

Neural Network Project

Statistical Methods for Machine Learning

Jan Wojdylak [mn: V09646]

Milano, 04.07.2023

1. Introduction

The aim of the project is to use neural networks for image binary classification of muffins and chihuahuas based on images provided by Kaggle dataset

<https://www.kaggle.com/datasets/samuelcortinhas/muffin-vs-chihuahua-image-classification>.

The dataset comprises a total of images of muffins and Chihuahuas. Each image is labeled accordingly, indicating whether it belongs to the "muffin" or "Chihuahua" class. Images are in JPG format and they are splitted in train and test sets.

The main goal of this report is to experiment with various network architectures, consisting of three different configurations, as well as explore different training hyperparameters for the task of binary classification, and use of a 5-fold cross-validation technique to compute accurate risk estimates.

2. Neural Network

To create neural networks architectures, I have decided to use the three Convolutional neural networks with varying layers. I chose the CNNs because of their effectiveness in image classification tasks and their ability to capture spatial relationships within images. They ensure following points:

- Localized Feature Extraction: CNNs capture local patterns and features by using convolutional layers, allowing them to focus on important image characteristics.
- Parameter Sharing: CNNs share weights across different spatial locations, enabling them to generalize well and capture spatial invariances.
- Hierarchical Representation Learning: CNNs learn hierarchical representations by stacking convolutional layers, capturing increasingly complex and abstract features.
- Translation Invariance: Pooling layers reduce spatial dimensions, making CNNs robust to variations in object position, scale, and orientation.
- Scalability: CNNs can handle images of varying sizes and resolutions, making them adaptable to different image classification tasks.
- Availability of Pretrained Models: Pretrained CNN models provide learned features and facilitate transfer learning to new classification tasks.

These are the layers that are used in CNN:

- Convolutional layers - responsible for learning and extracting features from input images. They apply a set of filters to the input, performing convolution operations to detect local patterns and features. Each filter learns to detect specific visual patterns, such as edges or textures, in different regions of the image.
- MaxPooling layers - reduce the spatial dimensions of the feature maps produced by the convolutional layers. They divide the input into non-overlapping regions and keep only the maximum value within each region. MaxPooling helps to downsample the feature maps,

reducing computational complexity and providing translation invariance, making the network more robust to variations in object position.

- Flatten layer - is used to convert the multidimensional feature maps from the previous layers into a one-dimensional vector. It "flattens" the input, preserving the information and preparing it to be fed into the fully connected layers.
- Dense layers - also known as fully connected layers, are traditional neural network layers where each neuron is connected to every neuron in the previous layer. They map the learned features to the desired output classes. Dense layers enable the network to learn complex relationships between the features and the target classes.
- Output layer - the final layer of the network responsible for producing the predictions. In binary classification tasks, the output layer typically consists of two neurons with a softmax activation function. Softmax converts the output values into probabilities, representing the likelihood of each class, and allows the network to make a prediction based on the highest probability.

These layers work together to process the input images, extract features, reduce spatial dimensions, and make predictions based on the learned representations. The specific arrangement and combination of these layers contribute to the overall architecture and functionality of the CNN models.

3. Preprocessing

In the preprocessing phase of this project, I performed several steps to prepare the dataset for training the neural network models. First, the dataset is downloaded and extracted using Kaggle CLI commands. The dataset comprises 5917 images of muffins and Chihuahuas, which are going to be classified into two categories.

Upon loading the images, each image undergoes image processing steps. OpenCV is used to read the images, and the color space is converted from JPG to RGB to ensure consistency. Additionally, all images are resized to a uniform size of 32x32 pixels. The images are then stored in the 'x' array, while the corresponding labels are stored in the 'y' array. There is no need for split data in training and test set, because since there is cross validation, the whole dataset is used for both training and testing.

Normalization of the pixel values is performed by dividing them by 255.0. This step is crucial as it brings the pixel values within a consistent range of 0 to 1. Normalization helps in preventing any particular feature from dominating the learning process and allows for better convergence during training.

One-hot encoding is used to convert the class labels into a suitable format for the classification task. It involves representing each class as a binary vector, where the index corresponding to the class is set to 1 and the rest are set to 0. This encoding scheme is employed to facilitate the learning process and enable the neural network to interpret the categorical labels effectively.

