Machine Learning Report

Silvio Klenk

30th April 2024

Abstract

This report explores the development of Convolutional Neural Network for image classification from the public dataset Muffins vs Chihuahuas. Specifically, several models and their performances are compared. Then, the influences of data augmentation and hyperparameter tuning are examined. In the last step, the risk of the best performing models is estimated by 5-fold cross-validation. [1]

[1] I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

Contents

List of Figures

List of Tables

1 Introduction

In this project, a set of Convolutional Neural Networks will be developed to test the task of binary image classification between muffins and chihuahuas. Convolutional Neural Networks (CNN) are a very powerful type of model to perform computer vision tasks. In the course of the project, several architectures will be presented and their performance will be evaluated and visualized using various techniques. More precisely, the first models are the baseline models in which the performance is compared and evaluated based on the depth of the individual architectures. In particular, the number of filters is increased. The problem of overfitting is then dealt with. Especially the influence of data augmentation and dropout layers was examined, more on this in the respective chapter. The last model is characterized by the fact that an improvement in performance was targeted through a hyperparameter tuing procedure. At the end of the project, the risk of the best performing model is estimated using a 5-fold cross validation.

2 Dataset and Preprocessing

The data set used for all analyses is a freely accessible data set from the Kaggle platform. It was inspired by the meme, 'Muffin or Chihuahua', in which people were shown pictures of Chihuahuas and muffins and these are seemingly difficult to differentiate. The data set was created based on these illusions. A binary image classification problem with 6017 images collected from google. It contains neither duplicates nor corrupted files. **Image transformation** was part of the initialisation of the project. Therefore the JPEG files have been converted into an RGB format with standard size of 128,128 pixels. In addition on that, for better model performances, the RGB values have been rescaled from a range of [0,255] to the range of [0,1]. Additional steps for preprocessing were to prepare the data for the machine learning pipeline, the images were first extracted from the original folders, labeled and stored in a combined Pandas dataframe. The dataframe was then shuffled, which benefits the learning process and helps the models to generalize. As a final step in data preparation, the dataframe was divided into test, training and validation datasets, following the conventional approach to machine learning model performance evaluation. The main reason for this division is to have a validation dataset on which the model has not been trained, so that hyperparameters can be optimized, configuration decisions can be made, and this without touching the test data set. Once the model has been finalized, the test dataset is used to evaluate performance. All this is done to prevent over- and underfitting. The percentages for the Training/Test/Validation datasets are 64%/20%/16% of the originally available amount of images.

3 Theoretical Constructs & empirical Methodologies

In order to provide a better understanding of the constructed models, the main features of Convolutional Neural Networks are discussed in detail below. These are typical approaches for computer vision tasks. In the first part, the characteristics of the individual layers are described.

3.1 Network Layers

The typical configuration of a convolutional neural network starts with the inclusion of an image as input. The subsequent architecture includes a sequential series of layers, convolutional layers and pooling layers (see figure 1), which are used for feature extraction. These initial layers apply filters to recognize patterns, edges and textures. Pooling layers are then applied to reduce the dimensionality while retaining the essential information. After arbitrary multiple layers are applied, the data is flattened into a vector and then passed through a fully-connected layer. This layer further interprets the features and makes predictions. The final output is a probability distribution over the classes, indicating the class to which the image most likely belongs. This hierarchical structure allows Convolutional Neural Network to efficiently handle complex image recognition tasks.

