DEEP LEARNING 2022

IMAGE CLASSIFICATION WITH CONVOLUTIONAL NEURAL NETWORKS
DEEP LEARNING PROJECT NO. 1 REPORT

April 12, 2022

Elżbieta Jowik (298821), Agata Makarewicz (298827)

1 Introduction

The objective of this document is to describe the methodology and summarise the results of the applied approach to the problem of CIFAR images classification with convolutional neural networks. The report consists of two main parts. The first one is focused on giving a theoretical background of the project and adopted assumptions most of which concern the experiments' setup. The other one concerns the practical application of the networks under consideration and presents the performance of both custom and widely-used pre-trained models. While the full code is included in the attached files, this paper points out only the most important facts like an overview of the design, performance of different models and interpretations of the results.

2 Data

A dataset used for this project, which is `CIFAR-10`, is an established computer-vision dataset for object recognition. The dataset contains images with a (32×32) resolution. All of them are of shape $(32, 32, 3)$ where 3 represents the number of channels i.e R-G-B (Red, Green & Blue). The task realized in this project belongs to the class of multi-classification problems and the `10` in the set name refers to the cardinality of the set of unique values of the target variable As for data preprocessing, we found it inevitable to perform target variable categorical encoding. The desired dummy form of the target variable was obtained by applying the `One-Hot Encoding` method. In the raw data, all the image pixels are in a range $[0, 255]$, therefore in order to accelerate the model training process, we scaled all the pixel values to the range of $[0, 1]$ which is a standard procedure in images classification. The desired result was obtained by division of all the pixel values by 255. In the experiments, we investigated both the robustness of the networks themselves and the power of ensembling. For this purpose, we performed data augmentation, which is a technique used to alter existing images in order to create new training data. We applied common data augmentation methods such as scaling, rotating and adding some noise to the picture.

3 Custom models

3.1 Architectures

In this project, we proposed and compared three different architectures of the neural network model. The idea behind all adopted approaches was to let the network learn of the simplest objects like lines or edges in the initial phase and then more complex ones which require more filters. All convolutional layers use 3×3 kernels, but they do differ in the number of filters. Due to the fact that the `ReLu` activation function is not associated with the vanishing gradient problem, it was applied in each layer of the network. To reduce the risk of an overfitting problem occurrence we added dropouts whereas to reduce the number of epochs (relevant for estimating the most accurate parameters) batch normalization and early stopping were applied.

Specific values of hyperparameters such as `learning rate` or `batch size = 32` were adjusted based on the literature or customized empirically throughout conducted experiments. The learning process required up to 60 epochs assuming that an early stopping was introduced. The fitting should be stopped if a monitored metric (`val_accuracy`) has not improved by 0.01 (`min_delta`) for 10 epochs (`patience`). As an optimizer, the ADAM method was used which is currently perceived as the most powerful one. The learning rate was adjusted gradually with the iterations (minimized according to the `decay = 1e-6` parameter).

We distinguished 3 versions of each of the models presented below, depending on the version of the training set on which they were trained - a basic version and 2 modifications (data augmentations).

Model 1

- Conv2D
- LeakyReLU
- Conv2D
- LeakyReLU
- MaxPooling2D
- Dropout
- Conv2D
- LeakyReLU
- Conv2D
- LeakyReLU
- MaxPooling2D
- Dropout
- Flatten
- Dense
- LeakyReLU
- Dropout
- Dense
- Activation

Model 2

- Conv2D
- Conv2D
- MaxPooling2D
- Dropout
- Conv2D
- Conv2D
- MaxPooling2D
- Dropout
- Conv2D
- Conv2D
- MaxPooling2D
- Dropout
- BatchNormalization
- Flatten
- Dense
- Dropout
- Dense

Model 3

- Conv2D
- MaxPooling2D
- Conv2D
- MaxPooling2D
- Flatten
- Dense
- Dropout
- Dense
- Dropout
- Dense
- Dropout
- Dense

3.2 Augmentation

As mentioned in the previous section, for each of the proposed models (architectures) we performed data augmentation - a technique used to alter existing instances (images) in order to create new training data. We compared two different approaches, consisting of modifications listed below.

Approach 1

- width shift (`range = 0.2`)
- height shift (`range = 0.2`)
- zoom (`range = 0.2`)
- rotation (`range = 20`)
- whitening (`zca epsilon = 1e-6`)

Approach 2

- width shift (`range = 0.1`)
- height shift (`range = 0.1`)
- zoom (`range = 0.1`)
- horizontal flip
- vertical flip

4 Pretrained models

One of the core objectives of this project was to utilize available pre-trained models. For this purpose, we tuned ResNet and VGG16 models pre-trained on ImageNet which is a dataset of

over 14 million images belonging to 1000 classes. For a high-level overview of the models, see the figure below. However, due to the fact that the classification with the use of pre-trained networks was of a benchmark nature, the implementation details of the existing architectures are beyond the scope of this document.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 1, 1, 2048)        23587712

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 10)                20490

=================================================================
Total params: 23,608,202
Trainable params: 23,555,082
Non-trainable params: 53,120
_________________________________________________________________
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 1, 1, 512)         14714688

 flatten_1 (Flatten)         (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 14,719,818
Trainable params: 14,719,818
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Pretrained models details

The parameterization of the learning process as to the value of the learning rate was consistent with the parameterization of custom architectures. The learning process was carried out in two variants - with and without the possibility of triggering early stopping.

