Deep Learning

PROJECT 1 - REPORT

March 28, 2023

Maciej Chylak, Agata Kaczmarek

Contents

1 Description of the problem

In this project, we implemented different convolutional neural network architectures and used them to solve the image classification problem for dataset CIFAR-10 [1]. Later we compared the obtained models to the built-in solution - ResNet18 [3]. Additionally, during the work on this problem, we checked the influence of hyperparameters change and data augmentation on the final effectiveness of the neural network. In the end, we formed a committee of neural networks.

2 Instruction of the application

All classes and functions in our project are implemented in one Jupyter notebook. We chose this setting, as in that way we can show examples of the usage right after code implementation. In our notebook examples of how to run each function and some of the results can be found. To run the code, Python 3.9 is recommended. All packages listed in the two first cells in the notebook are required to run all experiments.

3 Theoretical introduction

For this project, we trained Convolutional Neural Networks (CNN) with various hyperparameters. We trained them on CIFAR-10 dataset, which was augmented with different techniques. Short theoretical introduction to used techniques is described below.

3.1 Convolutional Neural Networks

Convolutional Neural Netowrks is a type of artificial neural netowrk which is most commonly used in image analysis. It works in such a way that it takes input images, then by updating the weights and biases it gives relevance to individual features in the image, so it is able to distinguish one class from another. This process is not computationally complex due to the use of appropriate filters for image data reduction, which preserve key image properties. Its advantage over the multilayer perceptron is that it is able to catch relationships between adjacent pixels in an image by applying relevant filters.

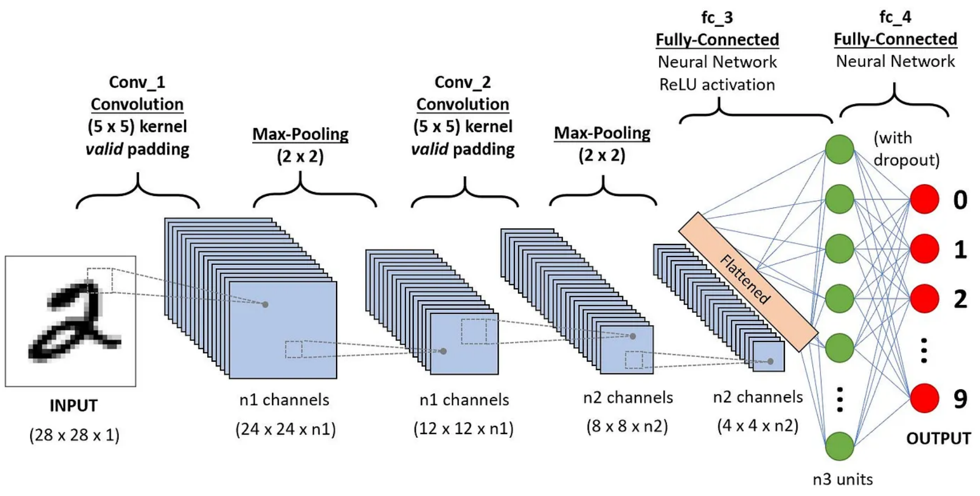


Figure 1: Layers in Convolutional Neural Network (source of the image).

A convolutional neural network consists of several layers which we can see in figure 1:

- convolutional kernel valid padding - aims to extract high-level features such as edges from the input image, resulting in a reduction in dimensionality
- max pooling - aims to extract dominant features which are rotational and positional invariant, also resulting in a reduction in dimensionality
- fully connected - helps in learning non-linear combinations of high-level features

3.2 ResNet18

ResNet is one of the most popular CNN-type neural networks. The number which is located at the end of the name means how many deep layers are inside. Thanks to the use of skip connections that bypass one or more layers, the most troublesome problem in deep neural networks - the vanishing gradient - has been solved.

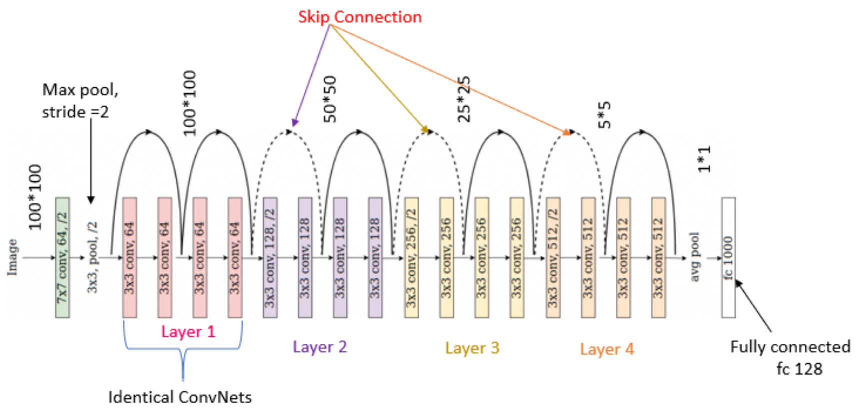


Figure 2: Architecture of ResNet18 (source of the image).

In figure 2 we can see exactly how the respective layers look and how the skip connections are implemented.

3.3 CIFAR-10

The CIFAR-10 [1] dataset consists of 60 000 images, each of them representing one out of 10 classes, which are equally numerous. The classes are as follows: airplane, automobile (car), bird, cat, deer, dog, frog, horse, ship, truck.

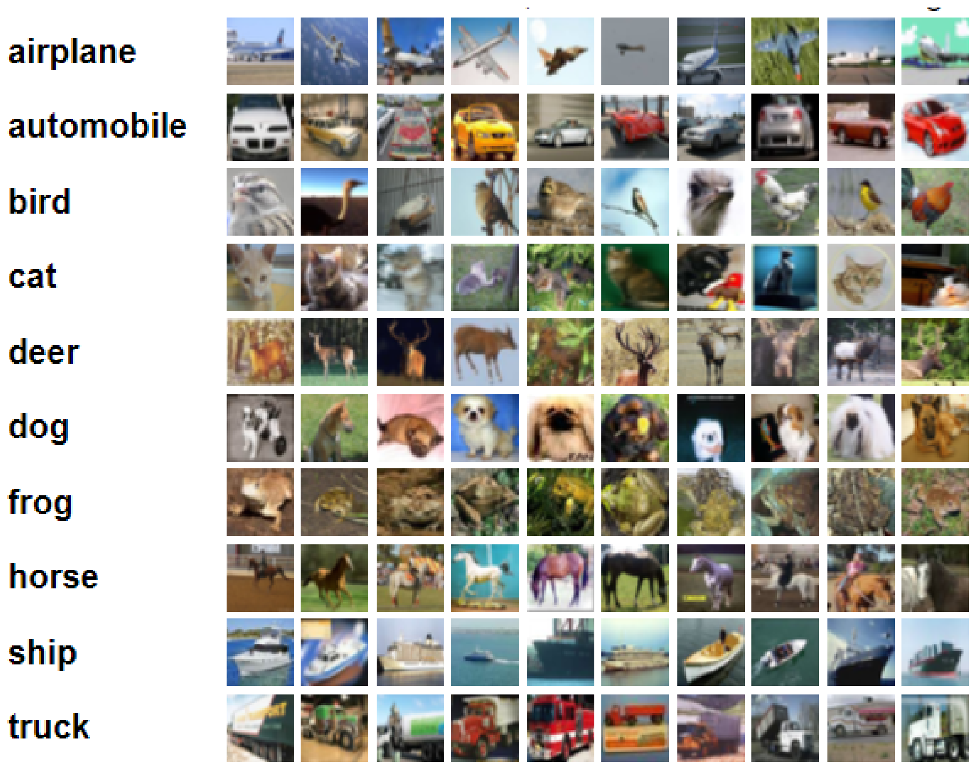


Figure 3: Examples of images representing all 10 classes from CIFAR-10 dataset (source of the image).

