WARSAW UNIVERSITY OF TECHNOLOGY
FACULTY OF MATHEMATICS AND INFORMATION SCIENCE

Project I Report

Deep Learning

Students:
Karina Tiurina (335943)
Mateusz Zacharecki (313549)

Supervisor:
mgr inz. Maciej Żelaszczyk

Warsaw 2024

Contents

1 Research problem

The topic of the project is Image classification with convolutional neural networks. We are considering CINIC-10 dataset from https://www.kaggle.com/datasets/mengcius/cinic10/data. CINIC-10 is an augmented extension of CIFAR-10. It contains the images from CIFAR-10 and a selection of ImageNet database images. It is split into three equal subsets - train, validation, and test - each of which contains 90,000 images.

The aim of the project is to:

- test and compare different network architecture, including convolutional neural networks
- Investigate influence of the following hyper-parameter change on obtained results:
  - at least 2 hyper-parameters related to training process
  - at least 2 hyper-parameters related to regularization
- investigate influence of at least X data augmentation techniques from the following groups:
  - standard operations (where X=3)
  - more advanced data augmentation techniques like mixup, cutmix, cutout (where X=1)
- consider application of ensemble (hard/soft voting, stacking).

2 Application instruction

Source code has the following structure.

- cnn
  - test.ipynb
  - train_history
    - cnn<test number>
      - <n>_accuracy.png
      - <n>_confusion_matrix.png
      - <n>_history.csv
      - <n>_loss.png
      - accuracy.csv
      - Boxplot.png
  - augmentation.png
- vgg16
  - test.ipynb
  - train_history
    - vgg<test_number>
      - ...
- voting
  - test.ipynb
  - train_history
    - ...
- main.ipynb

To train the model and reproduce the results presented in this report, please, run /cnn/test.ipynb, /vgg16/test.ipynb and /voting/test.ipynb

‘main.ipynb’ in the root folder contains plots and aggregated results of the training.

3 Theoretical introduction

Convolutional Neural Networks (CNNs) are highly important in modern artificial intelligence, especially in image recognition and related tasks. They are inspired by the human visual cortex and rely on mathematical convolution operations.

CNNs consist of the following layers:

1. Convolutional - apply learnable filters to input data, detecting features like edges or textures;

2. Pooling - reduce spatial dimensions, aiding generalization;

3. Fully connected layers – hidden and output layers.

Visual Geometry Group 16 (VGG16) is a deep convolutional neural network architecture known for its simplicity and effectiveness in image classification tasks. Developed by the Visual Geometry Group at the University of Oxford, VGG16 is characterized by its deep stack of convolutional layers, with 16 weight layers including 13 convolutional layers and 3 fully connected layers.

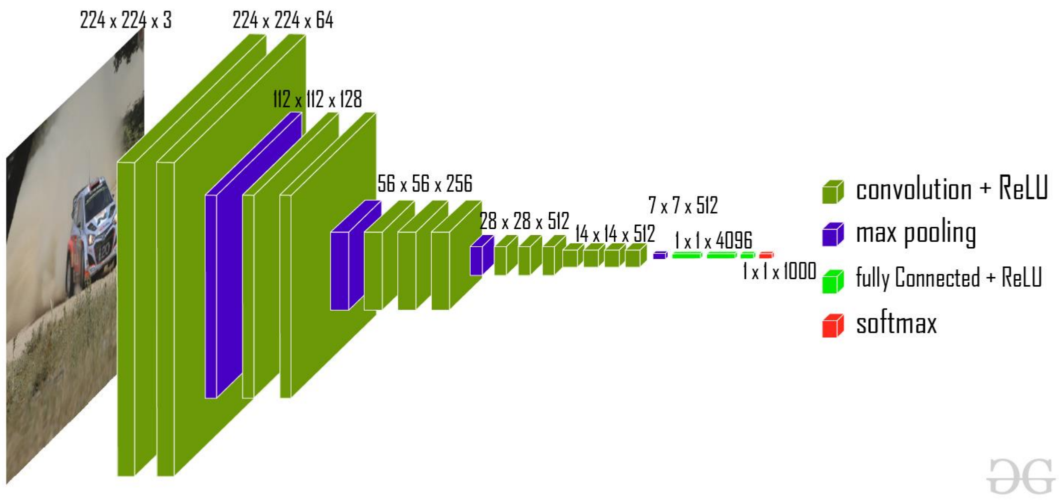


Figure 3.1 VGG16 architecture

VGG16 follows a straightforward design philosophy, using small 3x3 convolutional filters with a stride of 1 and max-pooling layers with a 2x2 filter size. Despite its simplicity compared to more recent architectures, VGG16 has achieved remarkable performance on benchmark datasets like ImageNet, demonstrating its robustness and versatility. Its straightforward structure makes it a popular choice for transfer learning and as a baseline model for various computer vision tasks.