4. Cross validation

I implemented cross-validation using the 5-fold cross validation with the KFold function from the scikit-learn library. For each fold, the data was split into training and validation sets. The training set was used to train the model, while the validation set was used to evaluate its performance. This process was repeated five times, with each fold serving as the validation set once. After training the model on the training set, it was evaluated on the validation set to compute zero-one loss.

The loss metric for the training part was calculated using the binary cross-entropy loss function. This loss function is commonly used for binary classification problems. It measures the dissimilarity between the predicted probabilities and the true labels. The lower the loss value, the better the model's predictions align with the ground truth. For the test part I used zero-one loss which is defined by $1 - \text{accuracy}$. Accuracy is a metric computed to assess the model's classification performance. It represents the percentage of correctly classified samples out of all the samples.

The results from each fold, including the hyperparameters, validation loss, and validation accuracy, were stored for further analysis. The average cross-validated zero one loss was computed by taking the mean of the zero one loss from all folds.

5. Neural network architectures and hyperparameters tuning

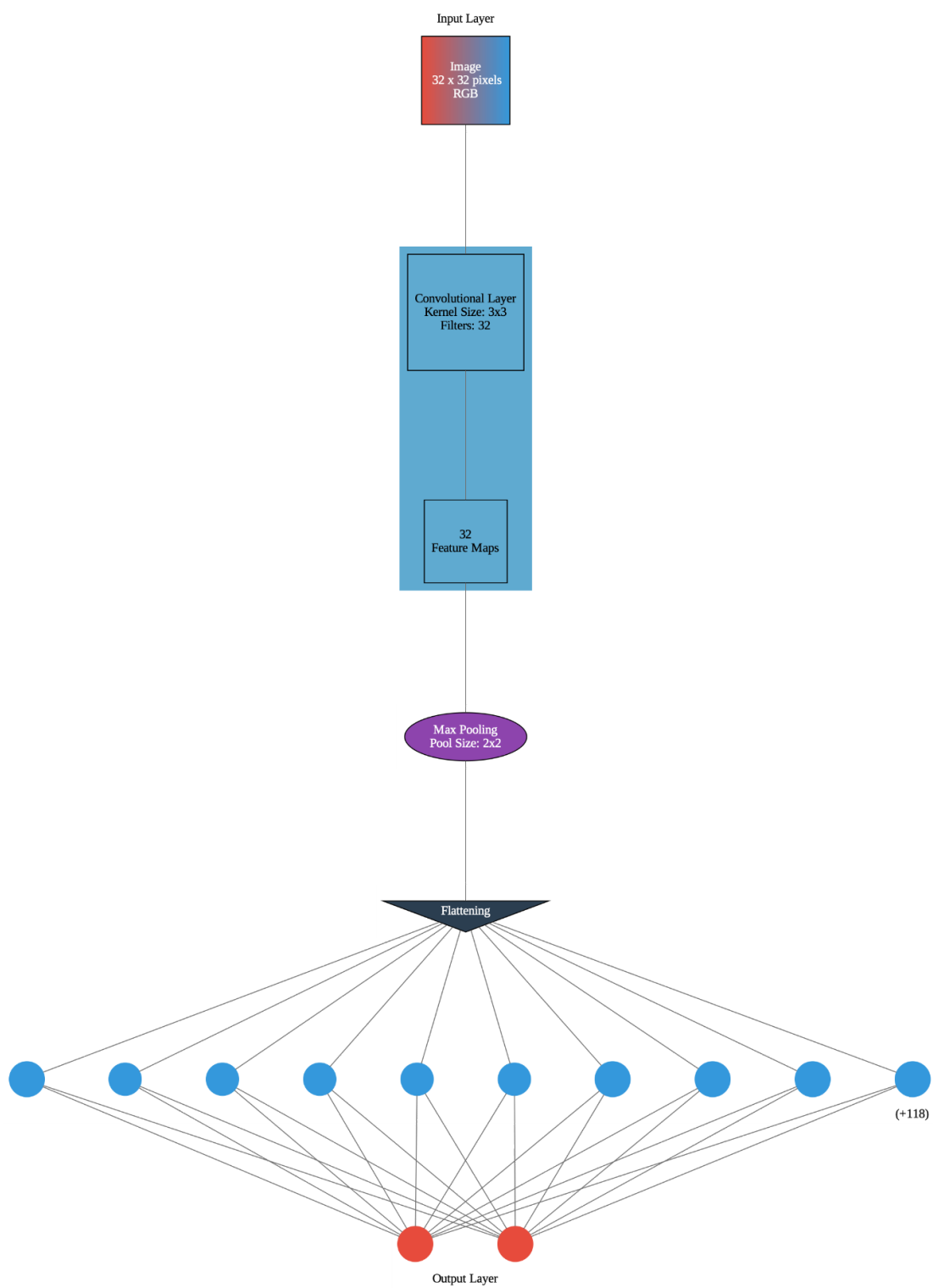
In the given code, three different neural architectures (model1, model2, model3) were implemented for image classification. Each architecture consists of a combination of convolutional layers, pooling layers, flatten layers, and dense layers.