3.4 Hyperparameters

Hyperparameters in general are used to control models' learning process. In our project we decided to check the influence of four various, two related to the training process (learning rate and batch size) and two related to regularisation (probability for dropout and L2 regularisation).

3.4.1 Learning rate

This hyperparameter controls how much do weights of a network change, with respect to the estimated error. If the chosen learning rate is too small, the training will be unnecessary longer. However, also too big value is not recommended - in that case the training process can be unstable.

3.4.2 Batch size

Batch size is the number of images from the dataset, that are passed to the network at the same time. If we have 1000 images, and the batch size equals to 10, The epoch contains 100 runs and updates of weights for the network. In general, the larger the batch size, the less time network needs to complete each epoch of the training process. However, this can cause worse metrics results of the model.

3.4.3 L2 regularisation

The regularisation methods for neural networks training are supposed to help in generalization of the model, without significant change in the training error. One of the methods is L2 regularisation (also called ridge regression). The method gives the model penalty for having too big weights.

$$L_2 = \frac{1}{m}\sum_{i=1}^{m} L(y_i, \hat{y}_i) + \frac{\lambda}{2m}||w||^2 \tag{1}$$

3.4.4 Dropout

Another one of the reqularisation techniques, its main goal is to prevent over-fitting of the neural network. With given probability some neurons are used and only they take part in one following learning step. Next step, next neurons are randomly chosen to be used. The result is, that the model is more robust on the noise in dataset. Single epoch in training process takes less time, however, the model needs more epochs to train.

3.5 Augmentation techniques

In general, if we want a model to be better, we train it on more data. In order to make the model generalize better, not only have a lower errors on train data, we should show the model more, but various data. The goal of the augmentation is to generate more samples of data, without having to collect them. It is faster, less complicated and also a cheaper option.

3.5.1 Rotations

The images from original dataset are simply rotated, but the label remains the same.

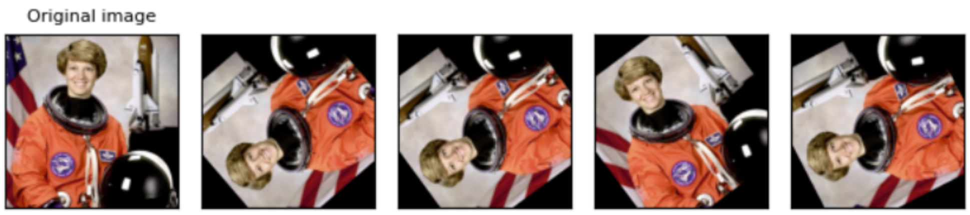


Figure 4: Examples of image after rotation (source of the image).

3.5.2 Horizontal flipping

The image from the original dataset is flipped horizontally. The label remains the same.

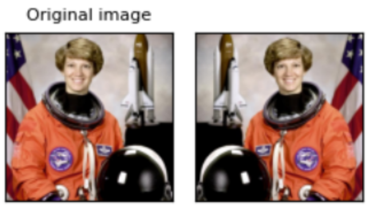


Figure 5: Original image and image after horizontal flipping (source of the image).

3.5.3 Convert image to grayscale

The original image is converted into grayscale. The label remains the same.

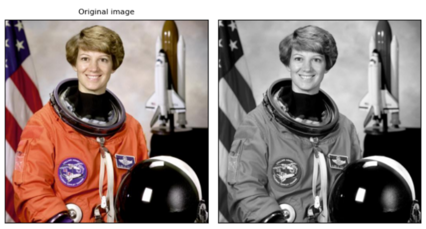


Figure 6: Original image and image after changing into grayscale (source of the image).

3.5.4 CutMix

The images of the original dataset are cut into smaller pieces and then stuck together in a mixed way. Also, labels are mixed proportionally to the parts of the image from various classes.

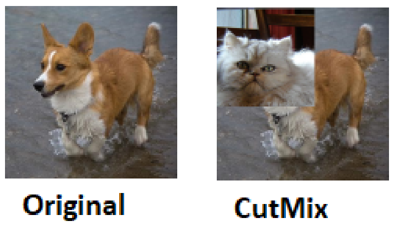


Figure 7: Original image and image after CutMix (source of images).

3.6 Ensemble of the models

Ensemble of models is one of the performance enhancement techniques involving training a set of neural networks and then solving the problem using the aggregated response. There are several ways to vary the set of neural networks. We can consider different architectures and train them on the same training set, and we can also consider the same architecture and train networks on subsets of the training set.

Among the techniques for aggregating responses, we can distinguish:

- majority vote - the dominant response is selected - used in classification problems
- averaging of responses - the average answer is chosen - used in regression problems

4 Experiments

To check the influence of the chosen hyperparameters and augmentation techniques on models accuracy, we conducted several experiments, which are described in details below.

4.1 Chosen technology

For this project, we used the Python language and its library, which supports the creation of neural networks and work with them - PyTorch. It also allows users to use both CPU and GPU for neural networks training.

4.2 Used architectures

During this project we worked with three different architectures of neural networks. Two simpler architectures were inspired by propositions of suitable models for the CIFAR 10 dataset in source 1 and source 2. The third architecture used is ResNet18 source 3, we used both not pre-trained and pre-trained version. The pre-trained one was supposed to be a baseline for other our solutions.

Therefore the architectures we decided to use are as follows:

- simple convolutional neural network (simple CNN)
  - convolutional layer with ReLU as activation function interspersed with max-pooling layers two times
  - flattening layer
  - two fully connected layers with ReLU as an activation function
  - fully connected layer

- more complex neural network (complex CNN)
  - two layers each convolutional layer interspersed with max pooling and dropout layer two times
  - flattening layer
  - dense layer with softmax as an activation function
- not pre-trained ResNet18
- pre-trained ResNet18

Due to the fact, that while conducting grid search of best hyperparameters, for some models the training process was stopping at some level, we decided to implement basic early stopping. If the changes in consecutive measurements of test accuracy are too small, the training is stopped.

4.3 Tested hyperparameters

During the work on this project, we tested influence of four hyperparameters on models bahaviour. These hyperparameters were learning rate, batch size, probability for dropout and L2 regularisation.

4.4 Data augmentation techniques

We also tested the influence of data augmentation techniques on the model's behaviour. We tested various rotations, horizontal flipping, convertion image to grayscale and also cutmix.

4.5 Description of conducted experiments

4.5.1 Single models

Before conducting all other experiments, we decided to train one, baseline model for each of the architectures. We did this to see what kind of performance, we will be looking for later.