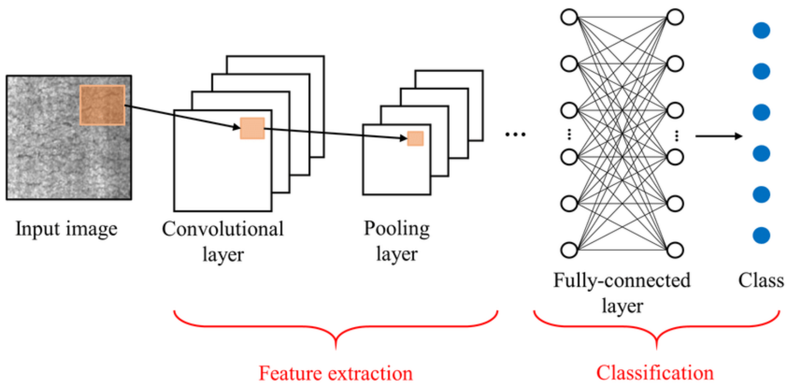


Figure 1: Typical architecture of a CNN [2]

3.1.1 Convolution Layer

One of the essential elements of Convolutional Neural Networks, which in this case is used specifically for the processing of image data. The Convolutional layers essentially apply Convolutional operations, which involve a series of adaptive filters on the input data. These filters can be thought of as small squares of pixels that move over the input data, trying to extract features such as edges, shapes and textures. By defining the individual parameters in the convolutional layer, you define how these operations should take place.

- **Number of filters:** specifies the number of filters used, determining the number of feature maps the layer will produce. More filters can capture a wider variety of features but also increase computational complexity.
- **Filter size:** This Argument represents the size of the filter, meaning the pixels that are seen at one time. Where smaller filters are able to detect finer details, lager filter would be used to capture more global features of the input data. During the development of the project the size was set to 3x3 pixels.
- **Activation Function:** The activation function defines whether the neuron of the network gets activated or not, in this project always *ReLu* for the convolution layers, determines the output of the respective nodes. It stands for Rectified Linear Unit and is mathematically defined as $f(x) = \max(0, x)$. *ReLu* is chosen mainly for the reason of inducing a non-linearity to the model without affecting the input fields of the convolutional layers and thus complex real world scenarios with complex patterns can be learned efficiently.
- **Padding:** By setting the parameter for padding to *same*, it ensures that the output feature maps have the same size as the input. This is done by adding zeros around the edges of the input image to allow the layers to include the border pixels for feature detection.
- **Input shape:** defines the shape of the input data that the layer expects, which, for colored images, typically includes two spatial dimensions (height and width of 128x128 pixels) and a depth for the color channels (3 for RGB).

[2]Image Source link CNN

3.1.2 Max Pooling Layer

Similar to the previously described convolutional layer, the max pooling layer moves a 2x2 filter across the input data, received from the previous layer. This serves to reduce the spatial size of the feature maps. Within the selected quadrant the maximum value gets chosen. The reduction serves not only through multiple max pooling layers to reduces the models complexity by diminishing the number of learnable parameters, but also allows the network to become more robust against noise and variance in the input data. Moreover, another effect of the max pooling layer is, that the extraction of the most prominent features has no more influence on the exact location on the feature map. This is mostly beneficial for recognizing patterns regardless of their position.

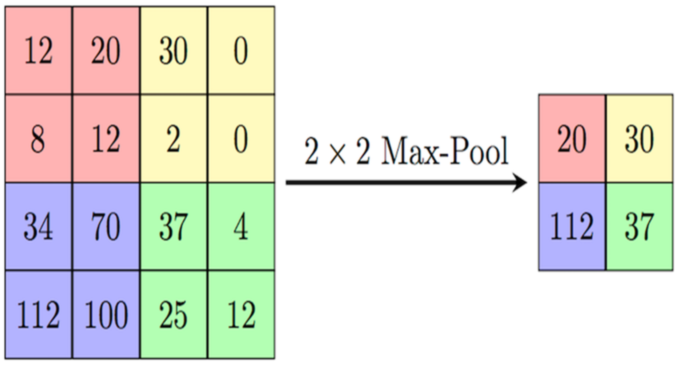


Figure 2: Max pooling with 2x2 filter [3]

3.1.3 Dropout Layer

In a neuronal network the Dropout layer serves as a regulatory mechanism to mitigate overfitting. This is done by randomly selecting a random subset of neurons to deactivate them during the training phase. The layer ensures that the model does not develop any dependencies between the neurons, which can have a potentially negative impact on the generalization performance. The rate set to this layer determines the fraction of the input units to be dropped, thereby increasing the robustness of the model, enhancing the predictive performance on the unseen data. Dropout is a form of regularization that effectively forces the network to learn more general patterns rather than memorizing the training data.