5.1. Model 1

| Layer (type) | Output Shape | Param # |
|------------------------------|--------------------|---------|
| conv2d (Conv2D) | (None, 30, 30, 32) | 896 |
| max_pooling2d (MaxPooling2D) | (None, 15, 15, 32) | 0 |
| flatten (Flatten) | (None, 7200) | 0 |
| dense (Dense) | (None, 128) | 921728 |
| dense_1 (Dense) | (None, 2) | 258 |
| Total params: 922,882 | | |
| Trainable params: 922,882 | | |
| Non-trainable params: 0 | | |

Picture 1. Summary of model 1

- The architecture begins with a convolutional layer with 32 filters of size 3x3, using the ReLU activation function. It is followed by a max pooling layer with a pool size of 2x2, which reduces the spatial dimensions of the feature maps.
- Next, a flatten layer is used to convert the multidimensional feature maps into a 1D vector.
- A dense layer with 128 units and the ReLU activation function is added to introduce non-linearity and capture more complex patterns.
- The final layer consists of a dense layer with 2 units and the softmax activation function, which produces the probability distribution over the two classes (muffin and Chihuahua).



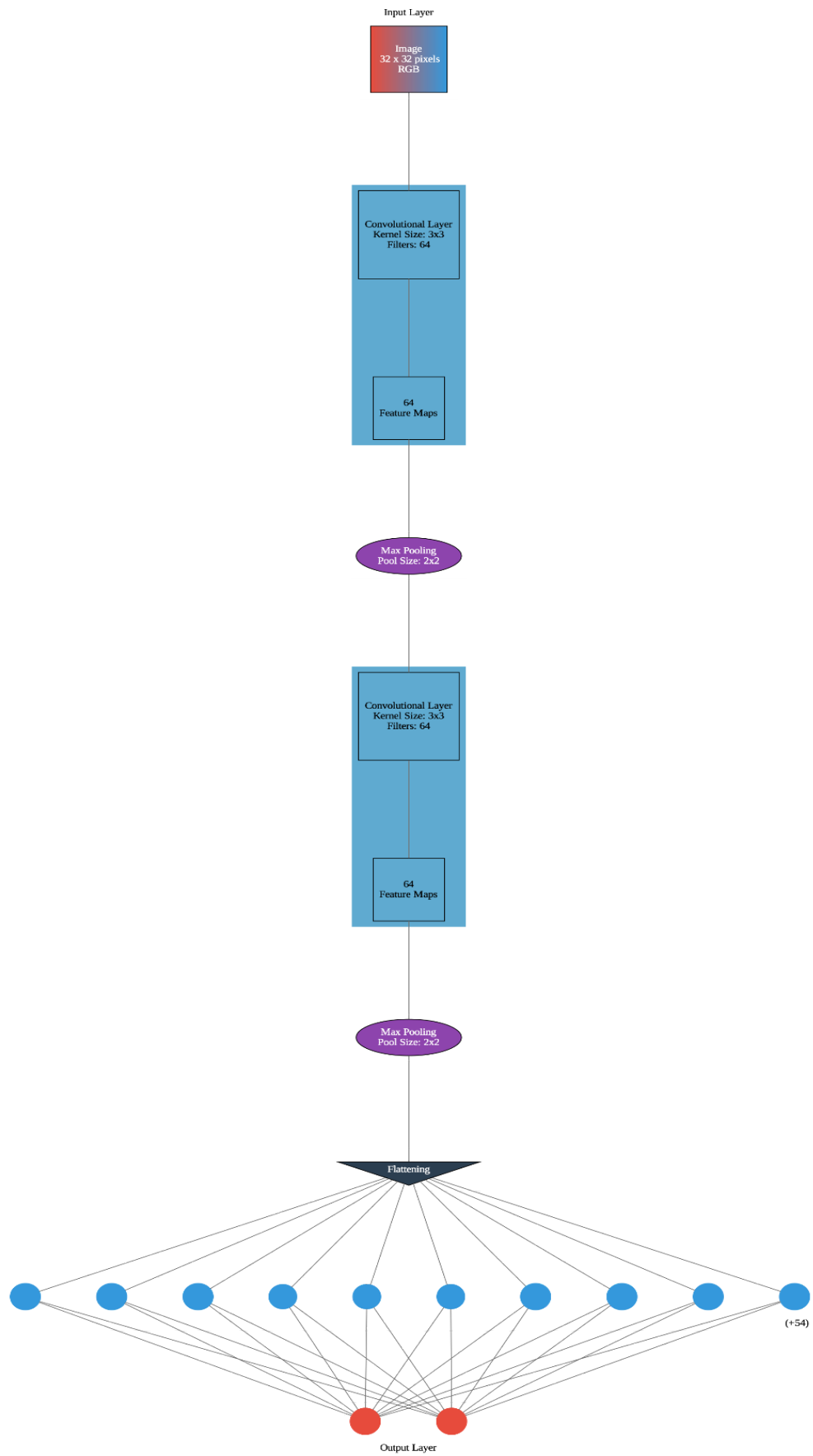