Simple CNN Simple CNN models do not have implemented dropout. At the beginning of the experiments, we trained a baseline model with the following hyperparameters:

- batch size - 16
- learning rate - 0.001
- transformation - base (no changes in images)
- dropout - not set
- L2 lambda - 0

We conducted the following experiments for simple CNN:

1. Grid search over various hyperparameters. Goal: Finding two best models, to perform further experiments on them.
2. For two best settings of hyperparameters, perform five different augmentation techniques: base, rotations, horizontal flipping, converting image to grayscale and CutMix. Goal: finding out if the augmentation has any influence on models accuracy.
3. Influence of momentum on training process.
4. Check, if early stopping implementation was needed.

For experiment 1 search was conducted on hyperparameters shown in table 1.

Hyperparameter	Grid
batch size	[4, 8, 16]
learning rate	[0.001, 0.05, 0.01]
transformation	[base, random rotation, random horizontal flip, grayscale, cutmix]
L2 lambda	[0.1, 0.001, 0.0001]

Table 1: Hyperparameter grid for experiments with simple CNN.

For experiment 2, we used following simple CNNs:

- batch size: 8, learning rate: 0.001, L2: 0.001, transformation: base,
- batch size: 4, learning rate: 0.001, L2: 0.0001, transformation: base.

Complex CNN At the beginning of the experiments, we trained a baseline model with following hyperparameters:

- batch size - 16
- learning rate - 0.01
- transformation - base (no changes in images)
- dropout - 0
- L2 lambda - 0

We conducted the following experiments for complex CNN:

1. Grid search over various hyperparameters. Goal: Finding two best models, to perform further experiments on them.
2. For two best settings of hyperparameters, perform five different augmentation techniques: base, rotations, horizontal flipping, converting image to grayscale and CutMix. Goal: finding out if the augmentation has influence on models accuracy.
3. During conducting other experiments, we saw, that there might be a correlation between batch size and both training time and accuracy. Goal: check if it is true.

For experiment 1 search was conducted on hyperparameters shown in table 2.
For experiment 2, we used following complex CNNs:

- batch size: 8, learning rate: 0.001, L2: 0.001, transformation: base,
- batch size: 4, learning rate: 0.001, L2: 0.0001, transformation: base.

For experiment 3, we used complex CNN with every hyperparameter the same, only batch size was either 4 or 16.

Hyperparameter	Grid
batch size	[4, 8]
learning rate	[0.001, 0.05, 0.01]
transformation	[base, random rotation, random horizontal flip, grayscale, cutmix]
L2 lambda	[0.1, 0.001, 0.0001]

Table 2: Hyperparameter grid for experiments with complex CNN.

Not pre-trained ResNet18 The experiments for both ResNets were conducted against a baseline model that included the following set of hyperparameters:

- batch size - 8
- learning rate - 0.003
- dropout - 0
- transformation - base
- L2 lambda - 0

1. Changing one parameter against the baseline by the values given in the table 3. Goal: check which value of a particular hyperparameter will give the best results

During the ResNet experiments, we chose to compare the change in hyperparameters singly against the baseline model. We did this because the training time of the ResNets was much longer than the networks we created, so there was no real opportunity to do this.

Pre-trained ResNet18

1. Changing one parameter against the baseline by the values given in the table 3. Goal: check which value of a particular hyperparameter will give the best results

As in the case of the not pre-trained ResNet, we decided to train our nets by changing only one hyperparameter at a time because of the large training time.

Hyperparameter	Grid
batch size	[4, 8, 16, 32]
learning rate	[0.001, 0.003, 0.01, 0.05, 0.1]
dropout	[0, 0.2, 0.4, 0.6, 0.8]
transformation	[base, random rotation, random horizontal flip, grayscale, cutmix]
L2 lambda	[0.1, 0.001, 0.0001, 0]

Table 3: Hyperparameter grid for experiments with ResNet18.

4.5.2 Ensemble

Additionally, we implemented the committee of neural networks. Final architecture of this solution was, that it contains multiple models of the same architecture, trained on the whole dataset, but with different seed set. Afterwards, there is a voting procedure, to chose the output of the ensemble, which is the most common value from predictions of models for given image.

In our experiments, we run the ensemble two times, for simple CNNs with slightly changed hyperparameters:

- batch: 8, learning rate: 0.001, transformation: random horizontal flip, L2: 0.001, momentum: 0
- batch: 8, learning rate: 0.001, transformation: random horizontal flip, L2: 0.001, momentum: 0.9

5 Results

5.1 Evaluation of the solution

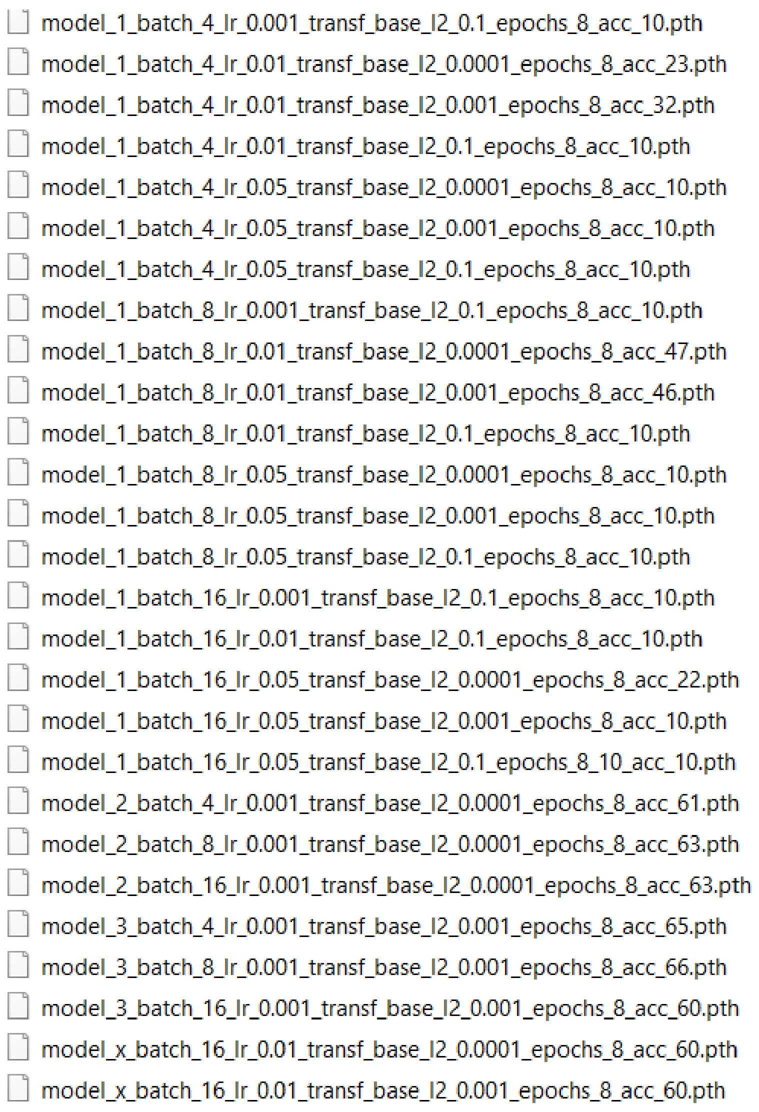
Results of conducted experiments were evaluated either according to accuracy, the models achieved on test set or with a confusion matrix.