Additionally, voting ensembles in classification allows to combine multiple classifiers to make predictions on a given dataset. Each individual classifier in the ensemble typically represents a different learning algorithm or a variation of a single algorithm, offering diverse perspectives on the data. Through various voting schemes such as majority voting, weighted voting, or averaging, the ensemble aggregates the predictions of its constituent classifiers to produce a final classification decision. This approach often leads to improved performance compared to individual classifiers, as it leverages the wisdom of crowds and mitigates the weaknesses of individual models.

4 Conducted experiments

4.1 Convolutional NN

There were 10 different tests conducted using CNN and keras library. Source code and results of the training are in /cnn folder.

1. CNN1 – single convolutional layer
2. CNN2 – single convolutional layer with normalization
3. CNN3 – single convolutional layer with dropout
4. CNN4 – 4 convolutional layers with (3, 3) filter
5. CNN5 – 4 convolutional layers with (2, 2) filter
6. CNN6 – 6 convolutional layers with Adamax optimizer
7. CNN7 – 6 convolutional layers with Nadam optimizer
8. CNN8 – 6 convolutional layers with Adam optimizer
9. CNN9 – learning rate tuning of Adamax optimizer
10. CNN10 – CNN6 with learning rate 0.005 and augmentation

Each test, except CNN9, was conducted 5 times to discover the accuracy distribution. Results are provided in the boxplots on figure 4.1.

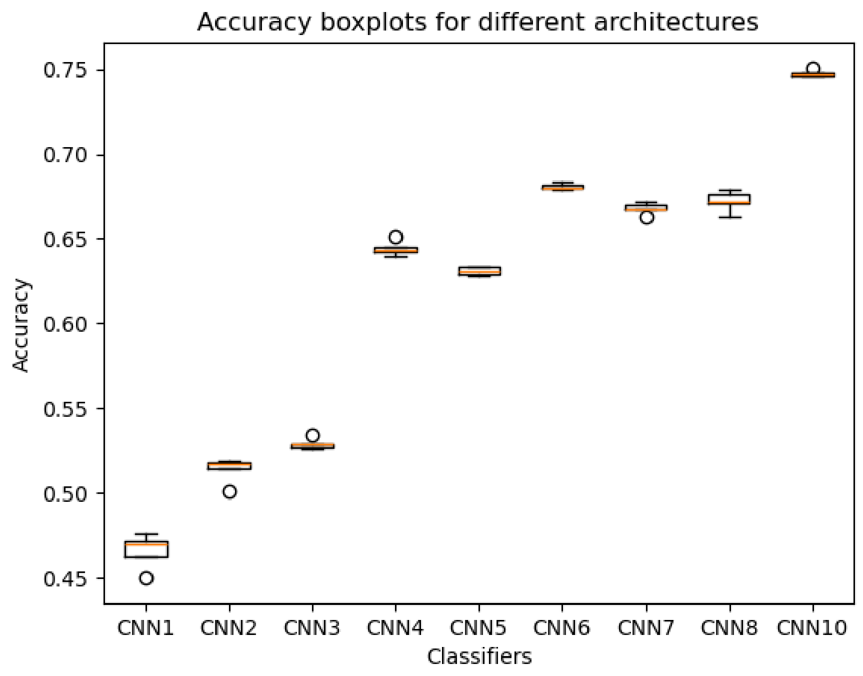
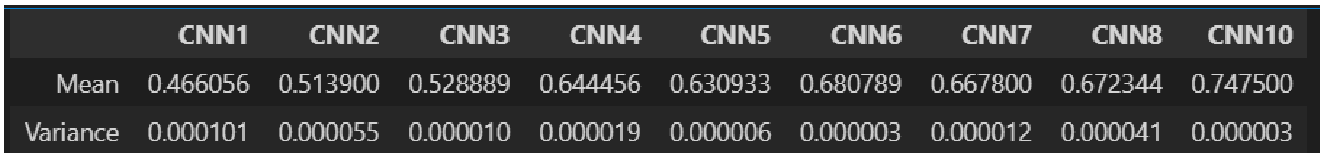


Figure 4.1 Boxplots on CNN tests

Table 4.1 below contains mean and variance of accuracy for each tested architecture.



	CNN1	CNN2	CNN3	CNN4	CNN5	CNN6	CNN7	CNN8	CNN10
Mean	0.466056	0.513900	0.528889	0.644456	0.630933	0.680789	0.667800	0.672344	0.747500
Variance	0.000101	0.000055	0.000010	0.000019	0.000006	0.000003	0.000012	0.000041	0.000003

Table 4.1 CNN tests summary

Best achieved accuracy is 74.06% of CNN10: 6 convolutional layers with Adamax optimizer, learning rate 0.005 and augmentation. Figures 4.2, 4.3 and 4.4 contain plots of accuracy and loss per epoch and a confusion matrix.

