Project I - Convolutional neural networks
Project Report

Kinga Frańczak, 313335
Grzegorz Zakrzewski, 313555

Contents

1 Description of the research problem **1**

2 Instruction of the application **1**

3 Theoretical introduction **2**

4 Description of the conducted experiments **3**

5 Results **4**

6 Conclusions **9**

1 Description of the research problem

The topic of this project is image classification with convolutional neural networks. The project was prepared using the CINIC-10 dataset, which was obtained from this source [3]. The CINIC-10 dataset contains 180,000 images, which are splitted into equally sized training and test subsets. Each sample is a 32×32 color pixel image, saved as a PNG file. Images come from one of ten classes: automobile, airplane, bird, cat, deer, dog, frog, horse, ship, and truck. Sample images from each class are shown in Figure 1. Images in the training subset are placed in ten directories, whose names correspond to specific classes. Images in the test subset are unlabeled and and therefore were not used.

As previously mentioned, the aim of this project is to prepare a model to classify images from the CINIC-10 dataset. The project instructions require testing and comparing different network architectures, of which at least one has to be a convolutional neural network. Additionally, it is required to investigate the influence of two hyper-parameters related to the training process, two hyper-parameters related to regularization, and four different augmentation techniques. The utilized hyper-parameters and augmentation techniques are introduced in Section 3, and the experiments are described in Section 4.

2 Instruction of the application

The project solution was developed using the Python programming language and Jupyter Notebook. All the code needed to conduct experiments is contained in the included *experiments.ipynb* file. The work was facilitated by the `keras` [4] library with TensorFlow backend. Other libraries such as `pandas`, `matplotlib`, and `seaborn` were utilized to process and visualize results. Version requirements for the mentioned libraries are listed at the top of the *experiments.ipynb* file. Additionally, the random number generator was initialized with a constant seed to ensure reproducibility of the experiments.

Figure 1: Sample images from each class in the CINIC-10 dataset.

3 Theoretical introduction

Due to our lack of prior experience with convolutional neural networks and a slight delay in the availability of lecture materials, our understanding of how to construct CNNs is primarily based on internet resources of questionable value. Specifically, the main sources of information during this project were a blog post [1], one of the examples available in the Keras package documentation [5], and some notebooks accessible in the *Code* tab on the CINIC-10 dataset Kaggle website [3].

Put simply, a convolutional neural network is a type of neural network that incorporates at least one layer performing convolution. Convolution involves a kernel, which slides along the input for the layer. The kernel is smaller in size than the input data matrix. During convolution, the kernel values are multiplied by the values in the input data matrix, resulting in the output of the convolutional layer. This layer facilitates the extraction of features and properties from the input data while mitigating the need for a large number of neurons and computational resources. In CNNs, pooling layers are often utilized alongside convolutional layers. The primary objective of pooling layers is to downsample the input and speed up the learning process.

In conducted experiments, described in Section 4, three network architectures were tested, comprising one, two, or three convolutional layers. These architectures are illustrated in the diagram presented in Figure 2. The number of dense layers remains consistent across all architectures.

Undoubtedly, the most critical hyper-parameters during the neural network learning process are the optimization method for an objective function (also known as an optimizer) and the step size closely associated with this optimizer, often referred to as the learning rate. Leveraging the diverse options provided by the Keras package's optimizers module, four optimizers were tested and compared: SGD, RMSProp, Adam, and Adadelta. Additionally, five values of the learning rate hyper-parameter were compared across the best-performing optimizer.

In addition to hyperparameters associated with the training process, we also investigated the impact of two hyperparameters related to the regularization process. We chose two popular techniques: dropout and early stopping. The *Dropout* layer in keras randomly nullifies the influence of selected units, aiding in the prevention of overfitting. Meanwhile, the *EarlyStopping* callback halts training after a specified number of epochs with no improvement.