Picture 2. Visualization of model 1

5.2. Model 2

| Layer (type) | Output Shape | Param # |
|---------------------------------|--------------------|---------|
| conv2d_1 (Conv2D) | (None, 30, 30, 64) | 1792 |
| max_pooling2d_1 (MaxPooling 2D) | (None, 15, 15, 64) | 0 |
| conv2d_2 (Conv2D) | (None, 13, 13, 64) | 36928 |
| max_pooling2d_2 (MaxPooling 2D) | (None, 6, 6, 64) | 0 |
| flatten_1 (Flatten) | (None, 2304) | 0 |
| dense_2 (Dense) | (None, 64) | 147520 |
| dense_3 (Dense) | (None, 2) | 130 |
| ===== | | |
| Total params: 186,370 | | |
| Trainable params: 186,370 | | |
| Non-trainable params: 0 | | |

Picture 3. Summary of model 2

- The architecture is similar to model1 but with some differences. It starts with a convolutional layer with 64 filters of size 3x3, followed by a max pooling layer.
- Another convolutional layer with 64 filters is added, along with its corresponding max pooling layer.
- After the pooling layers, a flatten layer is used to transform the feature maps into a 1D vector.
- A dense layer with 64 units and the ReLU activation function is included.
- The final layer is a dense layer with 2 units and the softmax activation function.