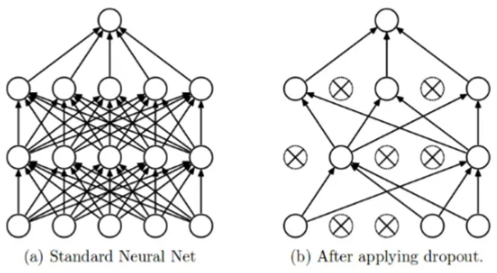


Figure 3: Applied Dropout [4]

[3]Image Source Link max pooling
[4]Image Source Link Dropout

3.1.4 Flatten & Dense Layer

Following a sequence of convolutional, max pooling and/or dropout layers for feature extraction, a flatten and dense layer normally results. The **Flatten layer** is used to transform the multi-dimensional output from the previous layers into a one-dimensional array. This transformation is essential for moving from the spatial feature extraction phases to the fully connected phases of the network, where pattern classification occurs.

After this process, one **dense layers** and one **output layer** is set. These layers consists of a preset amount of neurons, in which each is connected to all outputs from the previous layer. The first dense layer consists of 128 neurons, each connected to all the outputs from before and again set with the *ReLu* activation function. The second output layer, the overall final layer, consists of a single neuron, representing the likelihood of the input data belonging to one of the two classes. Here the *sigmoid* activation function is used. This function transforms the neuron's output into a value between 0 and 1. This enables a probabilistic interpretation of the result. In summary, the final layer condenses all the learned features of the complex network into a single prediction.

3.1.5 Data Augmentation layer

A common practice in image classification is Data Augmentation. This is a process where the images are randomly transformed before shown to the Neuornal Network model. This allows the model to see different aspects, to enable better generalization.

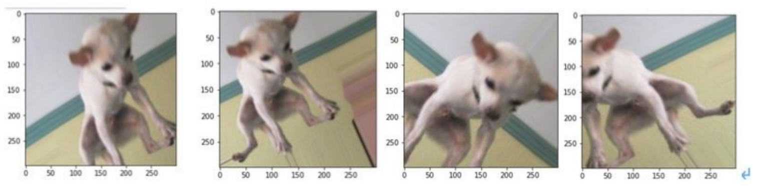


Figure 4: Data Augmentation example[5]

Several data augmentation techniques have been implemented to expand the training dataset and improve the robustness and the generalization ability of the models. This is essential to achieve higher accuracy, particularly when the training data set is limited:

1. **Random horizontal flip:** This augmentation technique involves flipping the images horizontally. Using this method helps the model to learn from a larger number of orientations and prevents it from over-fitting towards the specific orientation of the objects in the training data.

2. **Random rotation:** This augmentation technique rotates the images, allowing rotations of up to 10%. As a result, the model is exposed to a range of orientations and is trained to recognize objects regardless of minor changes in their directions. Thus, it is a simple but effective way to simulate the different viewing angles the model might face in real-life situations.

3. **Random zoom:** Applying a random zoom level of up to 10% helps to train your model to recognize and understand objects even when their relative scale changes. What makes this technique particularly valuable is for images where the objects are different sizes or are at different distances from the camera.

4. **Random Translation:** Here the images get shifted either vertically or horizontally by up to 10%. Nearest is specified as fill mode, which results in filling empty pixels arising after tranlation, with the nearest pixel values.

[5]Image Source Link data augmentation

3.2 Model Compilation

In the next part, the final pre-training step for the neural network model preparation is described. In the compilation phase, optimizer, loss function and metrics are selected, each with an important function for model performance and evaluation.