As the final step, four augmentation techniques were investigated. Randomly selected samples from the training dataset were flipped, rotated, shifted, or mixed up and then used as additional input data. The

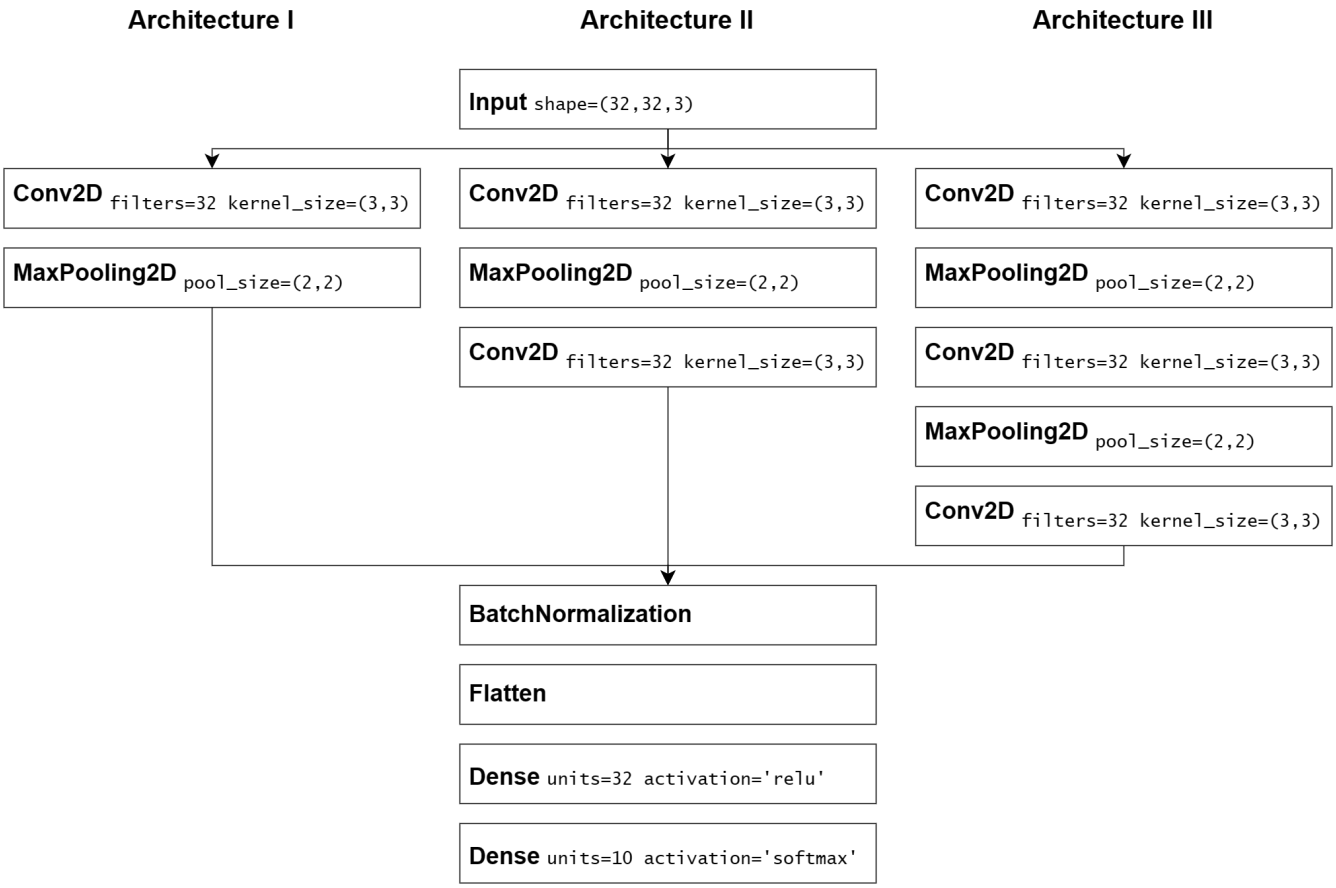


Figure 2: Three neural network architectures (keras nomenclature) used in the experiments.

RandomFlip, *RandomRotation*, and *RandomTranslation* layers are already included in the Keras library, but the mix-up augmentation technique was implemented by modifying the following example [5]. Sample effect of augmentation techniques is presented in Figure 3.