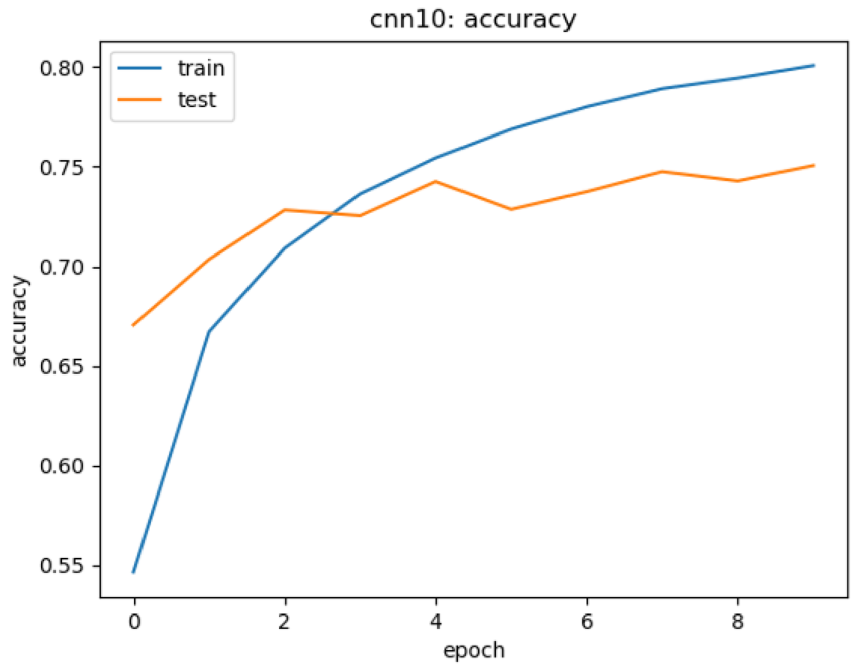


Figure 4.2 Train and test accuracy per epoch of CNN10

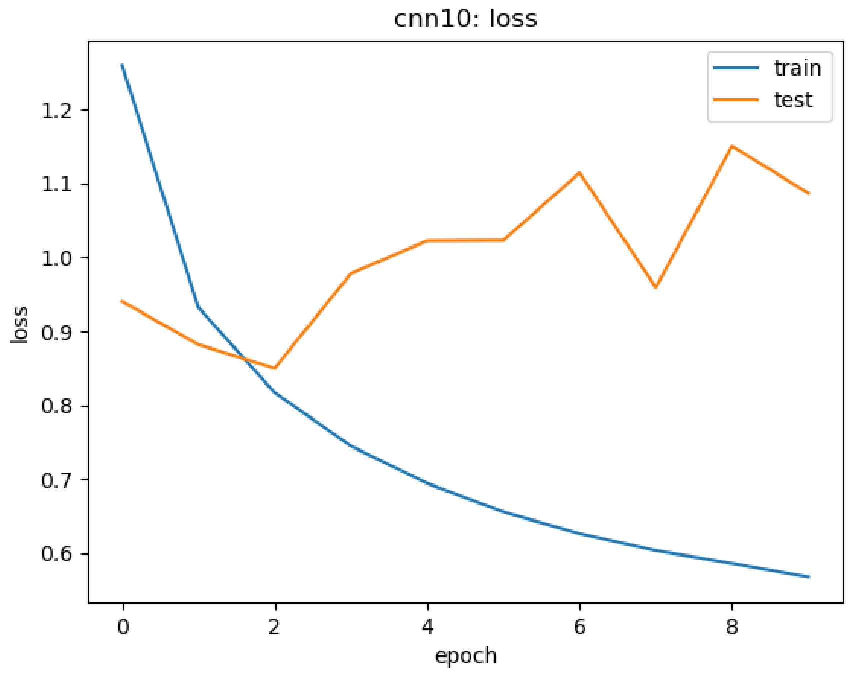
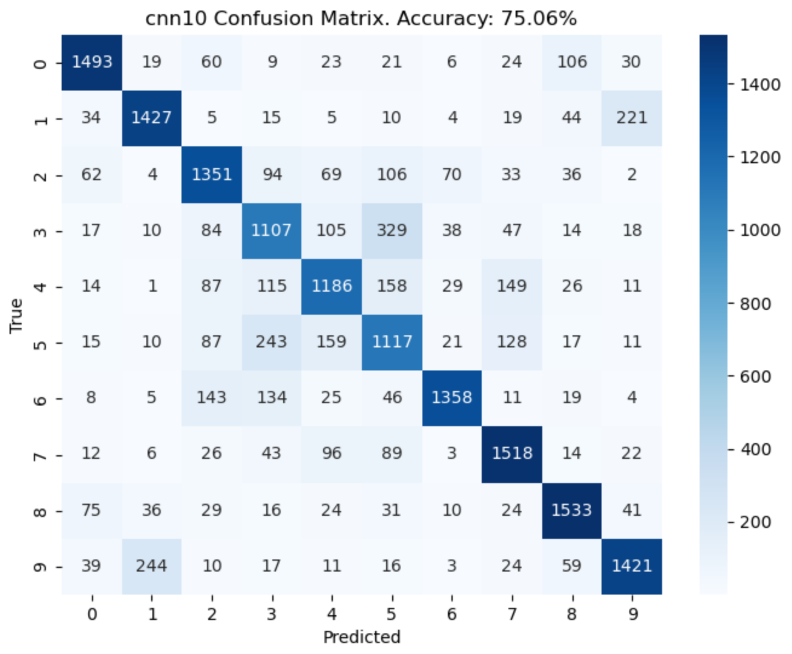