3.2.1 Optimizer

Adam represents an actualization of the RMSProp optimizer and merges it with the main feature of the momentum method. This optimization algorithm uses running averages with exponential forgetting of both the gradients and the second moments of the gradients. [6]

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

For each parameter, the Adam optimizer adjusts the learning rate individually based on estimates of the first and second moments of the gradients. The algorithm in practice will keep two moving averages per parameter; one for the gradients m_t and one for the square of the gradients v_t.

3.2.2 Loss Function

Binary cross-entropy is a loss function that is used for binary classification problems. Its used for measuring the performance of models, were the output represents a probability value between 0 and 1:

$$L = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

where y_i represents the true label for each point, p_i the predicted probability of the observation and N the number of observations.

3.2.3 Metrics

For all the following models, the *accuracy* metrics is used. It computes the ratio of the number of correct prediction over the total number of input samples :

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions made}}$$

4 Models description and Experimental Results

A detailed description on the models constructer is provided in the following section. The main focus is on the sequential composition, the type of compilation and the evaluation.

4.1 Model 1 & 2: Baseline Models

The baseline models are mainly characterized by the use of relatively simple model architectures to create a general understanding of convolutional neural networks.

4.1.1 Models Architecture

The first two model architectures are composed of different blocks of Convalution and Max pooling layers. The sequence of these blocks is followed by the two previously mentioned dense layers with 128 and 1 neurons.

Model 1: consists of two blocks of convolution and Max pooling layers, with 32 and 64 filters computed:

[6]Source Link Optimizer

```python
def model_1_specif():
  model_1 = Sequential([
        # First layer with 32 filters
        Conv2D(32, (3, 3), activation='relu', padding='same',
        ↪  input_shape=(128, 128, 3)),
        MaxPool2D((2, 2)),
        # Second layer with 64 filters
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        MaxPool2D((2, 2)),
        # Flatten layer
        Flatten(),
        # Fully connected layers
        Dense(128, activation='relu'),
        Dense(1, activation='sigmoid')])
  return model_1
```

Model 2: consists of three blocks of convolution and Max pooling layers, with 32, 64 and 128 filters computed:

```python
def model_2_specif():
  model_2 = Sequential([
        # First layer with 32 filters
        Conv2D(32, (3, 3), activation='relu', padding='same',
        ↪  input_shape=(128, 128, 3)),
        MaxPool2D((2, 2)),
        # Second layer with 64 filters
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        MaxPool2D((2, 2)),
        # Third layer with 128 filters
        Conv2D(128, (3,3), activation='relu', padding='same'),
        MaxPool2D((2,2)),
        # Flatten layer
        Flatten(),
        # Fully connected layers
        Dense(128, activation='relu'),
        Dense(1, activation='sigmoid')])
  return model_2

```

Starting from this, the idea was to create baseline models to observe the general training behaviour over the epochs. By comparing the behaviour of the models training and loss, the understanding of building deeper models becomes clear.

4.1.2 Models compilation and Fitting

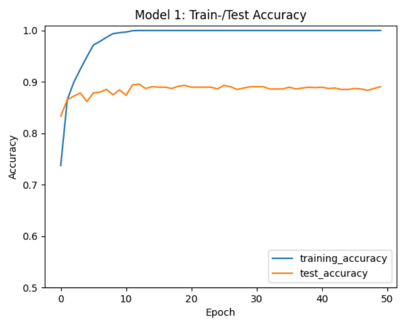
The specifications used to compile and fit the models 1 & 2 are the following:

1. **Number of epochs:** 50
2. **Loss Function:** Binary Crossentropy
3. **Optimizer:** Adam
4. **Metrics:** Accuracy

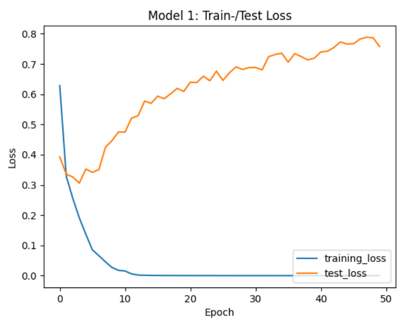
Both Models are trained on the same training dataset and evaluated on the same test dataset.