5.2 Obtained results

Below we present the results of conducted experiments.

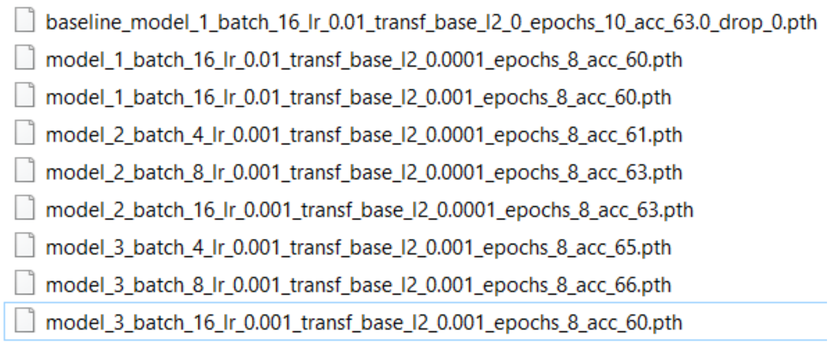
5.2.1 Simple CNN

Experiment 1 As can be seen on 8 and 9, there happened to be specific hyperparameters setting, which were giving better results than others. Based on this results we decided which models will be trained during experiment 2 (no matter batch size, they gave pretty good results). We also decided to introduce early stopping, as nearly half of trained models did not start learning, based on accuracy, but tried for a long time. We also think, that there can be a problem with L2 = 0.1 for simple CNN, as none of the models with this, was able to start training.



```
model_1_batch_4_lr_0.001_transf_base_l2_0.1_epochs_8_acc_10.pth
model_1_batch_4_lr_0.01_transf_base_l2_0.0001_epochs_8_acc_23.pth
model_1_batch_4_lr_0.01_transf_base_l2_0.001_epochs_8_acc_32.pth
model_1_batch_4_lr_0.01_transf_base_l2_0.1_epochs_8_acc_10.pth
model_1_batch_4_lr_0.05_transf_base_l2_0.0001_epochs_8_acc_10.pth
model_1_batch_4_lr_0.05_transf_base_l2_0.001_epochs_8_acc_10.pth
model_1_batch_4_lr_0.05_transf_base_l2_0.1_epochs_8_acc_10.pth
model_1_batch_8_lr_0.001_transf_base_l2_0.1_epochs_8_acc_10.pth
model_1_batch_8_lr_0.01_transf_base_l2_0.0001_epochs_8_acc_47.pth
model_1_batch_8_lr_0.01_transf_base_l2_0.001_epochs_8_acc_46.pth
model_1_batch_8_lr_0.01_transf_base_l2_0.1_epochs_8_acc_10.pth
model_1_batch_8_lr_0.05_transf_base_l2_0.0001_epochs_8_acc_10.pth
model_1_batch_8_lr_0.05_transf_base_l2_0.001_epochs_8_acc_10.pth
model_1_batch_8_lr_0.05_transf_base_l2_0.1_epochs_8_acc_10.pth
model_1_batch_16_lr_0.001_transf_base_l2_0.1_epochs_8_acc_10.pth
model_1_batch_16_lr_0.01_transf_base_l2_0.1_epochs_8_acc_10.pth
model_1_batch_16_lr_0.05_transf_base_l2_0.0001_epochs_8_acc_22.pth
model_1_batch_16_lr_0.05_transf_base_l2_0.001_epochs_8_acc_10.pth
model_1_batch_16_lr_0.05_transf_base_l2_0.1_epochs_8_10_acc_10.pth
model_2_batch_4_lr_0.001_transf_base_l2_0.0001_epochs_8_acc_61.pth
model_2_batch_8_lr_0.001_transf_base_l2_0.0001_epochs_8_acc_63.pth
model_2_batch_16_lr_0.001_transf_base_l2_0.0001_epochs_8_acc_63.pth
model_3_batch_4_lr_0.001_transf_base_l2_0.001_epochs_8_acc_65.pth
model_3_batch_8_lr_0.001_transf_base_l2_0.001_epochs_8_acc_66.pth
model_3_batch_16_lr_0.001_transf_base_l2_0.001_epochs_8_acc_60.pth
model_x_batch_16_lr_0.01_transf_base_l2_0.0001_epochs_8_acc_60.pth
model_x_batch_16_lr_0.01_transf_base_l2_0.001_epochs_8_acc_60.pth
```

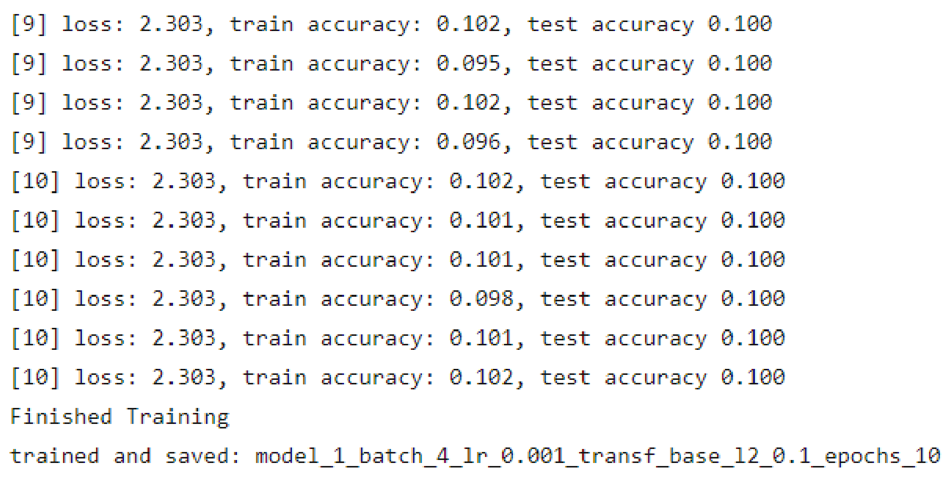
Figure 8: All simple CNNs trained while conducting grid search of the hyperparameters.



```
baseline_model_1_batch_16_lr_0.01_transf_base_l2_0_epochs_10_acc_63.0_drop_0.pth
model_1_batch_16_lr_0.01_transf_base_l2_0.0001_epochs_8_acc_60.pth
model_1_batch_16_lr_0.01_transf_base_l2_0.001_epochs_8_acc_60.pth
model_2_batch_4_lr_0.001_transf_base_l2_0.0001_epochs_8_acc_61.pth
model_2_batch_8_lr_0.001_transf_base_l2_0.0001_epochs_8_acc_63.pth
model_2_batch_16_lr_0.001_transf_base_l2_0.0001_epochs_8_acc_63.pth
model_3_batch_4_lr_0.001_transf_base_l2_0.001_epochs_8_acc_65.pth
model_3_batch_8_lr_0.001_transf_base_l2_0.001_epochs_8_acc_66.pth
model_3_batch_16_lr_0.001_transf_base_l2_0.001_epochs_8_acc_60.pth
```

Figure 9: Simple CNNs with better results trained while conducting grid search of the hyperparameters.