Figure 4.3 Train and test loss per epoch of CNN10



	0	1	2	3	4	5	6	7	8	9
0	1493	19	60	9	23	21	6	24	106	30
1	34	1427	5	15	5	10	4	19	44	221
2	62	4	1351	94	69	106	70	33	36	2
3	17	10	84	1107	105	329	38	47	14	18
4	14	1	87	115	1186	158	29	149	26	11
5	15	10	87	243	159	1117	21	128	17	11
6	8	5	143	134	25	46	1358	11	19	4
7	12	6	26	43	96	89	3	1518	14	22
8	75	36	29	16	24	31	10	24	1533	41
9	39	244	10	17	11	16	3	24	59	1421

Figure 4.4 Confusion matrix of CNN10 with accuracy 75.06%

Based on the loss, the model still has an overfitting problem. However, the accuracy overall is reasonable.

To augment images for CNN5, each image was modified 5 times in the following ways.

1. Flip
2. Rotation
3. Translation
4. Random brightness change
5. Random contrast change

The results of the augmentation can be visualized on figure 4.5.

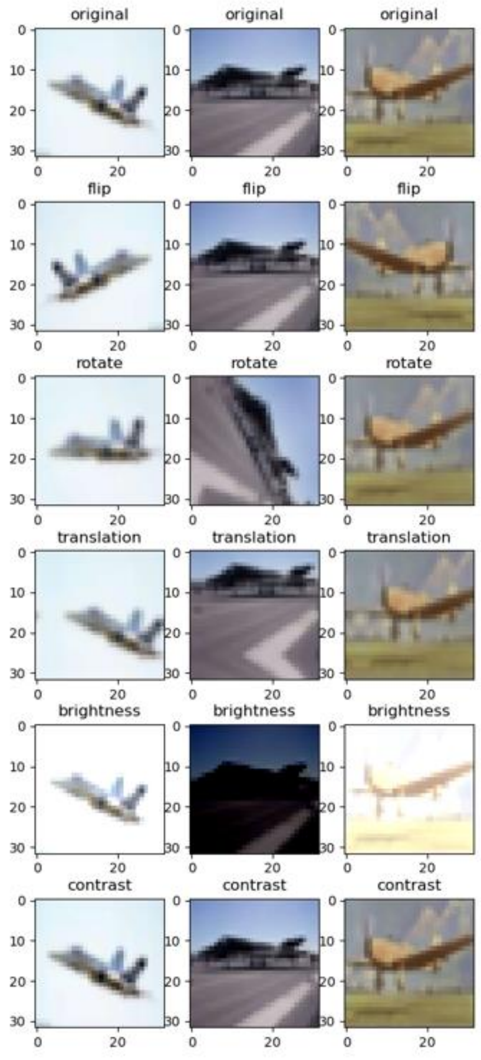


Figure 4.5 Image augmentation

4.2 Visual Geometry Group (VGG) transfer learning

For VGG16 pre-trained model there were 12 different tests conducted. Source code and results of the training are in /vgg folder.

