Human Recognition by Biometric Methods
Face classification

Mikołaj Małkiński

10 April 2018

1 Description of the assignment

The goal of this task was to design, implement and train a Convolutional Neural Network model for recognizing faces of (four) Polish rappers. Several different architectures were analysed, where the difference was mainly in the choice of base models: VGG16, ResNet50 and InceptionV3.

The original dataset was split into two parts: train (1600 images of each class) and validation (400 images of each class). The models are trained only on the train dataset and their accuracies are evaluated on the validation dataset.

The models are resizing the images before processing, so data of arbitrary size can be used. This procedure is done using ImageDataGenerator - a tool provided by Keras, which feeds data from directory into the model. Furthermore, it offers several data augmentation functionalities, which will be discussed later.

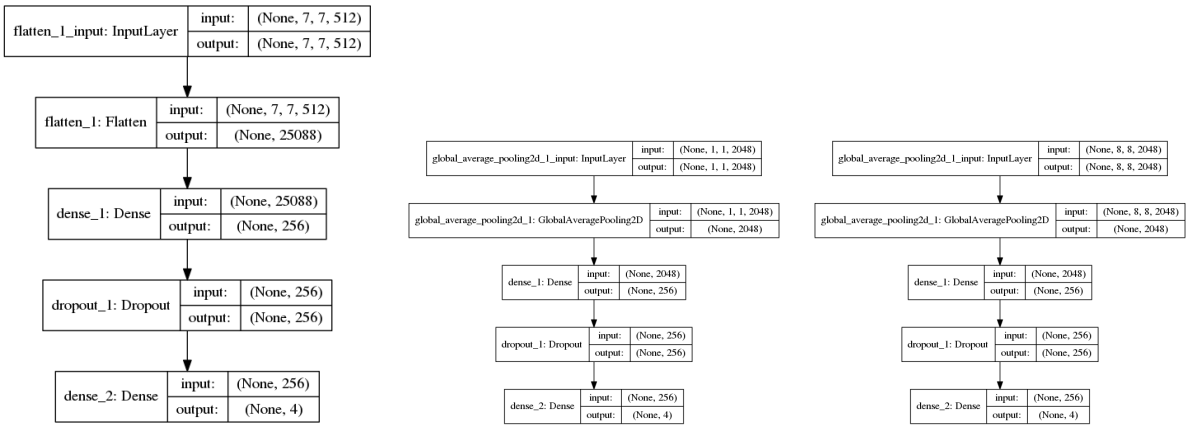
The project conists of following scripts:

- *evaluate* - evaluates model accuracy and loss on specified dataset,
- *predict* - predicts a class of specified image,
- *reproduce_results* - creates plots shown in the report,
- *train* - trains given model on specified dataset.

2 Overview of used models

Network architectures were created using already existing solutions, with modified top layers. Figure 1 presents which layers were added on top of base-line architectures, which were initialized with weights trained on the ImageNet dataset. During training, mostly the top layers were trained, with exception of one solution which used fine-tuning, which will be discussed later.

There was a noticeable difference between mentioned architectures. Figure 2 presents the training process. Networks created with the use of VGG16 and InceptionV3 gave satisfying results, which are presented in the Figure 3. However, ResNet50 seems to be way behind the competition.



(a) VGG16 (b) ResNet50 (c) InceptionV3

Figure 1: Custom top models

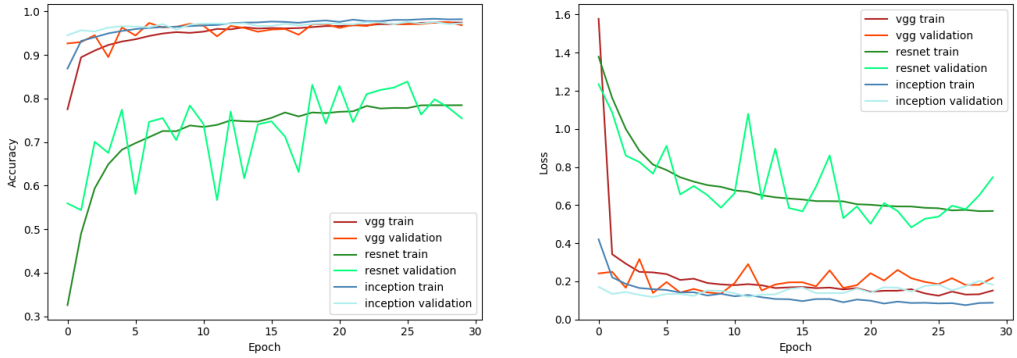


Figure 2: Performance of VGG16, ResNet50 and InceptionV3 with custom top models

	VGG16	ResNet50	InceptionV3
accuracy	0.9681	0.8231	0.9506
loss	0.2902	0.5178	0.7757