4.1.3 Model Evaluation

The plots in figure 5 and figure 6 show the obtained improvement of training and test accuracy as well as loss over all epochs from the first and second model. The overfitting problem can be clearly seen. While the training accuracy continuously improves, the test accuracy remains relatively constant at the same level.

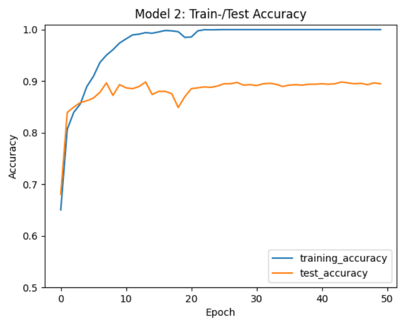


(a) Accuracy of Model 1

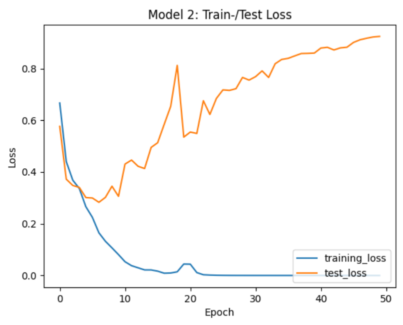


(b) Loss of Model 1

Figure 5: Model 1 results



(a) Accuracy of Model 2



(b) Loss of Model 2

Figure 6: Model 2 results

Proceeding from the previous figures, the following table presents the improvements of the deeper models, where adding blocks of Convolution and max pooling layers has been transpired.

Model	Test Accuracy	Test Loss
Model 1	0.887	0.782
Model 2	0.896	0.918

Table 1: Model Performance Metrics

As a result of this model performance, an attempt was therefore made to reduce the variance error and thus prevent overfitting.

4.2 Model 3: Reducing overfitting

An attempt with the earlier mentioned techniques was made, to prevent overfitting. Therefore, Model 3 was constructed and evaluated.

4.2.1 Models Architecture

Alongside the convolutional and max pooling layers that have already been implemented, now the problem of overfitting is being investigated. As well established methods, the additional layers of data augmentation and dropout are added to the network. Both are proven practices both as a regularization technique and as a method to enrich the dataset to improve the generalization abilities of a neural network. [7] [8] The code snippet below shows the Model 3 architecture:

```python
def model_3_specif():
  model_3 = Sequential([
        # Define data augmentation layers
        RandomFlip("horizontal", input_shape=(128, 128, 3)),
        RandomRotation(0.1),
        RandomZoom(0.1),
        RandomTranslation(height_factor=0.1, width_factor=0.1,
        ↪  fill_mode='nearest'),
        # First layer with 32 filters
        Conv2D(32, (3, 3), activation='relu', padding='same',
        ↪  input_shape=(128, 128, 3)),
        MaxPool2D((2, 2)),
        Dropout(0.5),
        # Second layer with 64 filters
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        MaxPool2D((2, 2)),
        Dropout(0.5),
        # Third layer with 128 filters
        Conv2D(128, (3,3), activation='relu', padding='same'),
        MaxPool2D((2,2)),
        Dropout(0.5),
        # Fourth layer with 256 filters
        Conv2D(256, (3,3), activation='relu', padding='same'),
        MaxPool2D((2,2)),
        Dropout(0.5),
        # Flatten layer
        Flatten(),
        Dropout(0.3),
        # Fully connected layers
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')])
  return model_3
```

As seen in the snippet the sequence of layers starts of with the randomization techniques for flipping, rotating, zooming and translation of the input images. After that, the previously seen boxes of convolutional and max pooling layers are added with the dropout layer. Here the specifications are set to a dropout of 50%. The flatten and fully connected layer have both been added with another dropout layer with the rate set to 30%. For a deeper model, now the blocks consist of 32, 64, 128 and 256 filters. The final dense layer remains the same.