1. VGG1 - VGG16 pre-trained model with extra layers
2. VGG2 - VGG1 with dropout layers, rescaling factor and 5 epochs
3. VGG3 - VGG1 with normalization, rescaling factor and 5 epochs
4. VGG4 - VGG3 with dropout layers
5. VGG5 - VGG3 with adamax optimizer and 25 epochs (without repeating)
6. VGG6 - VGG3 with sgd optimizer and 10 epochs (without repeating)
7. VGG7 - VGG5 with 10 epochs (without repeating) and learning rate tuning
8. VGG8 - VGG6 with learning rate tuning
9. VGG9 - VGG5 with 10 epochs (without repeating), learning rate = 0.001 and batch size tuning
10. VGG10 - VGG9 with batch size = 128 and extra layers
11. VGG11 - VGG9 with batch size = 128 and extra layers
12. VGG14 - VGG9 with batch size = 64 and data augmentation

The first 4 tests were performed 5 times, however, because of the evaluation time, the rest of tests were performed only once. Results for these four tests are presented in the following boxplots.

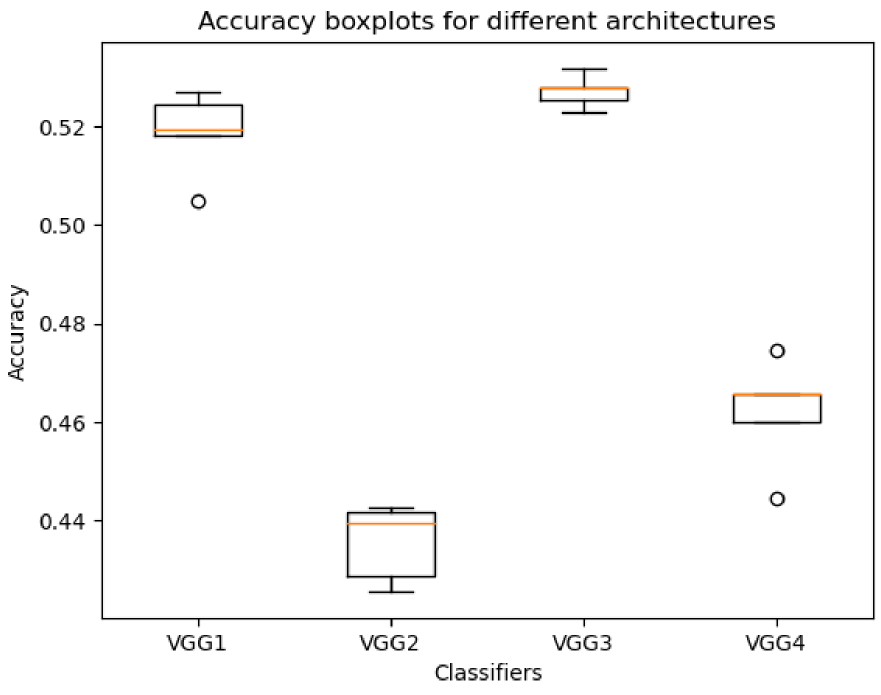


Figure 4.6 Boxplots on VGG tests

Table 4.2 below contains mean and variance of accuracy for these four tested architectures.

	VGG1	VGG2	VGG3	VGG4
Mean	0.518689	0.435589	0.527100	0.461967
Variance	0.000073	0.000062	0.000011	0.000124

Table 4.2 Means and variances for VGG tests

Table 4.3 below contains accuracy for parameters tested only once, except VGG7, VGG8, VGG9 architectures.

	VGG5	VGG6	VGG10	VGG11	VGG14
Accuracy	0.532111	0.532167	0.529389	0.519056	0.529556

Table 4.3 Accuracy for VGG tests

Accuracies for VGG7, VGG8 and VGG9 tests are presented in the following three graphs, depending on tuned parameters.