Figure 3: Results obtained on validation set

3 Data augmentation

One possibility to improve the accuracy of network, is to use data augmentation. Keras provides a utility called ImageDataGenerator, which is quite convenient for this task. It's options are well described in the documentation. Figure 4 presents the effect of applying data augmentation for VGG16 architecture. Unfortunately, it didn't give satisfactory results, probably because the dataset seems to be already influenced by some kind of data augmentation. The numerical results are shown in the Figure 5.

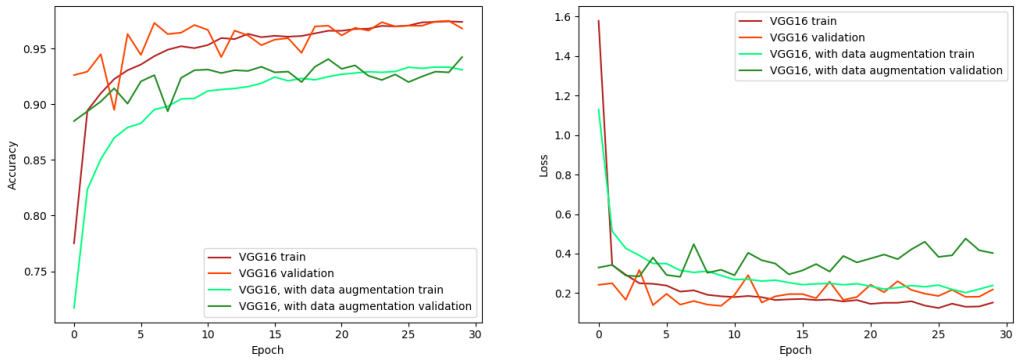


Figure 4: Influence of data augmentation on VGG16

	Original dataset	With data augmentation
accuracy	0.9681	0.9425
loss	0.2902	0.4028

Figure 5: Results of VGG16

4 Fine tuning

After training the top layers, it is often desirable to freeze them, and proceed with training several layers from the base architecture. This technique is called fine-tuning and can yield noticeable improvements. In this case, using VGG16 as base model, it gave improvement of more than 1%, which is a good result considering the fact, that plain base model had already accuracy of almost 97%. The training process is presented in the Figure 6. Observer that the fine-tuned version has big accuracy starting from the first epochs, because the top layers were already trained. Numerical results computed on validation and training sets are shown in Figures 7 and 8.

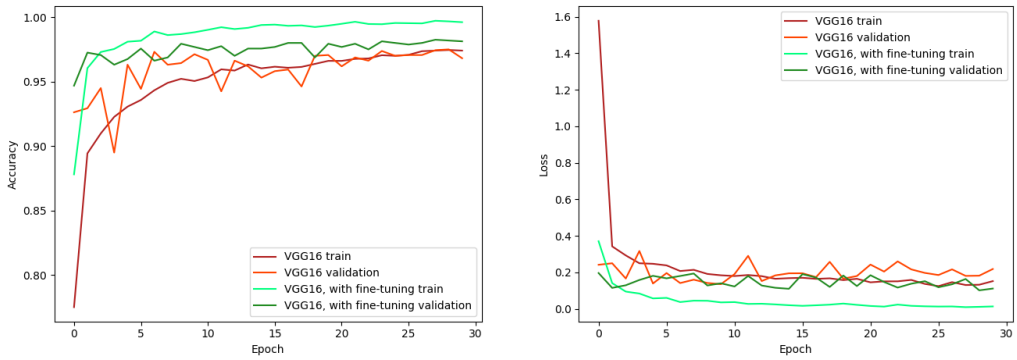


Figure 6: Influence of fine-tuning on VGG16

	Only top model trained	Fine-tuning
accuracy	0.9681	0.98125
loss	0.2902	0.11157

Figure 7: Results of VGG16 on validation set

	Only top model trained	Fine-tuning
accuracy	0.9858	0.9992
loss	0.0926	0.0049

Figure 8: Results of VGG16 on training set

5 Hyperparameters

5.1 Batch size

Batch size is one of the hyperparameters which can influence the network accuracy. During experiments, two batch sizes were analysed: 16 and 32. Unfortunately, starting from 64, the batch of images wasn't able to bewas too big to fit it into GPU memory. In the Figure 9 the influence of mentioned batch sizes are shown. It can be noticed, that larger batch size gave slightly better results, but the difference isn't huge.