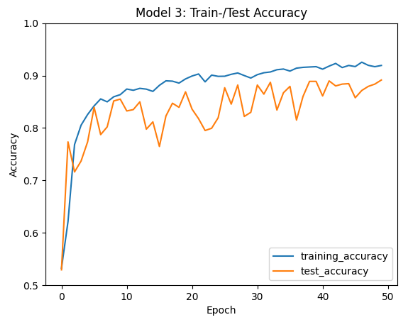
4.2.2 Models compilation and Fitting

Likewise to Model 1 & 2, this model was compiled with the same parameters.

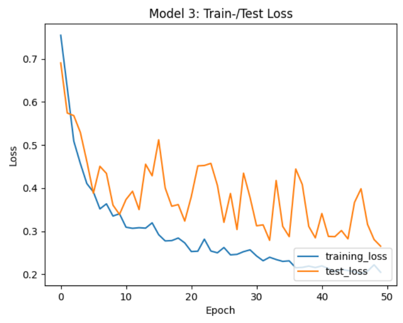
[7]Source Dropout Layers
[8]Source Data Augmentation Layers

4.2.3 Model Evaluation

The following figure 7 and table 2 present the Training and Test Accuracy/ Loss. As can be clearly seen, the methods used to prevent overfitting have resulted in significant improvements. Especially during the first couple epochs, training and validation accuracy and loss are moving closer together. Therefore, the model can be described as becoming better at generalization.



(a) Accuracy of Model 3



(b) Loss of Model 3

Figure 7: Model 3 results

Model	Test Accuracy	Test Loss
Model 3	0.891	0.269

Table 2: Model Performance Metrics

4.3 Model: Hyperparameter tuning

As a last model to be fitted, the powerful technique of hyperparameter tuning was performed. With this, an attempt of trying to find the optimal parameters to aim for improving the models performance has been done. Due to limitations in computational power not all the possible values can be explored.

4.3.1 Models Architecture

In order to find the parameters' values, the Hyperparameter Tuner was used to set a range for each parameter that would be the most meaningful. The following list shows ranges for the respective parameters:

1. **Rotation Factor**: Range 0 to 0.1 with steps of 0.01
2. **Zoom Range**: Range 0.01 to 0.1 with steps of 0.01
3. **Translation Factor (Height and Width)**: Range 0.01 to 0.1 with steps of 0.01
4. **Dropout Rate for Convolutional Layers**: Range 0.3 to 0.5 with steps of 0.1
5. **Number of Units in Convolutional Layers**:
    - Convolutional Layer 1: Range 32 to 64 with steps of 16
    - Convolutional Layer 2: Range 64 to 128 with steps of 32
    - Convolutional Layer 3: Range 128 to 256 with steps of 64
    - Convolutional Layer 4: Range 256 to 512 with steps of 128

6. **Dropout Rate for Dense Layers**: Range 0.3 to 0.5 with steps of 0.1

7. **Number of Units in Dense Layers**:
   - Dense Layer 1: Range 64 to 256 with steps of 64
   - Dense Layer 2: Range 64 to 256 with steps of 64

8. **Activation Functions**:
   - Convolutional Layers: ReLU
   - Output Layer: Sigmoid

Therefore the keras tuner BayesianOptimization has been applied and the output is displayed in the summary of the best hyperparameters found below:

```
{'rotation_factor': 0.06999999999999999,
 'zoom_factor': 0.05,
 'translation_height_factor': 0.060000000000000005,
 'translation_width_factor': 0.02,
 'conv_1_units': 32,
 'drop_conv_1': 0.4,
 'conv_2_units': 64,
 'drop_conv_2': 0.4,
 'conv_3_units': 192,
 'drop_conv_3': 0.3,
 'conv_4_units': 384,
 'drop_conv_4': 0.3,
 'drop_flat': 0.5,
 'dense_units_1': 192,
 'drop_dense_1': 0.4,
 'dense_units_2': 192,
 'drop_dense_2': 0.4}
```

In the end, a model with the best found hyperparameters has been fitted for 50 epochs. This happened to identify the best performing epoch, which resulted to be epoch 36 according to the validation accuracy. After that, the model has been fitted once more on the whole training and validation set for 39 epochs, to account for the now slightly bigger data set.