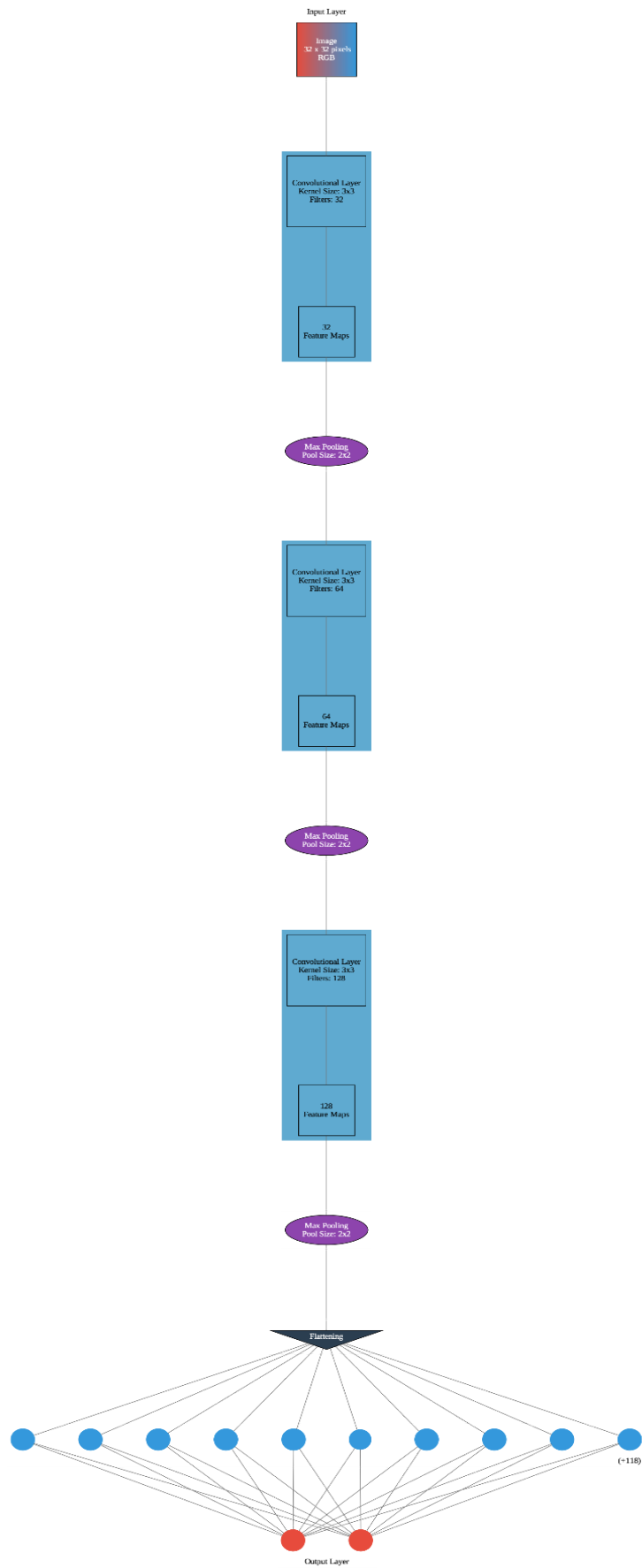
Picture 4. Visualization of model 2

5.3. Model 3

| Layer (type) | Output Shape | Param # |
|---------------------------------|--------------------|---------|
| conv2d_16 (Conv2D) | (None, 30, 30, 32) | 896 |
| max_pooling2d_16 (MaxPooling2D) | (None, 15, 15, 32) | 0 |
| conv2d_17 (Conv2D) | (None, 13, 13, 64) | 18496 |
| max_pooling2d_17 (MaxPooling2D) | (None, 6, 6, 64) | 0 |
| conv2d_18 (Conv2D) | (None, 4, 4, 128) | 73856 |
| max_pooling2d_18 (MaxPooling2D) | (None, 2, 2, 128) | 0 |
| flatten_7 (Flatten) | (None, 512) | 0 |
| dense_14 (Dense) | (None, 128) | 65664 |
| dense_15 (Dense) | (None, 2) | 258 |
| Total params: 159,170 | | |
| Trainable params: 159,170 | | |
| Non-trainable params: 0 | | |

Picture 5. Summary of model 3

- The third architecture incorporates a deeper network structure compared to the previous two models. It begins with a convolutional layer with 32 filters of size 3x3, followed by a max pooling layer.
- Two additional convolutional layers with 64 and 128 filters, respectively, are included, each followed by a max pooling layer.
- Similar to the previous models, a flatten layer is used to convert the feature maps into a 1D vector.
- A dense layer with 128 units and the ReLU activation function is added.
- The final layer is a dense layer with 2 units and the softmax activation function.



Picture 6. Visualization of model 3

5.4. Hyperparameters

In my neural network models, I have utilized three important hyperparameters: learning rate, number of epochs, and batch size.

Learning Rate: The learning rate determines the step size at which the model adjusts its internal parameters during training. It controls how much the weights are updated in response to the calculated gradients. A higher learning rate may lead to faster convergence, but it can also cause overshooting. On the other hand, a lower learning rate may result in slower convergence or getting stuck in local optima.

Number of Epochs: An epoch represents one complete pass of the entire training dataset through the neural network. The number of epochs determines how many times the network will iterate over the training data. Too few epochs may result in underfitting, where the model fails to capture the patterns in the data. Conversely, too many epochs can lead to overfitting, where the model becomes overly specialized to the training data and performs poorly on new, unseen data.

Batch Size: The batch size defines the number of training examples processed in one forward and backward pass of the network. It affects the speed of training and the memory requirements. Larger batch sizes can improve training speed, but they may also require more memory. Smaller batch sizes allow for more frequent updates to the model's weights but can slow down training.

