 11 |
| Inference time: Raspberry Pi, ms | 125 ± 17 | 110 ± 27 | - |
| Prediction (roc_auc score): Android | 0.994 | 0.992 | 0.989 |
| Prediction (roc_auc score): Raspberry Pi | 0.994 | 0.990 | - |

- Fully-connected layers for this task can be significantly compressed
- Still they are required for high-precision classification
- The effect of SVD decomposition could be more visible on more complicated tasks

# Team work: member contribution

## Maxim Blumental

- TensorFlow model adaptation for Android device
- Programming custom test software for Android device

## Marsel Faizullin

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

## Alexander Yudnikov

- SVD implementation for network graph rebuild
- SVD comparison tests

## Roman Riryanov

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