5 Experiments & evaluation

Both models with custom architectures and pre-trained models were evaluated on classification accuracy and the heatmaps of the confusion matrices. However, the setup of the experiments was different for each type of model and will be described in detail in the following sections.

5.1 Custom models

Firstly, we ran an experiment that let us identify the best value of the `learning rate` parameter for each of the models. Five potential values of the parameter were considered: [1e-2, 1e-3, 1e-4, 1e-5, 1e-6]. The results were evaluated on both convergence of the learning process and accuracy of the final prediction. Due to the hardware limitations, we considered only basic versions of the models and they were fitted only once, simply to get a high-level overview of their behaviour.

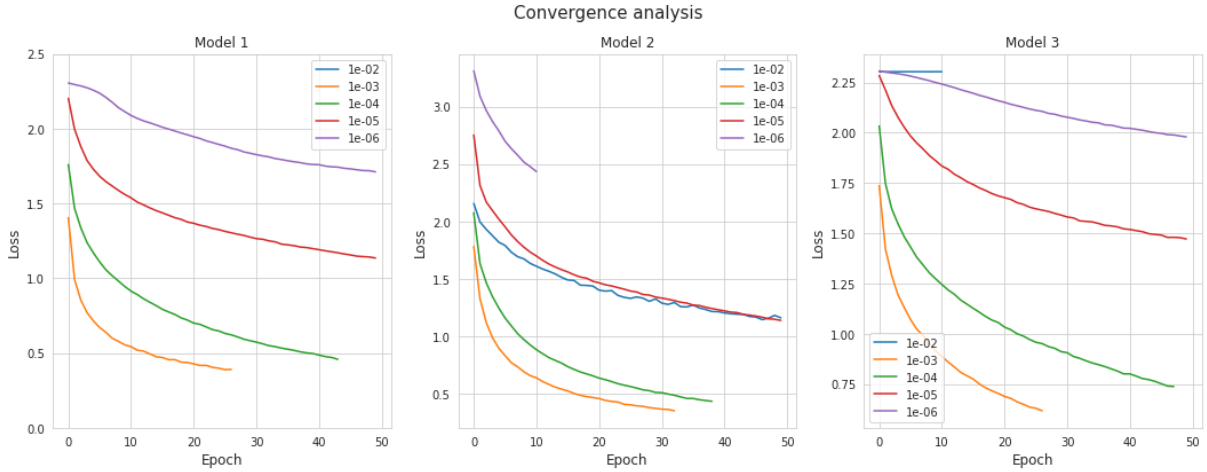

Figure 2: Learning curves of each model (basic version) measured on the train set, for different values of the learning rate parameter. The curves differ in length due to the usage of early stopping. The first plot does not present the learning curve for `learning rate = 1e-2`, because the loss was too high to show it next to the rest of the results.

We can observe that the bigger the learning rate, the faster the learning process. The speed of the fitting is highly correlated to the length of the process, due to the fact that we use early stopping to cease the learning before the model starts to overfit. This situation can be observed in the plots in the Figure 2, for all of the models concerned. The best results are obtained for the parameter equal to $1e-3$ and $1e-4$