4 Description of the conducted experiments

Beginning with some logistical details, the training subset was partitioned into two segments: 72,000 images for training (which accounts for 80% of the subset) and the remaining 18,000 images for validation. All metrics presented in this report pertain to the accuracy achieved on the validation dataset. The data was divided into batches of 32 images each. Each neural network model was trained using the `sparse_categorical_crossentropy` loss function and for 30 epochs. Furthermore, each model was trained three times with the same parameters. While it might have been preferable to repeat the experiments a larger number of times, the training process was exceedingly time-consuming.

There were three primary experiments conducted: the first concerning neural network architecture (Figure 2), the second investigating the influence of hyper-parameters, and the third focusing on augmentation techniques. The experiments were conducted iteratively, meaning that each subsequent experiment utilized the best setup achieved in the previous one. Details of the experiments are provided in Table 1.

During the experiments involving augmentation techniques, the training subset was expanded by 24,000 images obtained using the specific technique. All parameters not directly related to the experiment's topic remained at their default values.

Figure 3: Sample effect of augmentation techniques.

Experiment	Objective	Values
0	Architecture	1 conv. layer
		2 conv. layers
		3 conv. layers
1.1	Optimizer	SGD
		RMSProp
		Adam
		Adadelta
1.2	Learning rate	0.0001
		0.0005
		0.001
		0.005
		0.01
1.3	Dropout rate	0.2
		0.4
		0.6
1.4	Early stopping - patience	2
		4
		6
2	Augmentation technique	image flips
		image rotations
		image shifts
		mix-up

Table 1: Details of the experiments.

5 Results

The results of the experiments are presented in the form of tables and plots. Table 2 displays the mean (and standard deviation) of the best values of the loss function and accuracy achieved by models in each experimental setup. Similarly, the plots shown in Figure 4 depict the accuracy computed on the validation subset for each epoch. In the following paragraphs, each experiment is briefly discussed.

Experiment 0 (architecture) (Table 2a, Figure 4a) clearly indicates that the best architecture is the one with three convolutional layers. For models with one or two convolutional layers, the accuracy achieved on the training subset is very high and the accuracy on the validation set decreases with subsequent epochs. This is an obvious sign that simpler models are overfitted.

Experiment 1.1 (optimizer) (Table 2b, Figure 4b) does not have one definite winner. For an unknown reason, the Adadelta optimization method performs very poorly, but the other three methods — SGD, Adam, and RMSProp — obtain very similar results. Adam optimizer was selected for the following experiments due to its highest accuracy and lowest standard deviation.

Experiment 1.2 (learning rate) (Table 2c, Figure 4c) indicates that the learning rate hyper-parameter set to 0.01 or 0.005 is too high. Smaller values perform much better. Therefore, learning rate of 0.001 was retained for the subsequent experiments.

Experiment 1.3 (dropout rate) (Table 2d, Figure 4d) yields little surprise. All three compared values of the dropout rate allow the models to achieve similar results, with an accuracy of about 0.585 on the validation subset. However, there is one noteworthy observation: the accuracy on the training subset is lower for the models with regularization than for the models without it, such as those from Experiment 1.2 (learning rate). Furthermore, the higher the value of the dropout rate parameter, the lower the accuracy on the training subset. Nevertheless, the accuracy on the validation subset is higher than in the models without the regularization parameter. This indicates that the regularization is effective.

Experiment 1.4 (early stopping) (Table 2e, Figure 4e) did not have a good impact on the performance of the models. Perhaps the number of epochs was too small, or maybe the models are not very prone to overfitting. The only advantage of introducing the patience parameter was that it accelerated the learning process.

Experiment 2 (augmentation technique) (Table 2f, Figure 4f) yields puzzling results. For unknown reasons and contrary to intuition, the models' results are much worse than in the previous experiments. Additionally, the standard deviations of accuracy on both the training and testing subsets are the highest of all experiments. The only reasonable explanation is that the models are not capable of effectively handling augmented images. A potential solution to this problem could involve a more complex network architecture or a greater number of epochs.

After conducting all these experiments, the best model architecture appears as follows: three convolutional layers, Adam optimizer, a learning rate of 0.001, dropout rate of 0.2, and early stopping patience set to 6 epochs or longer. No augmentation technique should be used.