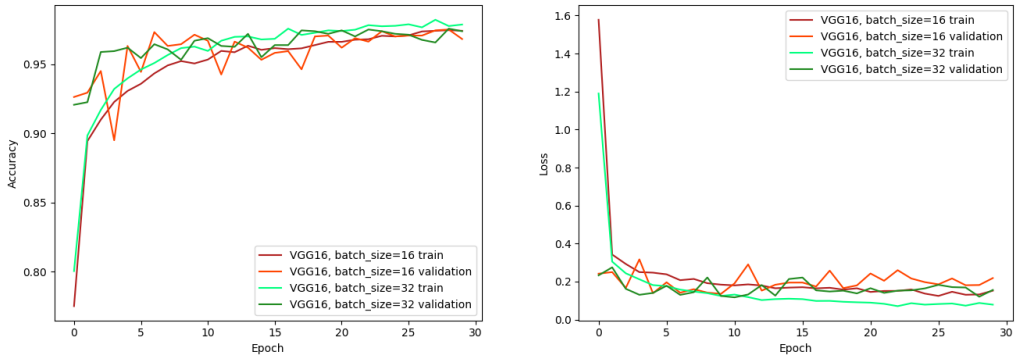


Figure 9: Performance of VGG16 using different batch size

5.2 Optimizers

Optimizers are another kind of hyperparameters which can influence the time needed to train the network as well as it's accuracy. Figure 10 presents results obtained using default RMSprop and Adam optimizers. The results are rather similar. However, the choice of learning rate can be crucial for reasonable convergence time and accuracy. In the Figure 11 a deteriorating impact of RMSprop optimizer with too big learning rate is presented. On the other hand, Figure 12 highlights how slow a network converges, when the learning rate is too low. During experiments, it was found out that default optimizer values bundled into Keras give satisfactory results.