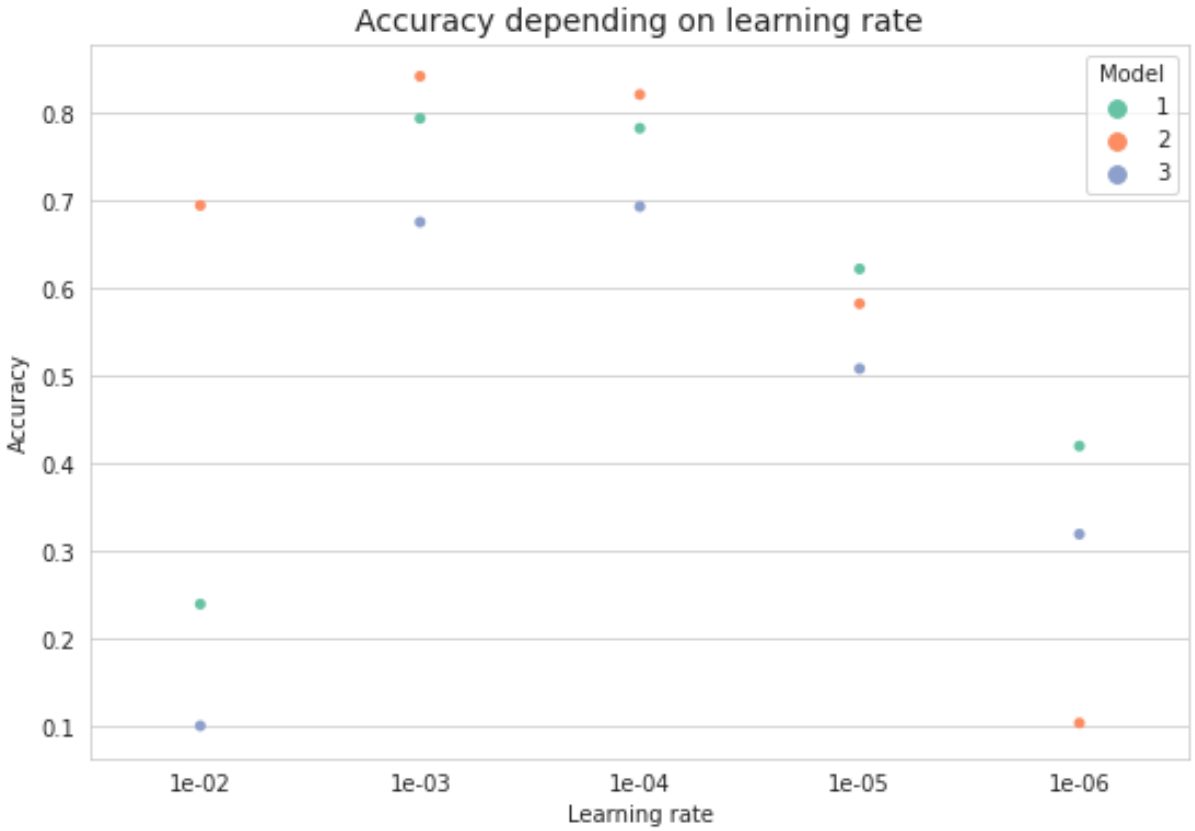

Figure 3: Accuracy of each model (basic version) measured on the test set, for different values of the learning rate parameter.

The scatterplot of accuracy values shown in Figure 3 confirms the conclusions from the line plots in Figure 2. Two of the models (1 & 2) achieve their best results for the `learning rate = 1e-3` whereas the 3rd model performs slightly better for the `learning rate = 1e-4`. However, these plots are based on only one fitting of each model and such small differences might become unrelatable in the long run. Therefore, further experiments were conducted, to investigate the real model behaviour, regardless of, for instance, the random initialization of the weights. For this purpose, we fit each of the models, in all 3 versions, 5 times, and evaluate it on the test data set. Based on that evaluations we measure simple statistics of the model's results - minimum, maximum, mean and standard deviation value of the accuracy metrics. We also investigate the mean number of epochs it took for the model to learn because due to the usage of early stopping, the number of iterations might differ in each run of the model.

The experiments were performed with the adopted set of default hyperparameters values listed below. The `learning rate` was selected based on the previously mentioned experiment, in which all the other parameters were set to default values. Two of the parameters - `patience` and `min delta` relate to the early stopping configuration. The fitting should be stopped if a monitored metric i.e. accuracy has not improved by 0.01 for 10 epochs on the test set.

- `batch size = 32`
- `epochs = 100`
- `learning rate = 1e-3`
- `patience = 10`
- `min delta = 0.01`
- `loss = binary crossentropy`

As we can see in the Table 1, it takes more time for the model to learn (and overfit) while performing data augmentation on the training data set. Depending on the model, it takes from 5 to 20 epochs more, on average. Generally, the second model performs best, obtaining the highest accuracy equal to 0.85. It is also characterized by one of the lowest standard deviations of the results. Model 3 compares badly with the other two, never reaching the level of 0.7 accuracy. However, it is quite understandable given the fact that its architecture is way more simplified than the other two. Additionally, for model 2 we can observe the smallest differences in performance between the basic version and the modified ones (with augmentation included).

Table 1: Statistics of the custom models in terms of accuracy. The models were run 5 times and the table presents the mean value, standard deviation, minimum value and maximum value of the obtained accuracy, as well as the mean number of epochs in which the result was obtained.

model	mean acc	std dev acc	min acc	max acc	mean n of epochs
model 1	0.799	0.003	0.795	0.804	25.8
model 1 aug 1	0.729	0.015	0.715	0.751	32.6
model 1 aug 2	0.708	0.018	0.690	0.733	30.2
model 2	0.843	0.007	0.835	0.852	33.6
model 2 aug 1	0.798	0.019	0.764	0.813	43.8
model 2 aug 2	0.781	0.020	0.765	0.812	52.0
model 3	0.679	0.007	0.666	0.686	24.2
model 3 aug 1	0.653	0.010	0.634	0.686	45.2
model 3 aug 2	0.629	0.014	0.616	0.645	43.4