Architecture	Loss	Accuracy	Validation loss	Validation accuracy
1 conv layer	0.455 (0.031)	0.837 (0.010)	1.405 (0.024)	0.506 (0.007)
2 conv layers	0.438 (0.047)	0.842 (0.018)	1.328 (0.027)	0.538 (0.004)
3 conv layers	1.017 (0.007)	0.636 (0.003)	1.227 (0.006)	0.565 (0.004)

(a) Experiment 0 - architecture

Optimizer	Loss	Accuracy	Validation loss	Validation accuracy
adadelta	1.939 (0.004)	0.289 (0.006)	1.963 (0.004)	0.291 (0.005)
adam	0.904 (0.000)	0.674 (0.000)	1.214 (0.004)	0.572 (0.003)
rmsprop	0.986 (0.046)	0.646 (0.016)	1.217 (0.016)	0.571 (0.004)
sgd	1.025 (0.014)	0.634 (0.006)	1.226 (0.018)	0.567 (0.006)

(b) Experiment 1.1 - optimizer

Learning rate	Loss	Accuracy	Validation loss	Validation accuracy
0.0001	1.051 (0.018)	0.625 (0.007)	1.253 (0.011)	0.557 (0.005)
0.0005	0.952 (0.043)	0.658 (0.015)	1.215 (0.005)	0.574 (0.001)
0.001	0.970 (0.053)	0.652 (0.017)	1.221 (0.011)	0.568 (0.001)
0.005	1.135 (0.027)	0.592 (0.011)	1.300 (0.029)	0.542 (0.004)
0.01	1.960 (0.595)	0.247 (0.253)	2.088 (0.668)	0.242 (0.245)

(c) Experiment 1.2 - learning rate

Drop rate	Loss	Accuracy	Validation loss	Validation accuracy
0.2	1.058 (0.052)	0.619 (0.020)	1.144 (0.022)	0.589 (0.008)
0.4	1.121 (0.019)	0.597 (0.007)	1.149 (0.028)	0.585 (0.010)
0.6	1.192 (0.018)	0.572 (0.008)	1.159 (0.013)	0.582 (0.009)

(d) Experiment 1.3 - drop rate

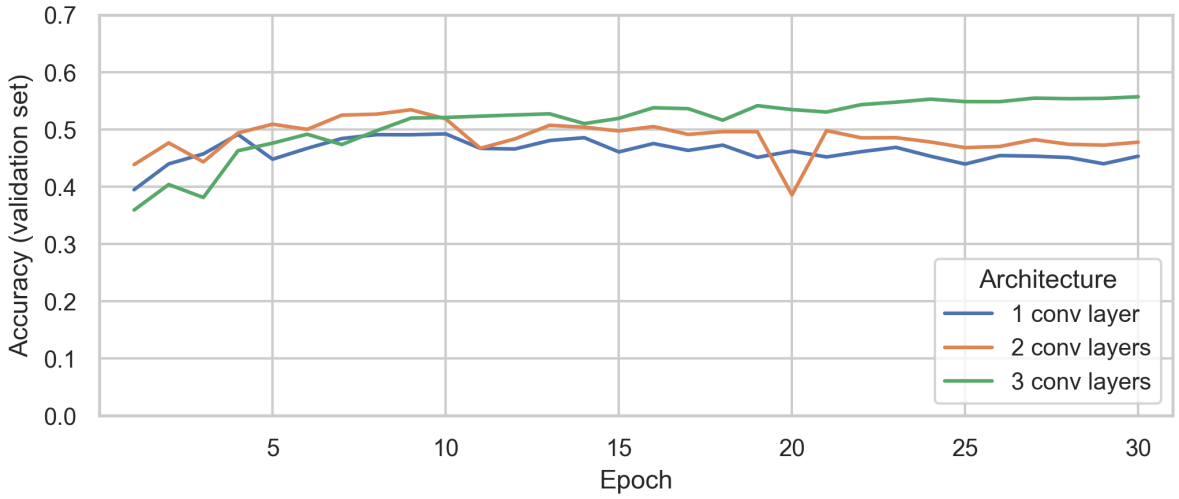
Patience	Loss	Accuracy	Validation loss	Validation accuracy
2	1.230 (0.098)	0.556 (0.037)	1.282 (0.093)	0.536 (0.040)
4	1.153 (0.008)	0.585 (0.002)	1.214 (0.020)	0.563 (0.008)
6	1.121 (0.011)	0.596 (0.005)	1.192 (0.032)	0.572 (0.012)