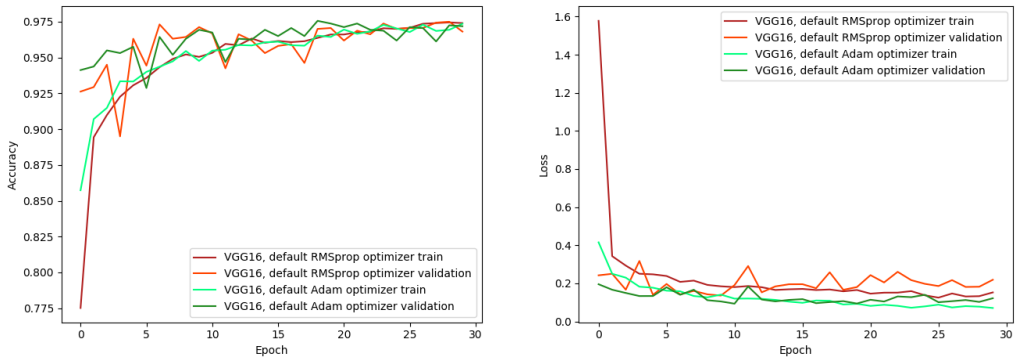


Figure 10: Default RMSprop and Adam optimizers for VGG16

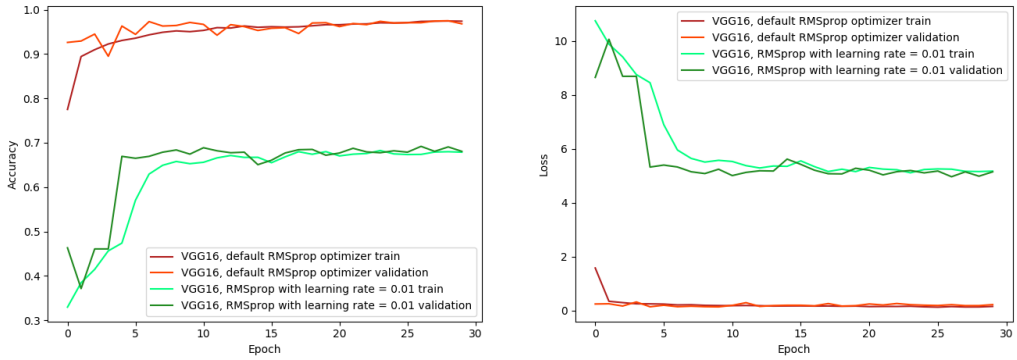


Figure 11: Influence of RMSprop optimizer with learning rate = 0.01 on VGG16

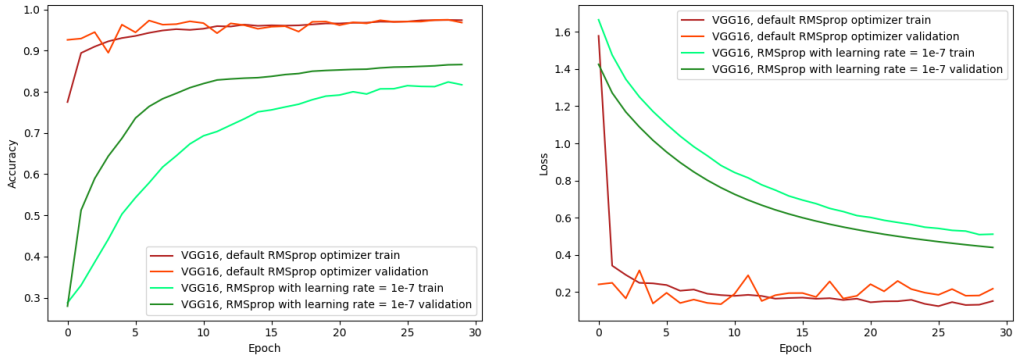


Figure 12: Influence of RMSprop optimizer with learning rate = 1e-7 on VGG16

5.3 Dropout

To prevent a strong network from overfitting, dropout can be applied. The Figure 13 shows the effect of including dropout layer, with value = 20%, into top model put on top of ResNet50 architecture. In this case, including dropout layer only reduced the accuracy. Finally in the Figure 14, mentioned architecture with dropout is analysed on larger amount

of epochs.

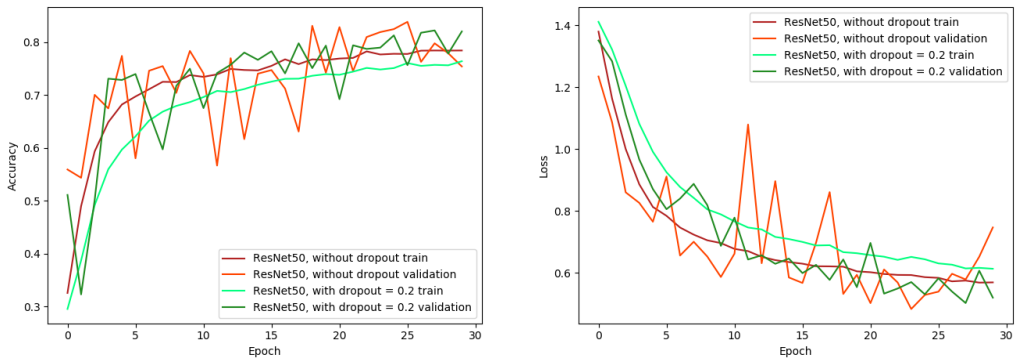


Figure 13: Performance of ResNet50 with dropout in top model

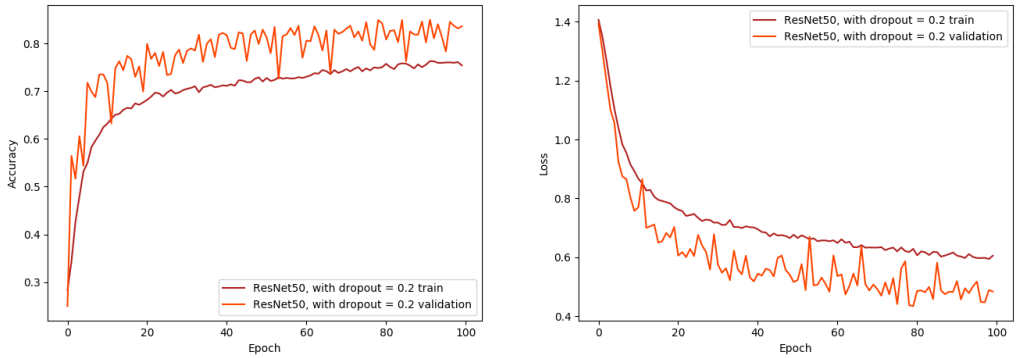


Figure 14: Performance of ResNet50 with dropout in top model in larger amount of epochs

6 Visualization of first convolutional filters

In the figure 15, first 6 filters from first convolutional block of VGG16 architecture are presented. They extract certain features from images. The first layers are responsible for rather simple features. However, the deeper the layer, the more abstract features it can detect. This is why the base architectures, like VGG16, ResNet50 or InceptionV3 without top layers, already trained on given dataset, are sometimes referred to as 'feature extractors'.