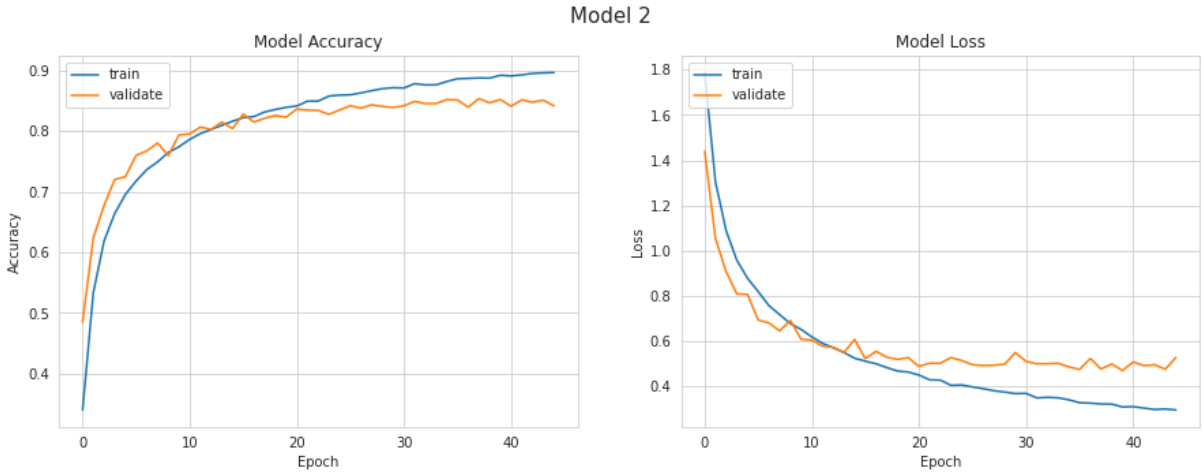

Figure 4: Learning curves (loss and accuracy) of the model 2 exemplary fitting (basic version).

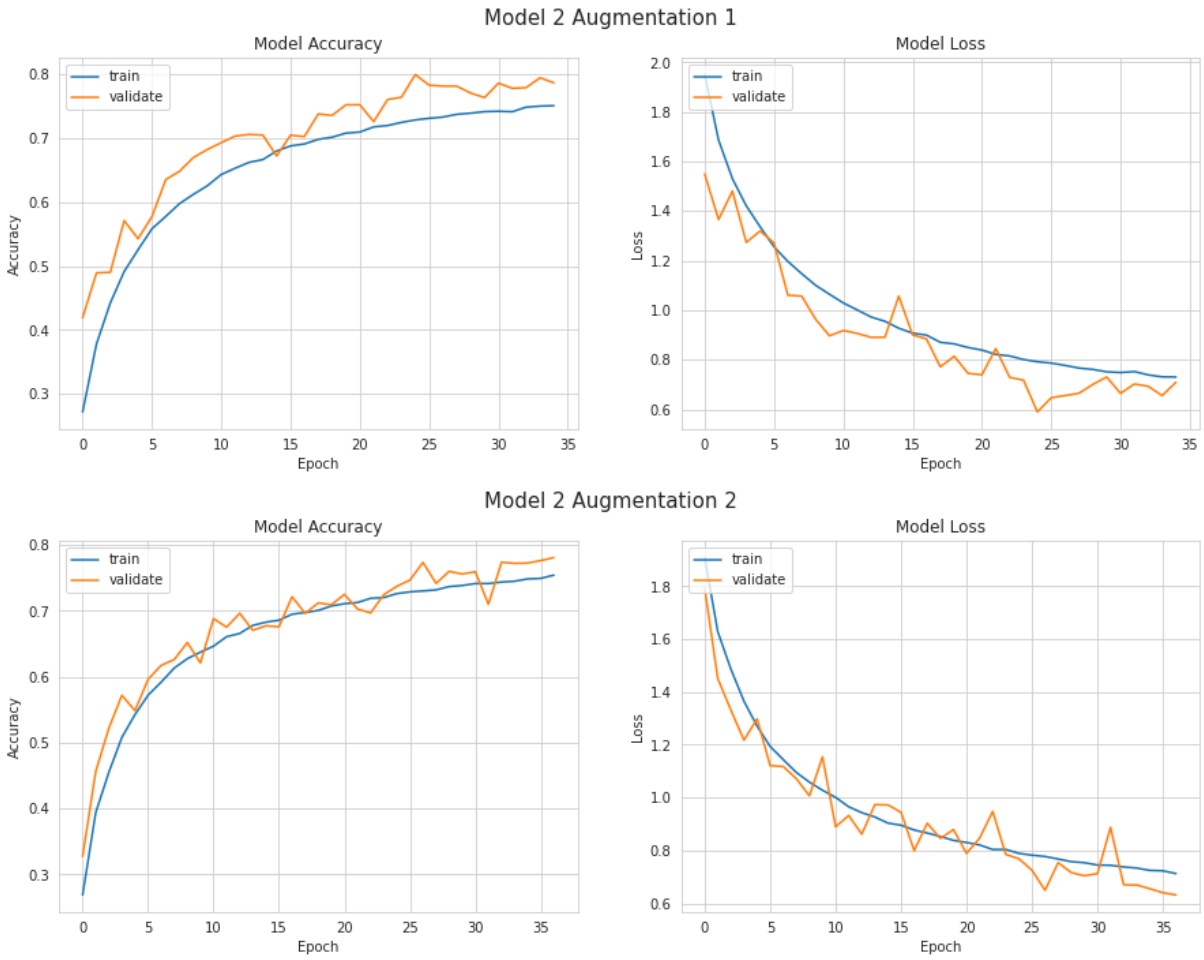

Figure 5: Learning curves (loss and accuracy) of the model 2 exemplary fittings (after augmentations).