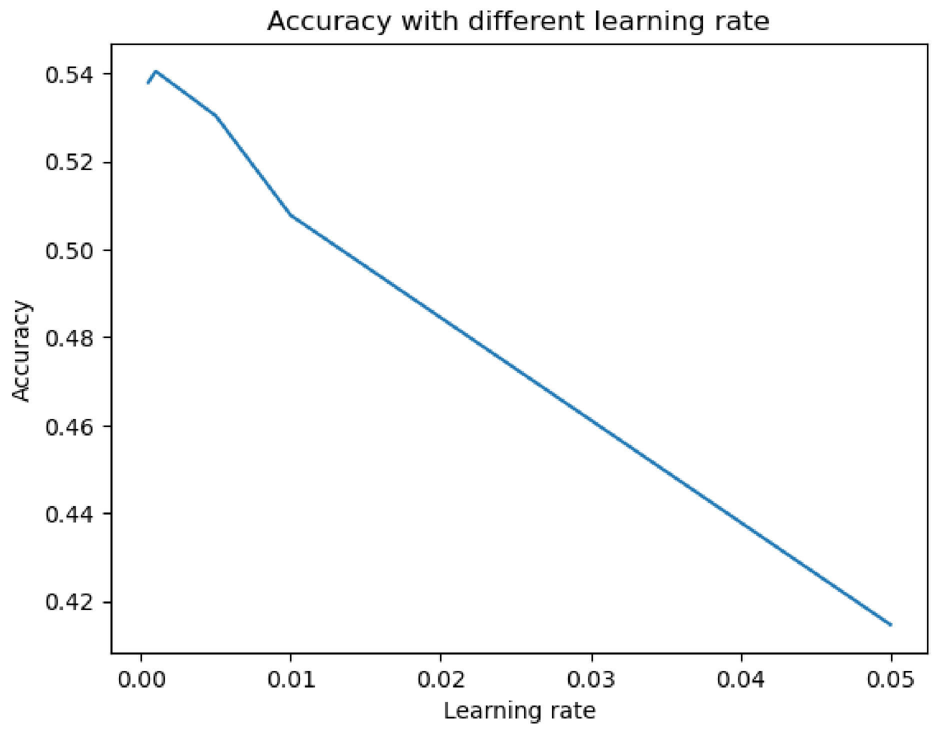


Figure 4.7 Accuracy score for VGG7 depending on tested learning rate

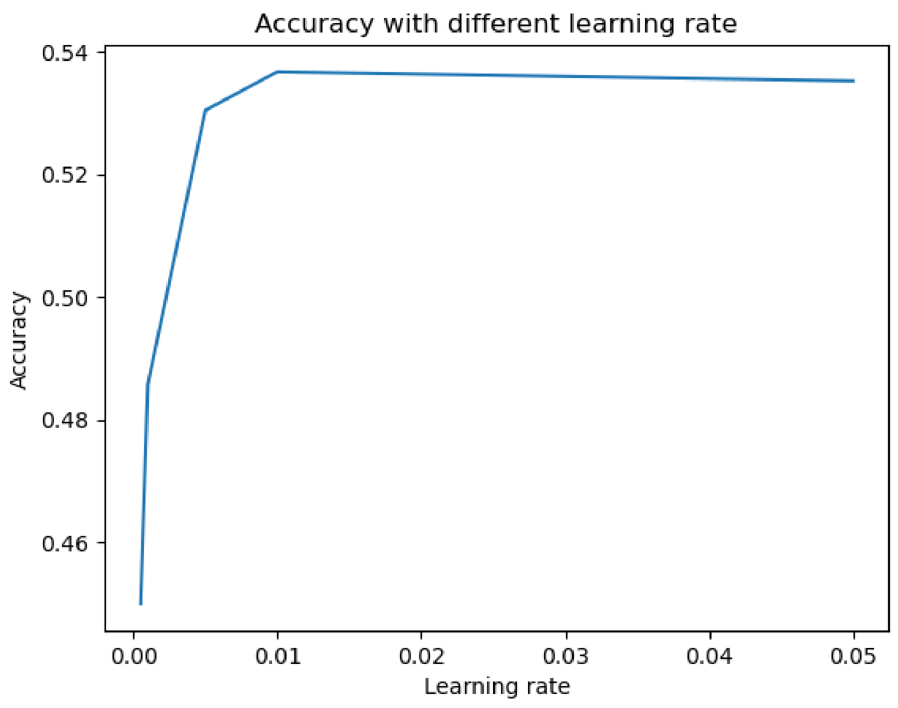


Figure 4.8 Accuracy score for VGG8 depending on tested learning rate

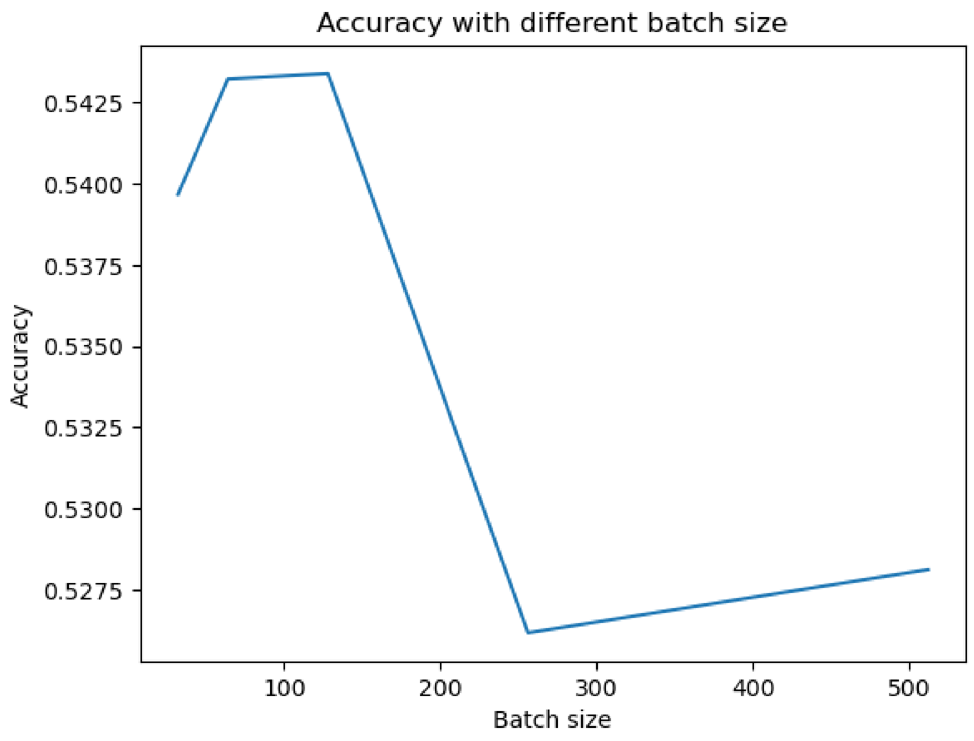


Figure 4.9 Accuracy score for VGG9 depending on tested learning rate

The best trained VGG model turned out to be VGG16 model with normalization layer and denses with respectively 50, 20 and 