On figure 10 can be seen, how the end of the training process, of not training model, looked like.



```
[9] loss: 2.303, train accuracy: 0.102, test accuracy 0.100
[9] loss: 2.303, train accuracy: 0.095, test accuracy 0.100
[9] loss: 2.303, train accuracy: 0.102, test accuracy 0.100
[9] loss: 2.303, train accuracy: 0.096, test accuracy 0.100
[10] loss: 2.303, train accuracy: 0.102, test accuracy 0.100
[10] loss: 2.303, train accuracy: 0.101, test accuracy 0.100
[10] loss: 2.303, train accuracy: 0.101, test accuracy 0.100
[10] loss: 2.303, train accuracy: 0.098, test accuracy 0.100
[10] loss: 2.303, train accuracy: 0.101, test accuracy 0.100
[10] loss: 2.303, train accuracy: 0.102, test accuracy 0.100
Finished Training
trained and saved: model_1_batch_4_lr_0.001_transf_base_l2_0.1_epochs_10
```

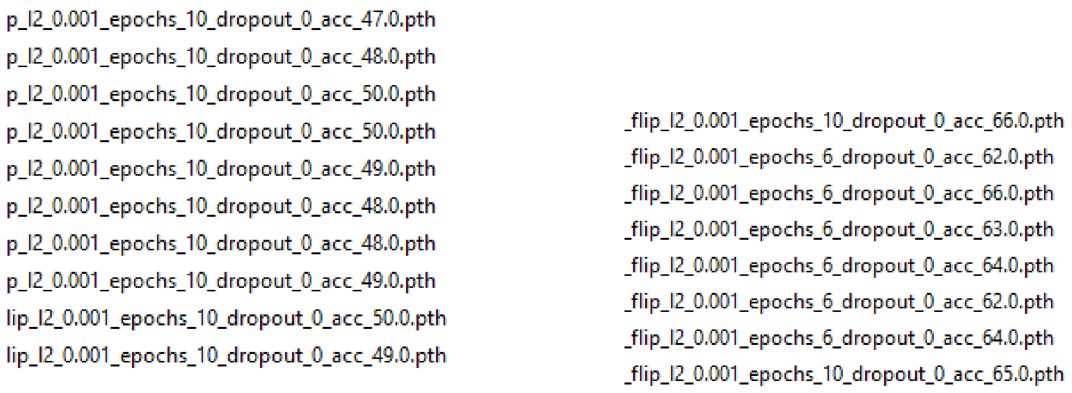
Figure 10: Simple CNN that was not getting better while training process.

Experiment 2 For chosen two best settings, we conducted training of new models, one model with one augmentation, in total ten new models. We repeated this three times. The results of the experiments can be found in table 4. In most cases, the results were similar for models trained on base and any other augmentation, except CutMix. However, this might be due to the fact, that for CutMix, the label is not just one class name, but more, so for a random model the accuracy is smaller than 10 accuracy. What is also interesting, models with horizontal flip seems to have the biggest stdev of accuracy than the others, whereas the mean is similar to other.

Model	Iter 1	Iter 2	Iter 3	Avg	Stdev
Batch: 4 trans: **base**	63	61	62	**62**	0.82
Batch: 4 trans: **grayscale**	61	61	63	61.7	0.94
Batch: 4 trans: **horizontal flip**	65	61	61	62.3	**1.89**
Batch: 4 trans: **rotation**	60	62	62	61.3	0.94
Batch: 4 trans: **cutmix**	54	54	56	**54.7**	0.94
Batch: 8 trans: **base**	66	59	63	**62.7**	**2.87**
Batch: 8 trans: **grayscale**	64	61	63	62.7	1.24
Batch: 8 trans: **horizontal flip**	65	58	62	61.7	**2.87**
Batch: 8 trans: **rotation**	62	61	62	61.7	0.47
Batch: 8 trans: **cutmix**	53	54	55	**54**	0.82

Table 4: Experiment 2 on simple CNN.

Experiment 3 While training models we found out, that the momentum has great influence on the final accuracy of the models, as can be seen on 11. Thanks to this knowledge, our final models in ensemble had momentum hyperparameter set to 0.9.

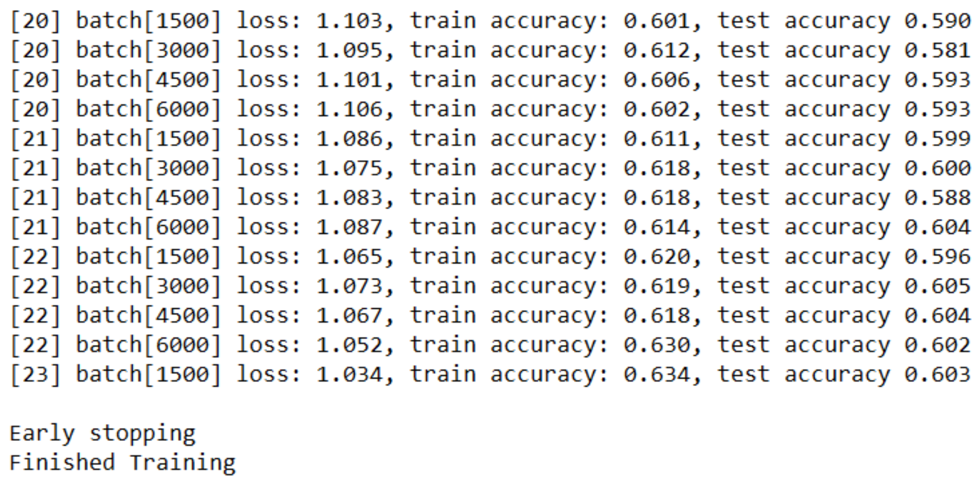


```
p_l2_0.001_epochs_10_dropout_0_acc_47.0.pth
p_l2_0.001_epochs_10_dropout_0_acc_48.0.pth
p_l2_0.001_epochs_10_dropout_0_acc_50.0.pth
p_l2_0.001_epochs_10_dropout_0_acc_50.0.pth
p_l2_0.001_epochs_10_dropout_0_acc_49.0.pth
p_l2_0.001_epochs_10_dropout_0_acc_48.0.pth
p_l2_0.001_epochs_10_dropout_0_acc_48.0.pth
p_l2_0.001_epochs_10_dropout_0_acc_49.0.pth
lip_l2_0.001_epochs_10_dropout_0_acc_50.0.pth
lip_l2_0.001_epochs_10_dropout_0_acc_49.0.pth
```

```
_flip_l2_0.001_epochs_10_dropout_0_acc_66.0.pth
_flip_l2_0.001_epochs_6_dropout_0_acc_62.0.pth
_flip_l2_0.001_epochs_6_dropout_0_acc_66.0.pth
_flip_l2_0.001_epochs_6_dropout_0_acc_63.0.pth
_flip_l2_0.001_epochs_6_dropout_0_acc_64.0.pth
_flip_l2_0.001_epochs_6_dropout_0_acc_62.0.pth
_flip_l2_0.001_epochs_6_dropout_0_acc_64.0.pth
_flip_l2_0.001_epochs_10_dropout_0_acc_65.0.pth
```

Figure 11: Results of the same models trained with momentum 0 (left) and momentum 0.9 (right).

Experiment 4 While training consecutive models, for example in grid search or later for others, not so successful, experiments, we saw that our implementation of early stopping is working, as shown on 12.