The results of the experiments shown in the Table 1 confirm the conclusions which can be drawn from the plots of the learning curves in Figures 4 and 5. Basic versions of the models are characterized by the lower standard deviation of the accuracy values however they learn (and overfit) faster.

Lastly, we analyzed the confusion matrices for all the versions of the models to verify how well each of the classes is recognized. Classes 2 (*bird*), 3 (*cat*), 4 (*deer*) and 5 (*dog*) seem to be the most problematic, especially for the 3rd model. On the other hand, classes 1 (*automobile*), 6 (*frog*) and 9 (*truck*) seem to be the easiest to recognize, regardless of the chosen architecture. Basic versions of models 1 and 2 work pretty well for all of the classes.

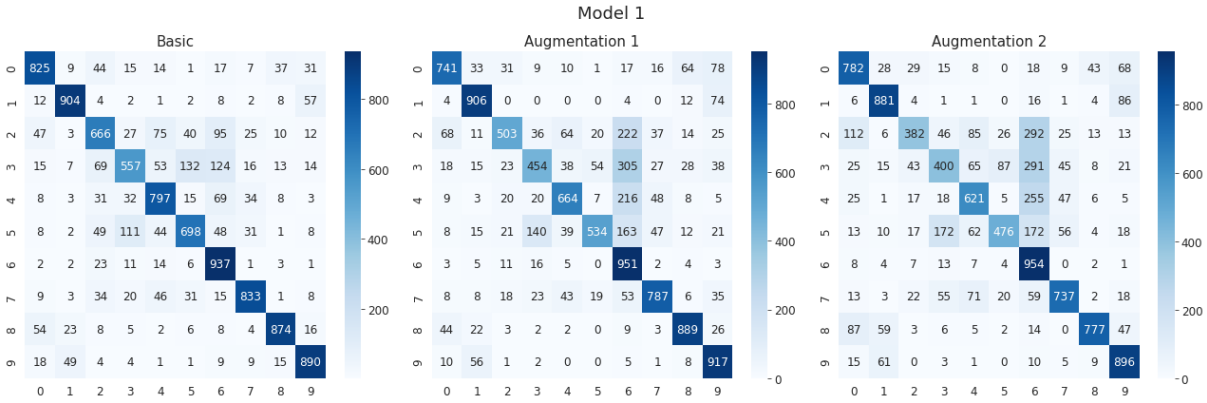

Figure 6: Confusion matrices of model 1 (basic and with augmentations).

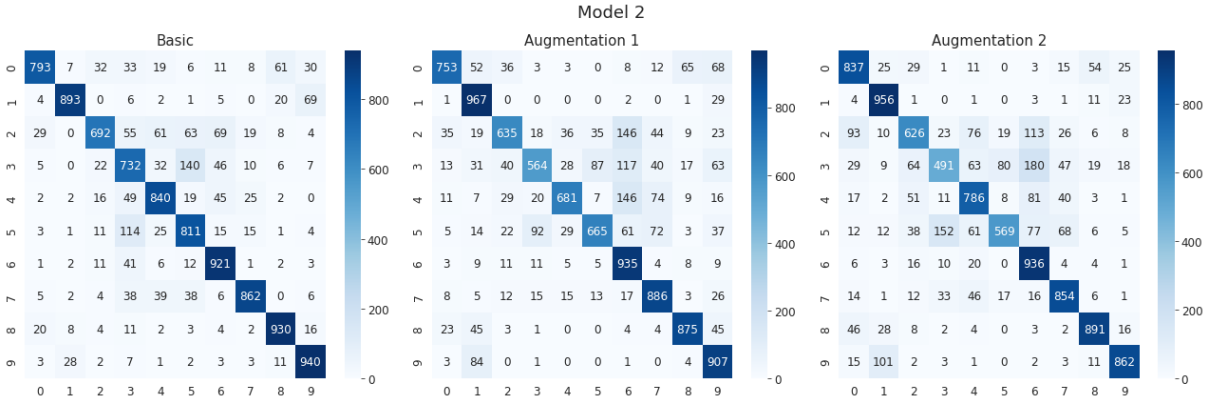

Figure 7: Confusion matrices of model 2 (basic and with augmentations).