(e) Experiment 1.4 - early stopping

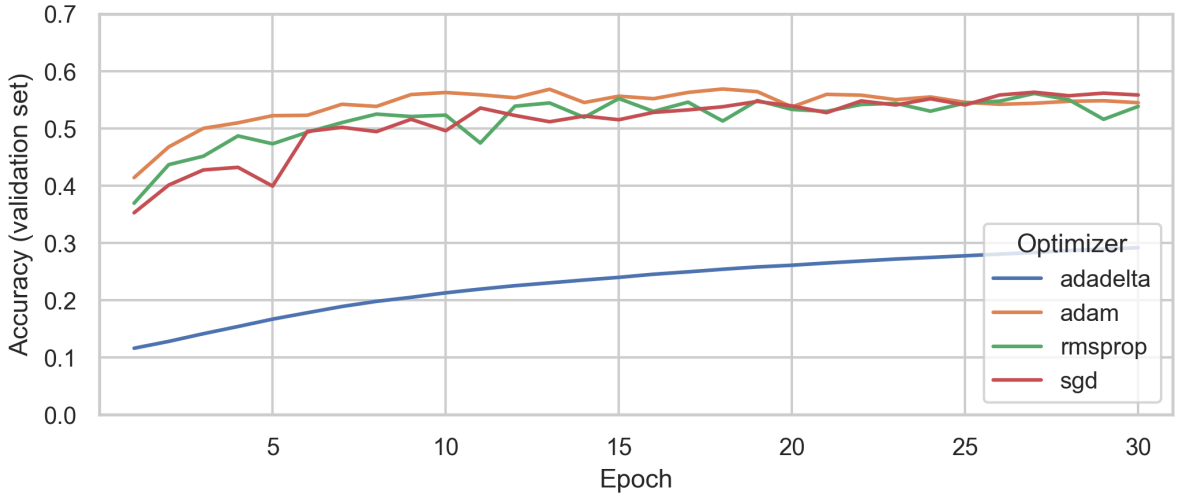
Augmentation	Loss	Accuracy	Validation loss	Validation accuracy
mix up	1.285 (0.047)	0.535 (0.017)	1.339 (0.056)	0.506 (0.026)
random flip	1.142 (0.077)	0.588 (0.030)	1.459 (0.458)	0.481 (0.160)
random rotation	1.296 (0.066)	0.531 (0.026)	1.496 (0.235)	0.451 (0.093)
random translation	1.305 (0.052)	0.526 (0.020)	1.504 (0.266)	0.448 (0.100)

(f) Experiment 2 - augmentation technique

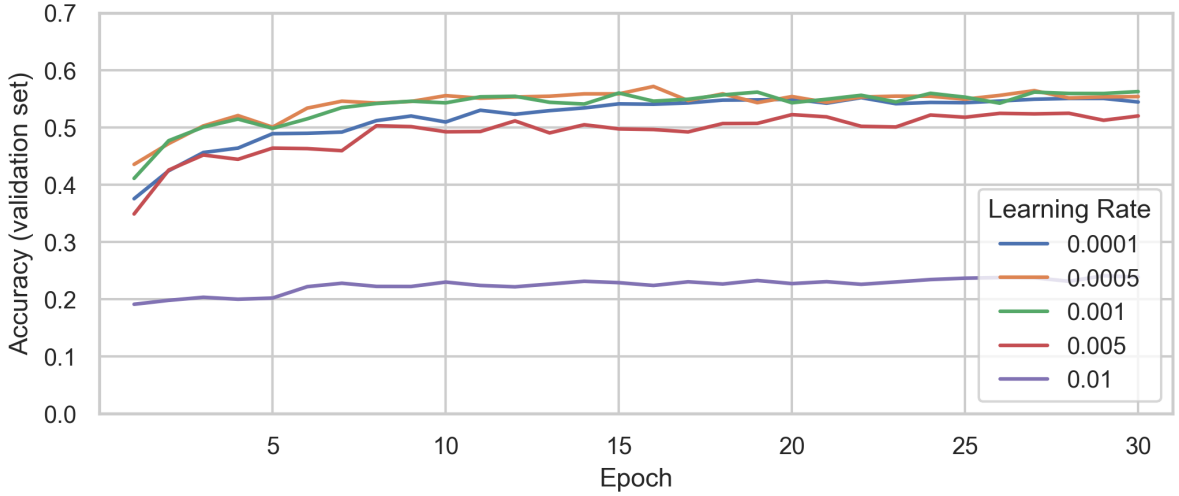
Table 2: Tables presenting the mean (and standard deviation) of the best values of the loss function and accuracy achieved by models in each experimental setup.