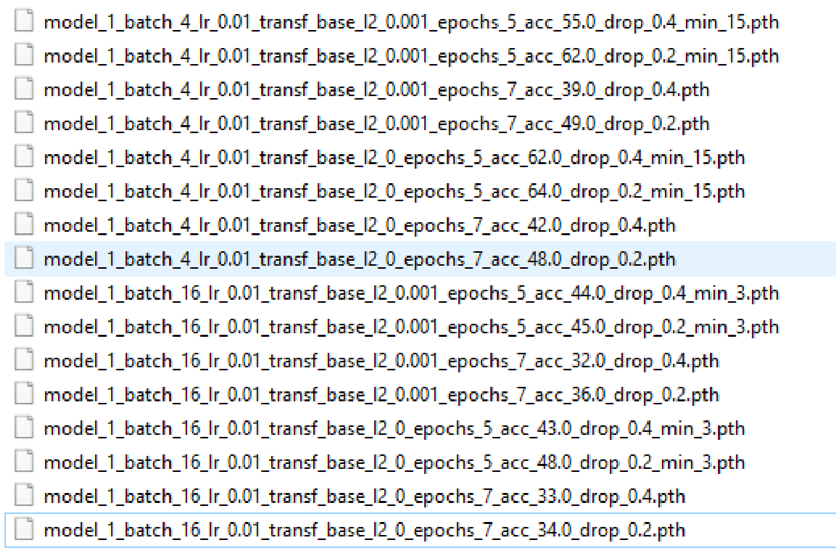
```
[20] batch[1500] loss: 1.103, train accuracy: 0.601, test accuracy 0.590
[20] batch[3000] loss: 1.095, train accuracy: 0.612, test accuracy 0.581
[20] batch[4500] loss: 1.101, train accuracy: 0.606, test accuracy 0.593
[20] batch[6000] loss: 1.106, train accuracy: 0.602, test accuracy 0.593
[21] batch[1500] loss: 1.086, train accuracy: 0.611, test accuracy 0.599
[21] batch[3000] loss: 1.075, train accuracy: 0.618, test accuracy 0.600
[21] batch[4500] loss: 1.083, train accuracy: 0.618, test accuracy 0.588
[21] batch[6000] loss: 1.087, train accuracy: 0.614, test accuracy 0.604
[22] batch[1500] loss: 1.065, train accuracy: 0.620, test accuracy 0.596
[22] batch[3000] loss: 1.073, train accuracy: 0.619, test accuracy 0.605
[22] batch[4500] loss: 1.067, train accuracy: 0.618, test accuracy 0.604
[22] batch[6000] loss: 1.052, train accuracy: 0.630, test accuracy 0.602
[23] batch[1500] loss: 1.034, train accuracy: 0.634, test accuracy 0.603

Early stopping
Finished Training
```

Figure 12: Training process stopped by early stopping.

5.2.2 Complex CNN

Experiment 1 As can be seen on 13, there happened to be specific hyperparameters setting, which was giving better results than others. Based on these results we decided which models will be trained during experiment 2.



```
model_1_batch_4_lr_0.01_transf_base_l2_0.001_epochs_5_acc_55.0_drop_0.4_min_15.pth
model_1_batch_4_lr_0.01_transf_base_l2_0.001_epochs_5_acc_62.0_drop_0.2_min_15.pth
model_1_batch_4_lr_0.01_transf_base_l2_0.001_epochs_7_acc_39.0_drop_0.4.pth
model_1_batch_4_lr_0.01_transf_base_l2_0.001_epochs_7_acc_49.0_drop_0.2.pth
model_1_batch_4_lr_0.01_transf_base_l2_0_epochs_5_acc_62.0_drop_0.4_min_15.pth
model_1_batch_4_lr_0.01_transf_base_l2_0_epochs_5_acc_64.0_drop_0.2_min_15.pth
model_1_batch_4_lr_0.01_transf_base_l2_0_epochs_7_acc_42.0_drop_0.4.pth
model_1_batch_4_lr_0.01_transf_base_l2_0_epochs_7_acc_48.0_drop_0.2.pth
model_1_batch_16_lr_0.01_transf_base_l2_0.001_epochs_5_acc_44.0_drop_0.4_min_3.pth
model_1_batch_16_lr_0.01_transf_base_l2_0.001_epochs_5_acc_45.0_drop_0.2_min_3.pth
model_1_batch_16_lr_0.01_transf_base_l2_0.001_epochs_7_acc_32.0_drop_0.4.pth
model_1_batch_16_lr_0.01_transf_base_l2_0.001_epochs_7_acc_36.0_drop_0.2.pth
model_1_batch_16_lr_0.01_transf_base_l2_0_epochs_5_acc_43.0_drop_0.4_min_3.pth
model_1_batch_16_lr_0.01_transf_base_l2_0_epochs_5_acc_48.0_drop_0.2_min_3.pth
model_1_batch_16_lr_0.01_transf_base_l2_0_epochs_7_acc_33.0_drop_0.4.pth
model_1_batch_16_lr_0.01_transf_base_l2_0_epochs_7_acc_34.0_drop_0.2.pth
```

Figure 13: All complex CNNs trained while conducting grid search of the hyperparameters.

Experiment 2 For chosen two best settings, we conducted training of new models, one model with one augmentation, in total ten new models. We repeated this three times. The results of the experiments can be found in table 5. This time the results were less stable, than for simple CNN. It is hard to choose one best hyperparameters setting here, but good candidate for this might be model trained with batch size 4 and rotation as transformation.

Model	Iter 1	Iter 2	Iter 3	Avg	Stdev
Batch: 4 trans: **base**	53	50	45	49.3	3.3
Batch: 4 trans: **grayscale**	60	46	55	53.7	5.79
Batch: 4 trans: **horizontal flip**	51	46	46	47.7	2.36
Batch: 4 trans: **rotation**	59	49	55	**54.3**	**4.1**
Batch: 4 trans: **cutmix**	41	52	52	48.3	5.18
Batch: 16 trans: **base**	58	45	48	50.3	5.56
Batch: 16 trans: **grayscale**	61	44	43	49.3	8.26
Batch: 16 trans: **horizontal flip**	57	48	62	55.6	5.79
Batch: 16 trans: **rotation**	59	49	46	51.3	5.56
Batch: 16 trans: **cutmix**	41	54	35	43.3	7.93

Table 5: Experiment 2 on complex CNN.

Experiment 3 While training models we found out, that the batch size for this model has influence on both training time and accuracy. Models having batch size set to 4, were training longer (in minutes), but their results were faster (in epochs) better 14.