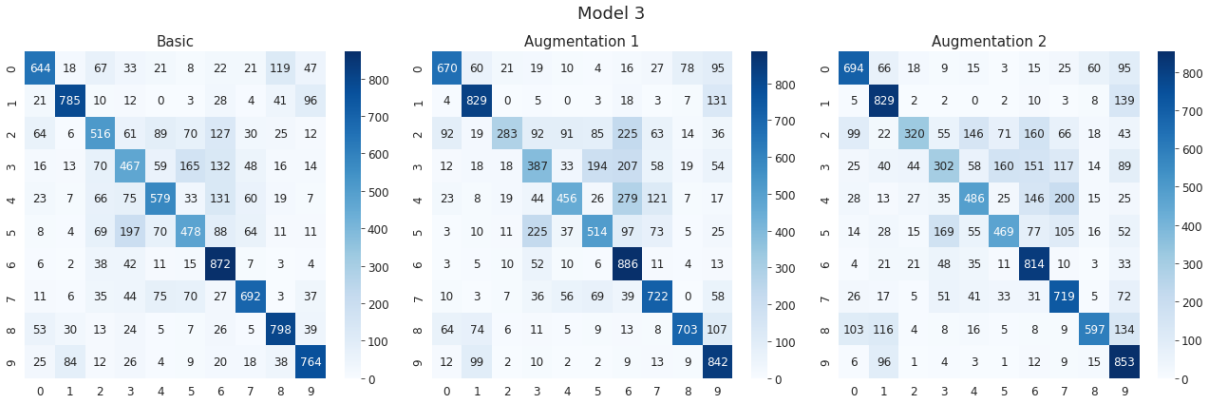

Figure 8: Confusion matrices of model 3 (basic and with augmentations).

5.2 Pretrained models

5.2.1 Experiment set-up

In the case of pre-trained models, unlike for custom architectures, the learning process was not repeated. The decision to train the models once was motivated by the full reproducibility of the results, which was a direct consequence of the fact that in the case of pre-trained models, the initial weights are non-random and the division into training and test set was constant. Additionally, in order to eliminate other random factors in the experimental environment, the

seed was arbitrarily set.

First, for both selected architectures, the learning process was carried out for the learning rate value, which was indicated as optimal in the case of custom models. Then, due to the poor results of ResNet, this one network was re-trained with a reduced value of this parameter.

5.2.2 Parametrization details

ResNet model

- epochs: 10
- batch size: 32
- learning rate: 1e-3 or 2e-5
- weights: imagenet
- activation: softmax
- loss: categorical_crossentropy
- optimizer: tf.keras.optimizers.RMSprop

VGG16 model

- epochs: 10
- batch size: 32
- learning rate: 1e-3
- weights: imagenet
- momentum: 0.9
- loss: binary_crossentropy
- optimizer: tf.keras.optimizers.SGD

5.2.3 Results of both pre-trained models with no Early Stopping callback

Residual neural network (ResNet)

	precision	recall	f1-score	support
0	0.77	0.65	0.70	1000
1	0.67	0.55	0.61	1000
2	0.72	0.51	0.59	1000
3	0.50	0.56	0.53	1000
4	0.63	0.73	0.67	1000
5	0.70	0.43	0.54	1000
6	0.61	0.86	0.71	1000
7	0.74	0.75	0.74	1000
8	0.81	0.78	0.80	1000
9	0.62	0.83	0.71	1000
accuracy			0.67	10000
macro avg	0.68	0.67	0.66	10000
weighted avg	0.68	0.67	0.66	10000

VGG16

	precision	recall	f1-score	support
0	0.86	0.89	0.87	1000
1	0.92	0.92	0.92	1000
2	0.77	0.83	0.80	1000
3	0.66	0.73	0.69	1000
4	0.81	0.82	0.81	1000
5	0.83	0.69	0.75	1000
6	0.92	0.82	0.87	1000
7	0.86	0.86	0.86	1000
8	0.93	0.92	0.93	1000
9	0.87	0.92	0.90	1000
accuracy			0.84	10000
macro avg	0.84	0.84	0.84	10000
weighted avg	0.84	0.84	0.84	10000