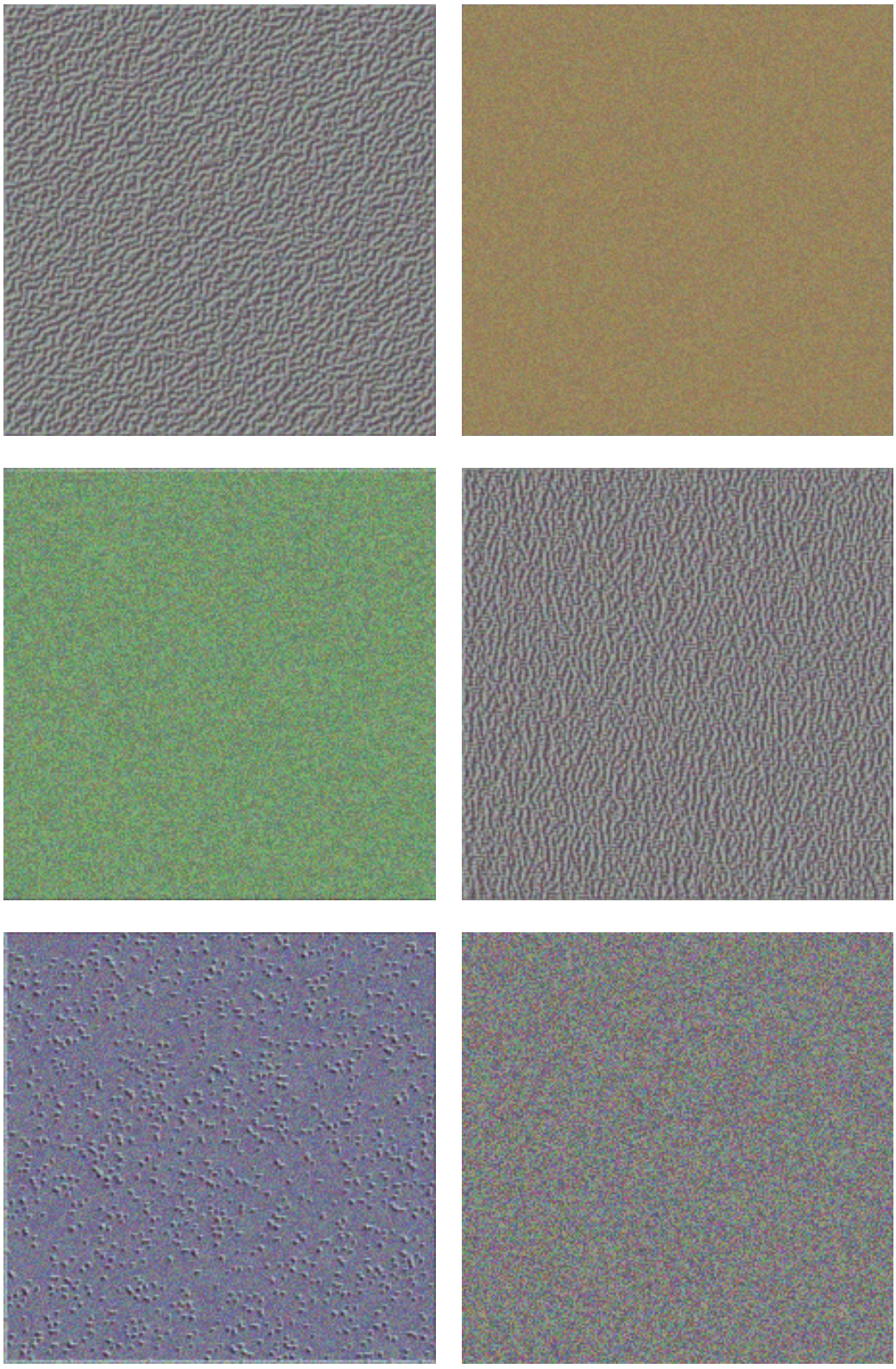


Figure 15: First convolutional filters

7 Examples of good/bad predictions