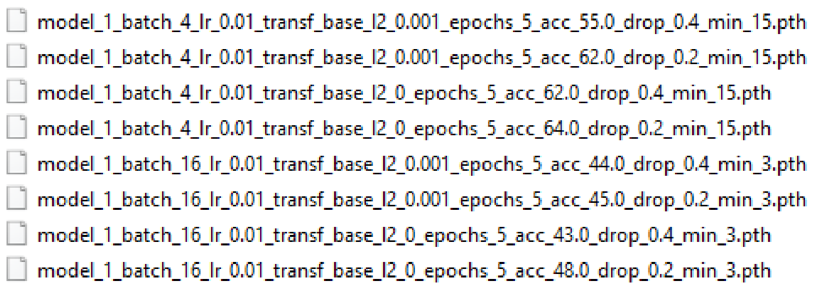

model_1_batch_4_lr_0.01_transf_base_l2_0.001_epochs_5_acc_55.0_drop_0.4_min_15.pth
model_1_batch_4_lr_0.01_transf_base_l2_0.001_epochs_5_acc_62.0_drop_0.2_min_15.pth
model_1_batch_4_lr_0.01_transf_base_l2_0_epochs_5_acc_62.0_drop_0.4_min_15.pth
model_1_batch_4_lr_0.01_transf_base_l2_0_epochs_5_acc_64.0_drop_0.2_min_15.pth
model_1_batch_16_lr_0.01_transf_base_l2_0.001_epochs_5_acc_44.0_drop_0.4_min_3.pth
model_1_batch_16_lr_0.01_transf_base_l2_0.001_epochs_5_acc_45.0_drop_0.2_min_3.pth
model_1_batch_16_lr_0.01_transf_base_l2_0_epochs_5_acc_43.0_drop_0.4_min_3.pth
model_1_batch_16_lr_0.01_transf_base_l2_0_epochs_5_acc_48.0_drop_0.2_min_3.pth

Figure 14: Influence of batch size on training time and accuracy.

5.2.3 Not pre-trained ResNet18

In tables 6 7 8 9 10 we can see the results of the training we have carried out. The column name and description indicates which changing hyperparameter the table refers to.

In the case of experiments involving the change of batch size, we can see that a batch size of 16 appears to be the most optimal, while one of 4 did not perform very well. In particular, it is worth noting that a learning rate that was too high prevented network learning. In the case of dropout, we can see that the results are similar. In the case of various data augmentation techniques, a special mention must be made of random horizontal flip, which clearly improved the performance of our networks. Perhaps this is because it is the only technique that leaves the image in its true state (as random rotation results in a cropped image). Regularisation, on the other hand, does not seem to change the performance of the nets significantly.

batch size	accuracy
4	0.709
8	0.754
16	0.738
32	0.732

Table 6: Experiment 1 of not pre-trained ResNet18 results - batch size.

learning rate	accuracy
0.001	0.733
0.003	0.748
0.01	0.770
0.05	0.100
0.1	0.100

Table 7: Experiment 1 of not pre-trained ResNet18 results - learning rate.

dropout	accuracy
0	0.748
0.2	0.746
0.4	0.753
0.6	0.760
0.8	0.742

Table 8: Experiment 1 of not pre-trained ResNet18 results - dropout.

transformation	accuracy
base	0.748
cutmix	0.727
gray_scale	0.720
random_horizontal_flip	0.765
random_rotation	0.728

Table 9: Experiment 1 of not pre-trained ResNet18 results - transform.

L2 lambda	accuracy
0	0.748
0.0001	0.753
0.001	0.754
0.01	0.170

Table 10: Experiment 1 of not pre-trained ResNet18 results - L2 lambda.

5.2.4 Pre-trained ResNet18

In the case of pre-trained resnet, we can draw similar conclusions regarding the influence of hyperparameters, which we can see in the tables 11 12 13 14 15. The greatest difference can be seen in the case of the grayscale transformation, which significantly worsens the effectiveness against the baseline. In addition, too much regularization caused a significant decrease in network quality.

batch size	accuracy
4	0.816
8	0.825
16	0.504
32	0.793

Table 11: Experiment 1 of pre-trained ResNet18 results - batch size.

learning rate	accuracy
0.001	0.824
0.003	0.816
0.01	0.614
0.05	0.582
0.1	0.100

Table 12: Experiment 1 of pre-trained ResNet18 results - learning rate.

dropout	accuracy
0	0.816
0.2	0.814
0.4	0.811
0.6	0.799
0.8	0.808

Table 13: Experiment 1 of pre-trained ResNet18 results - dropout.

transformation	accuracy
base	0.816
cutmix	0.817
gray_scale	0.771
random_horizontal_flip	0.826
random_rotation	0.808

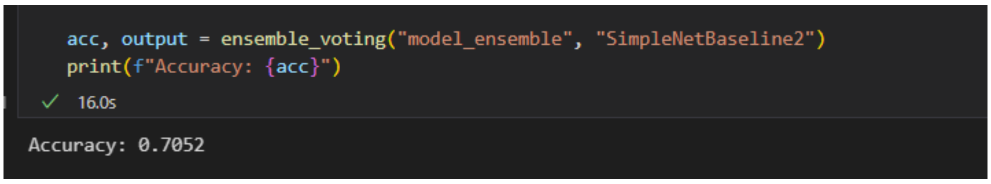
Table 14: Experiment 1 of pre-trained ResNet18 results - transform.

L2 lambda	accuracy
0	0.816
0.0001	0.807
0.001	0.748
0.01	0.180

Table 15: Experiment 1 of pre-trained ResNet18 results - L2 lambda.

5.2.5 Ensemble

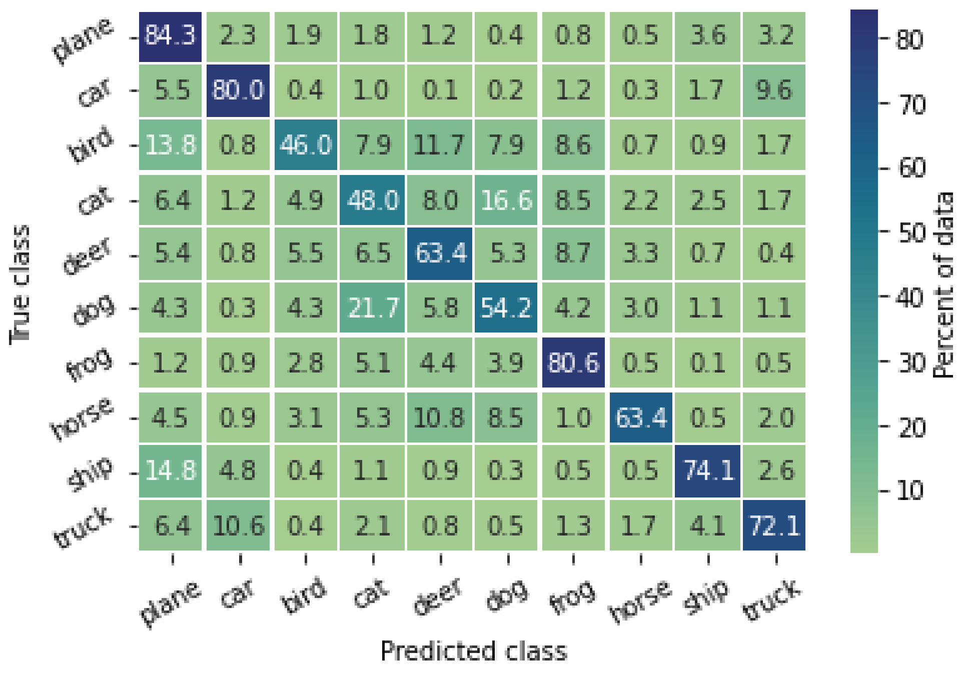
To check if ensemble of the model has better performance than single models, we run ensemble two times. Runs had similar hyperparameters, the only difference was momentum value. Both times the result of the ensemble model was higher than accuracy of single models. One example is shown on 15. Accuracy of single models was between 62 and 66, and for the ensemble it is over 70. On figure 16 we provide heatmap for one of the models. It is clearly visible, that there are classes like plane, car or frog, that are relatively easy for model to learn. But there are also some classes like cat and dog, that are often mixed.



```
acc, output = ensemble_voting("model_ensemble", "SimpleNetBaseline2")
print(f"Accuracy: {acc}")
```

```
Accuracy: 0.7052
```