10 units, with adamax optimizer and learning rate = 0.001, trained with batch size = 128 (VGG9 with batch size = 128). This model achieved accuracy 54.34%. The following graphs present respectively accuracy per epoch, loss per epoch and confusion matrix for this model.

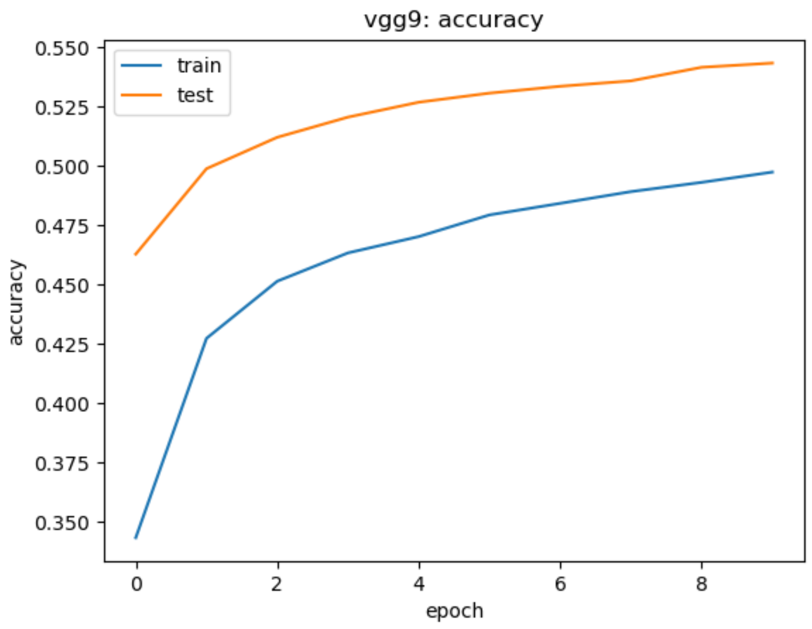


Figure 4.10 Training and test accuracy per epoch for VGG9 (128)

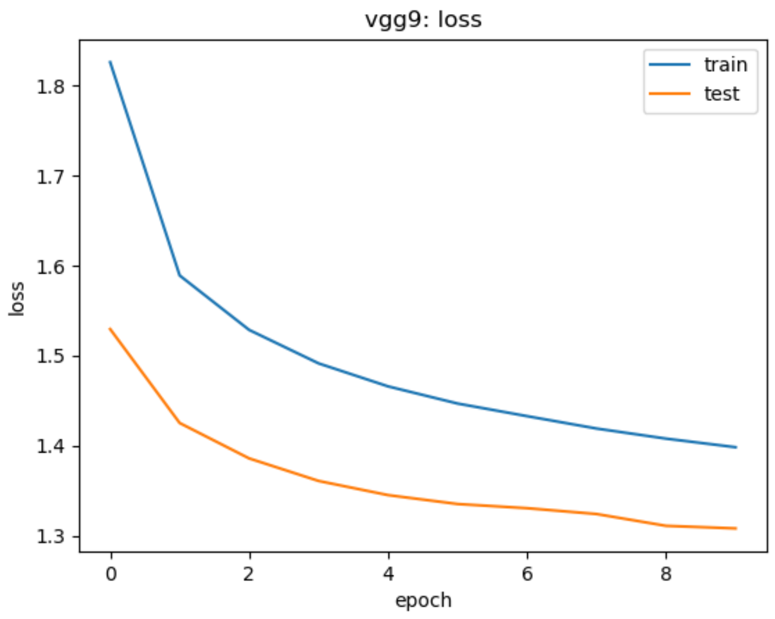
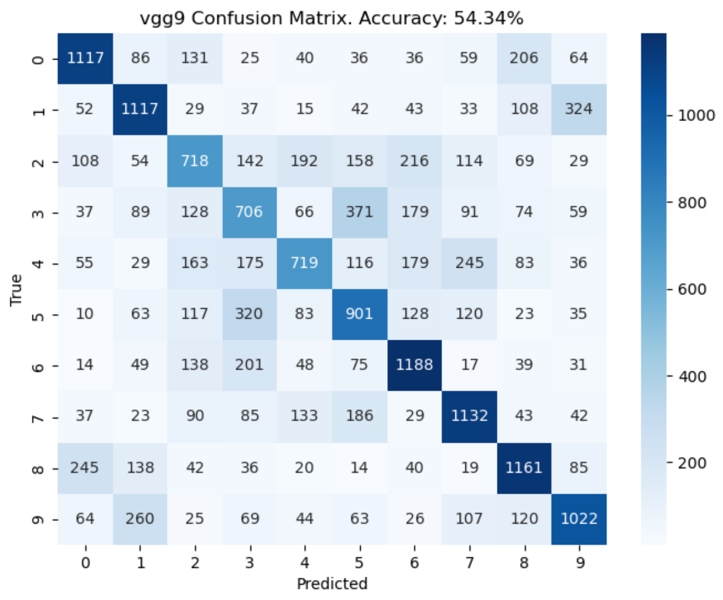