6. Results and analysis

To determine the best combination of hyperparameters, I decided to test all configurations with different models using the following values: learning_rate: 0.01, 0.001, 0.0001, batch_size: 16, 64, 128, and epochs: 8, 16, 24. For each configuration, I measured the zero-one loss as a mean of each zero-one loss of each fold and the execution time for one iteration of cross-validation. By systematically exploring these variations, I aimed to assess the impact of different learning rates, batch sizes, and numbers of epochs on the performance of the neural network.

6.1. Model 1

The results for model 1 are shown in table 1.

| # | Epochs | Learning Rate | Batch Size | Cv zero-one loss | Training time for one cv iteration [s] |
|---|--------|---------------|------------|------------------|--|
| 1 | 8 | 0,01 | 16 | 0,280051 | 39,458 |
| 2 | 16 | 0,01 | 16 | 0,361072 | 80,918 |
| 3 | 24 | 0,01 | 16 | 0,332619 | 133,972 |
| 4 | 8 | 0,01 | 64 | 0,374107 | 25,960 |
| 5 | 16 | 0,01 | 64 | 0,291734 | 65,019 |
| 6 | 24 | 0,01 | 64 | 0,221567 | 119,169 |

| | | | | | |
|----|----|--------|-----|----------|---------|
| 7 | 8 | 0,01 | 128 | 0,401332 | 31,839 |
| 8 | 16 | 0,01 | 128 | 0,241012 | 76,445 |
| 9 | 24 | 0,01 | 128 | 0,239984 | 79,262 |
| 10 | 8 | 0,001 | 16 | 0,200769 | 58,983 |
| 11 | 16 | 0,001 | 16 | 0,160557 | 135,678 |
| 12 | 24 | 0,001 | 16 | 0,194014 | 202,707 |
| 13 | 8 | 0,001 | 64 | 0,192672 | 40,272 |
| 14 | 16 | 0,001 | 64 | 0,216665 | 82,659 |
| 15 | 24 | 0,001 | 64 | 0,199768 | 137,893 |
| 16 | 8 | 0,001 | 128 | 0,200115 | 37,962 |
| 17 | 16 | 0,001 | 128 | 0,166477 | 79,450 |
| 18 | 24 | 0,001 | 128 | 0,193504 | 125,745 |
| 19 | 8 | 0,0001 | 16 | 0,163599 | 71,976 |
| 20 | 16 | 0,0001 | 16 | 0,159718 | 122,844 |
| 21 | 24 | 0,0001 | 16 | 0,205666 | 202,739 |
| 22 | 8 | 0,0001 | 64 | 0,196217 | 41,002 |
| 23 | 16 | 0,0001 | 64 | 0,174421 | 81,960 |
| 24 | 24 | 0,0001 | 64 | 0,193180 | 142,679 |
| 25 | 8 | 0,0001 | 128 | 0,246765 | 40,295 |
| 26 | 16 | 0,0001 | 128 | 0,187426 | 82,696 |
| 27 | 24 | 0,0001 | 128 | 0,195034 | 127,201 |

Table 1. Results for model 1

The best three hyperparameter configurations for minimizing zero-one loss are: 20, 11 and 19.

To examine the influence of different parameter values, I calculated the average zero-one loss and training time for each parameter and recorded the results in the following tables:

| Learning Rate | Zero-One Loss | Time (s) |
|---------------|---------------|----------|
| 0.01 | 0.304831 | 72.449 |
| 0.001 | 0.191316 | 100.150 |
| 0.0001 | 0.191636 | 101.488 |

Table 2. Learning rate comparison for model 1

Lower learning rates (0.001 and 0.0001) yield lower zero-one loss values compared to a learning rate of 0.01. However, they also result in longer training times.

| Batch Size | Zero-One Loss | Time (s) |
|------------|---------------|----------|
| 16 | 0.228674 | 116.586 |
| 64 | 0.228926 | 81.846 |
| 128 | 0.230183 | 75.655 |

Table 3. Batch size comparison for model 1

The zero-one loss values are relatively similar across different batch sizes (16, 64, 128), with no significant differences. However, the best three results were for batch size 16. On the other hand, smaller batch sizes tend to require longer training times.

| Number of Epochs | Zero-One Loss | Time (s) |
|------------------|---------------|----------|
| 8 | 0.250625 | 43.083 |
| 16 | 0.217676 | 89.741 |
| 24 | 0.219482 | 141.263 |

Table 4. Number of epochs comparison for model 1

Increasing the number of epochs generally leads to lower zero-one loss values, but the difference between 16 and 24 epochs is small. For a bigger number of epochs there are longer training times.