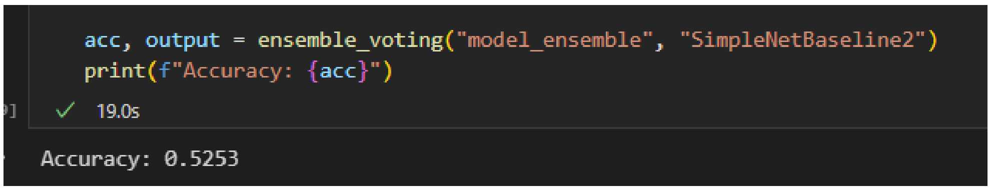
Figure 15: Accuracy of ensemble.



	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	84.3	2.3	1.9	1.8	1.2	0.4	0.8	0.5	3.6	3.2
car	5.5	80.0	0.4	1.0	0.1	0.2	1.2	0.3	1.7	9.6
bird	13.8	0.8	46.0	7.9	11.7	7.9	8.6	0.7	0.9	1.7
cat	6.4	1.2	4.9	48.0	8.0	16.6	8.5	2.2	2.5	1.7
deer	5.4	0.8	5.5	6.5	63.4	5.3	8.7	3.3	0.7	0.4
dog	4.3	0.3	4.3	21.7	5.8	54.2	4.2	3.0	1.1	1.1
frog	1.2	0.9	2.8	5.1	4.4	3.9	80.6	0.5	0.1	0.5
horse	4.5	0.9	3.1	5.3	10.8	8.5	1.0	63.4	0.5	2.0
ship	14.8	4.8	0.4	1.1	0.9	0.3	0.5	0.5	74.1	2.6
truck	6.4	10.6	0.4	2.1	0.8	0.5	1.3	1.7	4.1	72.1

Figure 16: Heatmap for one of the models used for the ensemble.

Above ensemble is made from models with a momentum 0.9. We also tried conducting one with simple CNNs, trained with the same architecture except from momentum, set to 0. In the second case, models were having worse accuracy (between 47 and 50) and the result of the voting was only over 52, but still higher than the average of the single model's results 17.



```
acc, output = ensemble_voting("model_ensemble", "SimpleNetBaseline2")
print(f"Accuracy: {acc}")
✓ 19.0s

Accuracy: 0.5253
```

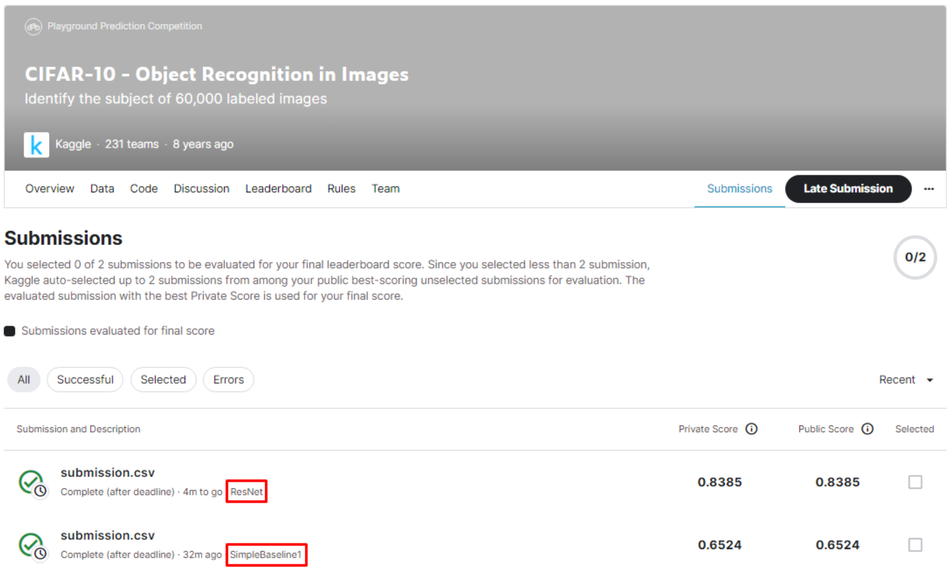
Figure 17: Accuracy of worster enseble.

5.2.6 Kaggle competition

In order to give credibility to our solutions, we decided to upload the predicted values for the test set for one best-performing network each from the SimpleBaseline and ResNet18 classes to the Kaggle competition page. The hyperparameters we selected were as follows

- SimpleBaseline - batch size 8, learning rate 0.001, random horizontal flip, lambda l2 0.001, dropout 0
- ResNet18 - batch size 16, learning rate 0.003, random horizontal flip, lambda l2 0.001, dropout 0

We can see the results presented in Figure 18 are very close to those achieved on the test data loaded from torchvision.


Playground Prediction Competition
CIFAR-10 - Object Recognition in Images
Identify the subject of 60,000 labeled images
Kaggle · 231 teams · 8 years ago
Overview
Data
Code
Discussion
Leaderboard
Rules
Team
Submissions
Late Submission
Submissions
0/2
All
Successful
Selected
Errors
Recent

You selected 0 of 2 submissions to be evaluated for your final leaderboard score. Since you selected less than 2 submission, Kaggle auto-selected up to 2 submissions from among your public best-scoring unselected submissions for evaluation. The evaluated submission with the best Private Score is used for your final score.

■ Submissions evaluated for final score

Submission and Description	Private Score	Public Score	Selected
submission.csv Complete (after deadline) · 4m to go ResNet	0.8385	0.8385	☐
submission.csv Complete (after deadline) · 32m ago SimpleBaseline1	0.6524	0.6524	☐

Figure 18: Kaggle competition results

6 Conclusions

During work on this project, we tried to find, how changes in hyperparameters influence the model's accuracy. For batch size it was clearly visible, that it can influence time of the training and also accuracy. For learning rate, it also happened, that models were learning slower when learning rate was too small. L2 helped us when trying to train models, that do not overfit. The use of dropout contributes to preventing overfitting of the neural network. However, when training the ResNet, we could not observe a significant difference in results for different values. Additionally, we found that momentum can have a big influence on the model's accuracy.
When it comes to augmentation techniques, we see that random flipping horizontally significantly improved ResNet's performance. In addition, we also have the conclusion that training with the CutMix technique is more difficult than the others and requires additional implementation.

Future work It would be worth considering training neural networks using several data augmentation techniques simultaneously. This would allow an even wider extension of the training set, which would certainly result in the quality of the final solution. Another idea could be to extend the training time during grid search experiments. It could then turn out that some other set of hyperparameters turned out to be better.

References

[1] Krizhevsky A. "Learning Multiple Layers of Features from Tiny Images", 2009

[2] Sangdoo Y. et al "CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features", 2019

[3] Kaiming He et. al "Deep Residual Learning for Image Recognition", 2016