Figure 9: ResNet and VGG16 classification summary for learning rate = 1e-3

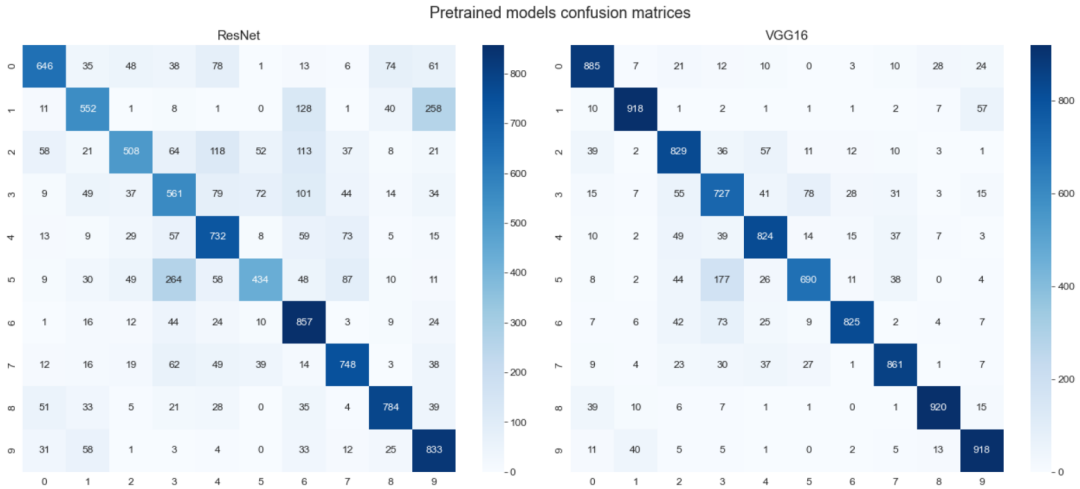

Figure 10: ResNet and VGG16 confusion matrices for learning rate = 1e-3

5.2.4 Results of the `ResNet` model with no `Early Stopping` callback after the `learning rate` value change

	precision	recall	f1-score	support
0	0.81	0.82	0.82	1000
1	0.86	0.86	0.86	1000
2	0.77	0.70	0.73	1000
3	0.59	0.63	0.61	1000
4	0.72	0.75	0.73	1000
5	0.69	0.68	0.68	1000
6	0.83	0.84	0.83	1000
7	0.84	0.81	0.83	1000
8	0.87	0.86	0.86	1000
9	0.84	0.84	0.84	1000
accuracy			0.78	10000
macro avg	0.78	0.78	0.78	10000
weighted avg	0.78	0.78	0.78	10000

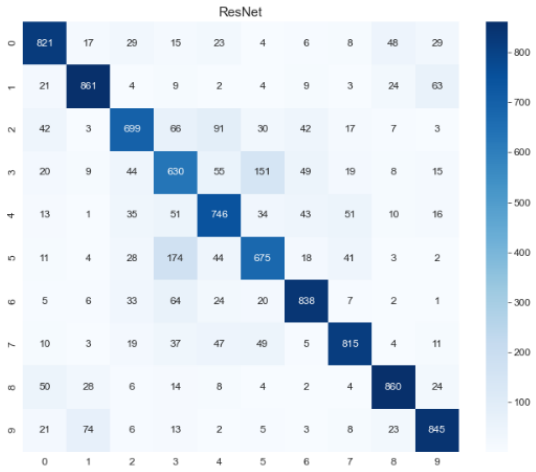

Figure 11: `ResNet` confusion matrix for `learning rate = 2e-5`

5.2.5 Results of both pre-trained models with `Early Stopping` callback

```
Epoch 1/10
1563/1563 [==============================] - 2701s 2s/step - loss: 2.3281 - accuracy: 0.4085 - val_loss: 2.2163 - val_accuracy:
0.3313
Epoch 2/10
1563/1563 [==============================] - 1839s 1s/step - loss: 1.7368 - accuracy: 0.5525 - val_loss: 1.9985 - val_accuracy:
0.3717
```

Figure 12: Plots showing ResNet network tuning process details for `learning rate = 1e-3`

```
Epoch 1/10
1562/1562 [==============================] - 1513s 968ms/step - loss: 0.1810 - accuracy: 0.6177 - val_loss: 0.1450 - val_accuracy: 0.6959
Epoch 2/10
1562/1562 [==============================] - 1493s 956ms/step - loss: 0.1211 - accuracy: 0.7545 - val_loss: 0.1138 - val_accuracy: 0.7726
```

Figure 13: Plots showing VGG16 network tuning process details for `learning rate = 1e-3`

Residual neural network (ResNet)

	precision	recall	f1-score	support
0	0.25	0.83	0.39	1000
1	0.55	0.34	0.42	1000
2	0.35	0.20	0.25	1000
3	0.31	0.24	0.27	1000
4	0.48	0.17	0.25	1000
5	0.52	0.13	0.21	1000
6	0.72	0.23	0.35	1000
7	0.37	0.29	0.32	1000
8	0.62	0.43	0.51	1000
9	0.37	0.86	0.52	1000
accuracy			0.37	10000
macro avg	0.45	0.37	0.35	10000
weighted avg	0.45	0.37	0.35	10000

VGG16

	precision	recall	f1-score	support
0	0.82	0.79	0.81	1000
1	0.87	0.89	0.88	1000
2	0.74	0.67	0.70	1000
3	0.61	0.59	0.60	1000
4	0.78	0.63	0.70	1000
5	0.72	0.68	0.70	1000
6	0.76	0.86	0.81	1000
7	0.79	0.82	0.81	1000
8	0.86	0.89	0.87	1000
9	0.76	0.91	0.83	1000
accuracy			0.77	10000
macro avg	0.77	0.77	0.77	10000
weighted avg	0.77	0.77	0.77	10000

Figure 14: `ResNet` and `VGG16` classification summary for `learning rate = 1e-3`