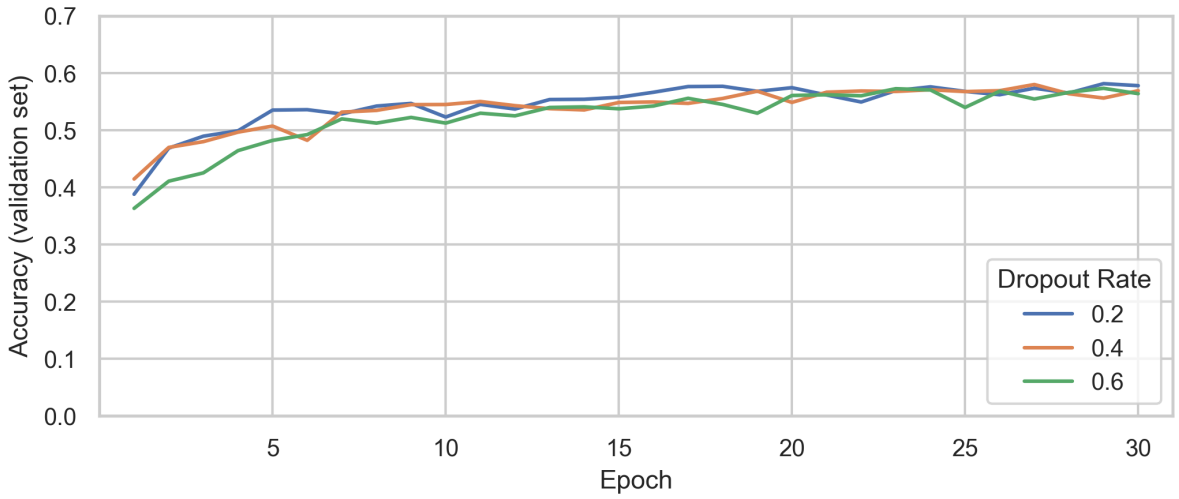
(a) Experiment 0 - architecture



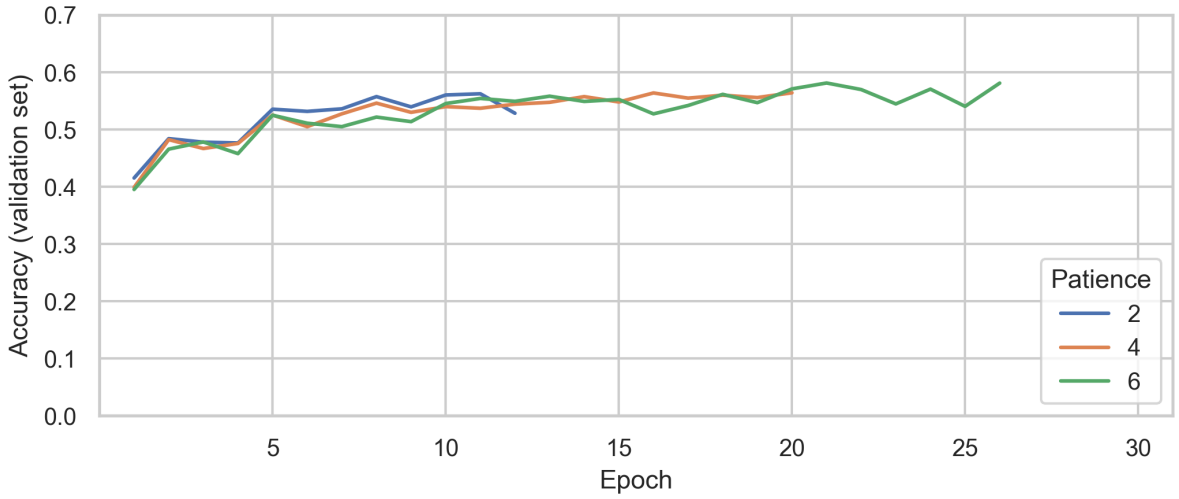
(b) Experiment 1.1 - optimizer



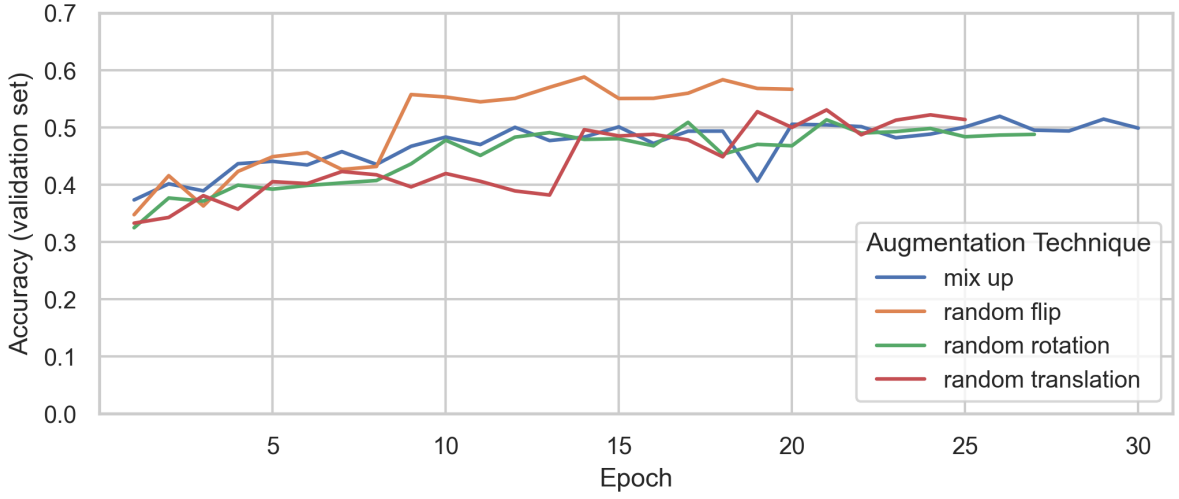
(c) Experiment 1.2 - learning rate



(d) Experiment 1.3 - drop rate



(e) Experiment 1.4 - early stopping



(f) Experiment 2 - augmentation technique

Figure 4: Plots presenting the accuracy computed on the validation subset for every epoch.

6 Conclusions

The primary goal of this project, which is to prepare a model to classify images from the CINIC-10 dataset, was successfully achieved. Experiments comparing network architectures, the influence of hyperparameters related to the training process and regularization, and data augmentation techniques were conducted. The results of these experiments enabled us to draw conclusions and select the best hyperparameter settings.

However, the accuracy achieved on the validation subset (around 0.58) was not as high as one would expect. According to a benchmark study [2], some models can achieve an accuracy rate of around 0.95. Further investigation should consider the neural network architecture, which may need to be more complex. It may be worth considering adding more dense layers after the convolutional layers. Additionally, augmentation techniques may prove to be more useful with increased complexity of the models.

Last but not least, we are solely responsible for starting this project too late. Due to long model training time and initial issues with the TensorFlow library, experiments were only repeated three times. Additionally, pretrained models were not tested as originally planned.

References

[1] A Beginner's Guide To Understanding Convolutional Neural Networks. https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/.

[2] CINIC-10 Benchmark. https://paperswithcode.com/sota/image-classification-on-cinic-10/.

[3] CINIC-10 dataset. https://www.kaggle.com/datasets/eclaircat/cinic-eclair-test/.

[4] keras. https://keras.io/.

[5] keras mix-up example. https://keras.io/examples/vision/mixup/.