Figure 4.11 Training and test loss per epoch for VGG9 (128)



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1117	86	131	25	40	36	36	59	206	64
1	52	1117	29	37	15	42	43	33	108	324
2	108	54	718	142	192	158	216	114	69	29
3	37	89	128	706	66	371	179	91	74	59
4	55	29	163	175	719	116	179	245	83	36
5	10	63	117	320	83	901	128	120	23	35
6	14	49	138	201	48	75	1188	17	39	31
7	37	23	90	85	133	186	29	1132	43	42
8	245	138	42	36	20	14	40	19	1161	85
9	64	260	25	69	44	63	26	107	120	1022

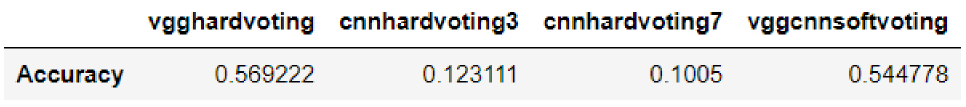
Figure 4.12 Confusion matrix for VGG9 (128)

4.3 Voting ensemble

For hard voting there were 3 different tests conducted. Source code and results of the training are in /voting folder.

1. VGGHardVoting – Hard voting for VGG8 (learning rate = 0.01), VGG9 (batch size = 128), VGG14
2. CNNHardVoting3 – Hard voting for CNN6 (first attempt), CNN7 (fifth attempt), CNN10 (fifth attempt)
3. CNNHardVoting7 – Hard voting for CNN6 (first attempt), CNN7 (fifth attempt), CNN10 (all attempts)
4. VGGCNNSoftVoting – Hard voting for VGG9 (batch size = 128) and CNN10 (fifth attempt)

The following table presents accuracies for considered voting ensembles.



	vgghardvoting	cnnhardvoting3	cnnhardvoting7	vggcnnsoftvoting
Accuracy	0.569222	0.123111	0.1005	0.544778

Table 4.4 Accuracies for considered voting ensembles

Voting works good for VGG models, meanwhile results for CNN architecture are significantly worse.

5 Summary

The more convolution layers are added to the CNN, the better accuracy it is able to achieve. However, there is a high chance to face with an overfitting problem. This can be addressed by using a more advanced architectures like VGG16 and ResNet or by usage of ensemble algorithms.

Unfortunately, our VGG16 experiments did not produce an expected accuracy above 80%. The reasons for that might be: wrong activation functions, insufficient parameter tuning or incorrect pre-trained model selection.

The following research could be further conducted to improve transfer learning accuracy.

1. Test different activation functions;
2. Unfreeze first layers of the VGG network to train with the model;
3. Change the pre-trained model to ResNet or EfficientNet;
4. Check misclassified output to understand failures.

Unfortunately, applying voting ensembles on considered architectures does not improve accuracies. The expected results are not achieved and we get even worse scores than for single models.

References

[1] CINIC-10 challenge https://www.kaggle.com/code/eclaircat/cinic-10-challenge
[2] Pytorch transfer learning tutorial
https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

[3] VGG-16 transfer learning
https://www.kaggle.com/code/carloalbertobarbano/vgg16-transfer-learning-pytorch

[4] VGG-16 | CNN model https://www.geeksforgeeks.org/vgg-16-cnn-model/