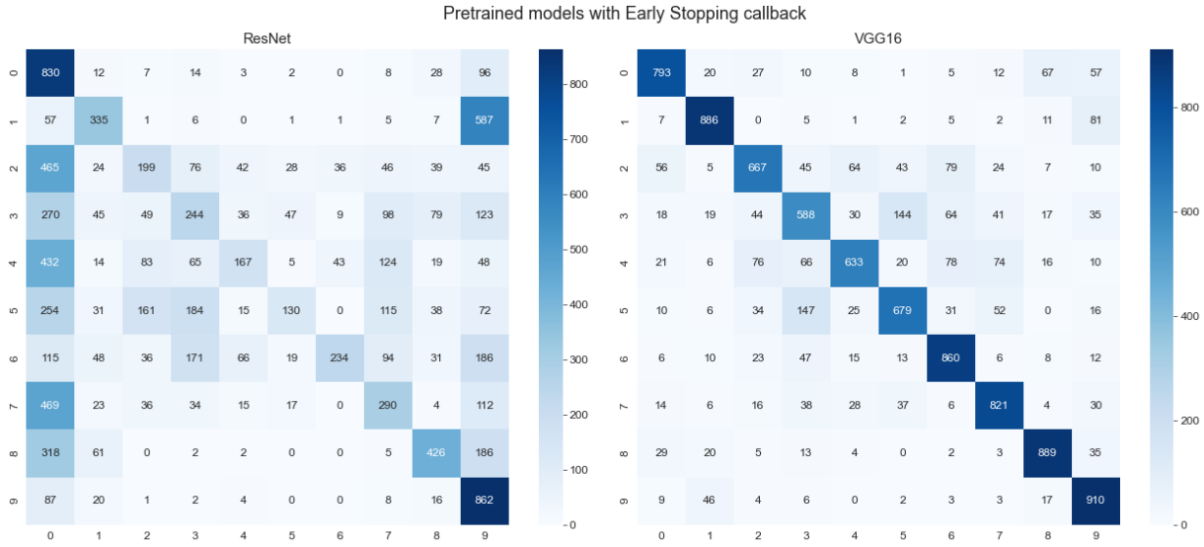

Figure 15: ResNet and VGG16 confusion matrices for `learning rate = 1e-3`

5.2.6 Results of the `ResNet` model with `Early Stopping` callback after the `learning rate` value change

```
Epoch 1/10
1563/1563 [==============================] - 1742s 1s/step - loss: 1.8825 - accuracy: 0.4032 - val_loss: 1.3141 - val_accuracy: 0.5806
Epoch 2/10
1563/1563 [==============================] - 1699s 1s/step - loss: 1.0816 - accuracy: 0.6350 - val_loss: 1.0139 - val_accuracy: 0.6806
<keras.callbacks.History at 0x1dc4d906a60>
```

Figure 16: Plots showing ResNet network tuning process details for `learning rate = 2e-5`

	precision	recall	f1-score	support
0	0.72	0.70	0.71	1000
1	0.79	0.78	0.78	1000
2	0.61	0.59	0.60	1000
3	0.51	0.53	0.52	1000
4	0.63	0.58	0.60	1000
5	0.60	0.57	0.59	1000
6	0.74	0.74	0.74	1000
7	0.72	0.72	0.72	1000
8	0.74	0.81	0.77	1000
9	0.74	0.79	0.76	1000
accuracy			0.68	10000
macro avg	0.68	0.68	0.68	10000
weighted avg	0.68	0.68	0.68	10000

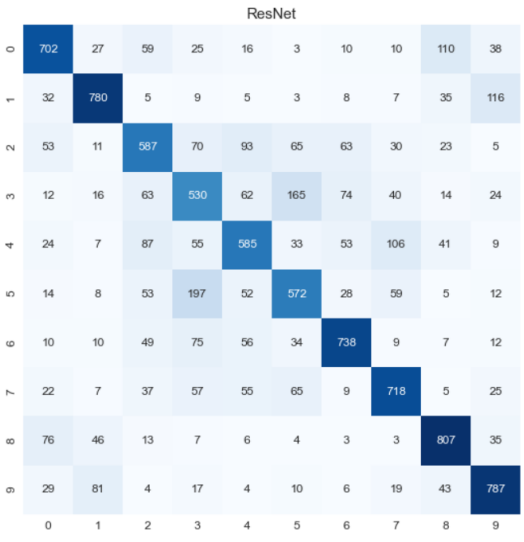

Figure 17: `ResNet` confusion matrix for `learning rate = 2e-5`

6 Conclusions

In case of custom models, model 2 turned out to perform best on the analyzed dataset, achieving the highest accuracy and having the best-looking confusion matrix (there were not any significantly worse-recognized classes). Generally, data augmentation does not improve the results of the proposed models but prevented quick overfitting. Unfortunately, it also led to a higher standard deviation of the results. For all the models, classes 2 (*bird*), 3 (*cat*), 4 (*deer*) and 5 (*dog*) seemed to be the most problematic, and classes 1 (*automobile*), 6 (*frog*) and 9 (*truck*) were the easiest to recognize.

In case of pretrained models, ResNet has much better results for `learning rate = 2e-5`, although it starts overfitting practically at the same time as with the `learning rate = 1e-3` value - in both cases early-stopping stops learning after 2 epochs.
For both methods, the most difficult to recognize classes are indexed with numbers from the range $\{2, 3\}$ (2 - *bird*, 3 - *cat*). On the other hand, the highest efficiency of identification can be noticed for $\{1, 8\}$ classes, i.e. 1-automobile, 8-ship.