4.3.2 Models compilation and Fitting

Likewise to Model 1, 2 & 3, this model was compiled with the same parameters.

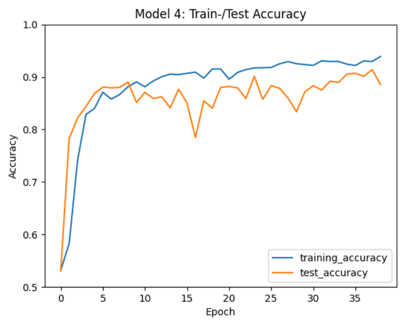
4.3.3 Model Evaluation

In figure 8 and table 4 the model evolution process and final performance is displayed:

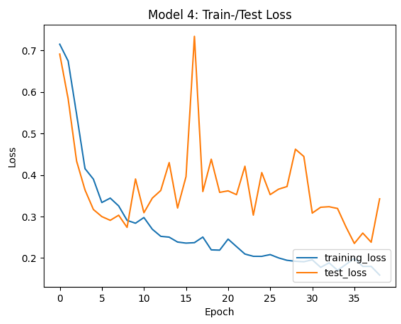
Model	**Test Accuracy**	**Test Loss**
Model 4	0.914	0.211

Table 3: Model Performance Metrics

As we can see from model 4's performance, the overfitting techniques applied were quite successful. Out of all models compiled, Model 4 has the highest test accuracy and the lowest test loss.



(a) Accuracy of Model 4



(b) Loss of Model 4

Figure 8: Model 4 results

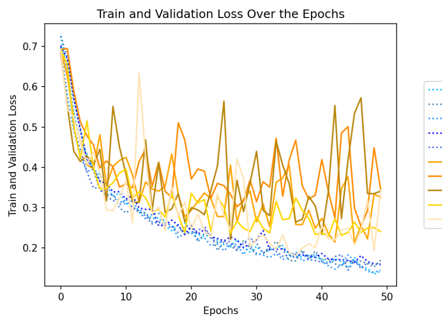
4.4 5-Fold Cross Validation Risk Estimates

For the estimation of the performance of machine learning models, often a k-Fold Cross Validation is used. With this method, the data set gets partitioned into complementary subsets, whereby the training is done on one part of the subset and the validation on the other. The process is repeated multiple times (folds), with different partitions, to reducs variability and ensure that the model's performance is as reliable as possible. Since the model 4 performed the best, the 5-fold cross validation risk estimate is performed on that model. The mean of this estimate is shown in the table below:

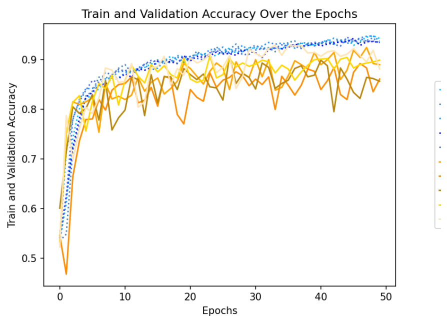
Model	**Test Accuracy**	**Test Loss**
Model 4	0.878	0.311

Table 4: Mean model performance metrics

In the end figure 9 displays the trend of the training and validation accuracy and loss over the epochs:



(a) Accuracy of Model 3



(b) Loss of Model 3

Figure 9: Model 4 CV Accuracy and Loss trend over epochs

5 Conclusion

All in all, the applied project was rather exiting. During the development process of this project, three fundamental conclusions have emerged:

- By deepening the model structures, it can be seen that convolutional neural networks learn better and thus have a better performance and generalization. However, it must be noted that such models are easily exposed to the problem of overfitting.
- By using regularization techniques such as data augmentation and dropout, overfitting can be prevented and the performance improved.
- Techniques such as hyperparameter tuning can be used as a helpful method to explore the best possible hyperparameters for the model structure.