For this model after analyzing the results I draw the following conclusions. For lower learning rates (0.001 and 0.0001) yield lower zero-one loss values compared to a learning rate of 0.01. This indicates that a smaller learning rate allows the model to make more precise updates to the weights, resulting in better convergence and lower loss. However, lower learning rates also result in longer training times.

For Batch Size the zero-one loss values are relatively similar across different batch sizes (16, 64, 128) in model 1, with no significant differences. This suggests that the choice of batch size does not have a substantial impact on the model's performance in this case. Smaller batch sizes tend to require longer training times due to the more frequent updates of the weights. However, the differences in training times between the different batch sizes in model 1 are not significant.

Increasing the number of epochs generally leads to lower zero-one loss values, indicating improved performance as the model has more iterations to learn from the data.

However, the difference in zero-one loss values between 16 and 24 epochs is small, suggesting that the additional epochs may not significantly contribute to further reducing the loss. With a higher number of epochs, the training time also increases. It is important to find a balance between achieving lower loss and managing training time efficiently.

Overall, in model 1, lower learning rates and a higher number of epochs tend to result in lower zero-one loss values. Batch size does not show a significant impact on the zero-one loss in this case.

6.2. Model 2

For model 2 I got following results:

| # | Epochs | Learning Rate | Batch Size | Cv zero-one loss | Training time for one cv iteration [s] |
|----|--------|---------------|------------|------------------|--|
| 1 | 8 | 0,01 | 16 | 0,400137 | 80,705 |
| 2 | 16 | 0,01 | 16 | 0,278680 | 181,931 |
| 3 | 24 | 0,01 | 16 | 0,436821 | 256,928 |
| 4 | 8 | 0,01 | 64 | 0,214272 | 75,995 |
| 5 | 16 | 0,01 | 64 | 0,214141 | 139,534 |
| 6 | 24 | 0,01 | 64 | 0,227998 | 251,315 |
| 7 | 8 | 0,01 | 128 | 0,269589 | 72,010 |
| 8 | 16 | 0,01 | 128 | 0,192324 | 139,978 |
| 9 | 24 | 0,01 | 128 | 0,204838 | 201,631 |
| 10 | 8 | 0,001 | 16 | 0,131824 | 133,696 |
| 11 | 16 | 0,001 | 16 | 0,139940 | 226,476 |
| 12 | 24 | 0,001 | 16 | 0,164446 | 334,735 |
| 13 | 8 | 0,001 | 64 | 0,125916 | 105,643 |
| 14 | 16 | 0,001 | 64 | 0,144163 | 185,995 |
| 15 | 24 | 0,001 | 64 | 0,127095 | 251,094 |
| 16 | 8 | 0,001 | 128 | 0,169513 | 77,149 |
| 17 | 16 | 0,001 | 128 | 0,162917 | 191,693 |
| 18 | 24 | 0,001 | 128 | 0,127765 | 262,739 |
| 19 | 8 | 0,0001 | 16 | 0,175948 | 142,715 |
| 20 | 16 | 0,0001 | 16 | 0,167493 | 263,074 |
| 21 | 24 | 0,0001 | 16 | 0,157505 | 373,899 |
| 22 | 8 | 0,0001 | 64 | 0,258438 | 142,834 |
| 23 | 16 | 0,0001 | 64 | 0,159382 | 197,884 |
| 24 | 24 | 0,0001 | 64 | 0,194030 | 314,408 |
| 25 | 8 | 0,0001 | 128 | 0,268900 | 133,134 |
| 26 | 16 | 0,0001 | 128 | 0,226808 | 197,946 |
| 27 | 24 | 0,0001 | 128 | 0,181512 | 303,355 |

Table 6. Results for model 2

The best three hyperparameter configurations for minimizing zero-one loss are: 13, 15 and 18. For model 2 I also calculated following average values:

| Learning Rate | Zero-One Loss | Time (s) |
|---------------|---------------|----------|
| 0.01 | 0.270978 | 155.559 |
| 0.001 | 0.143731 | 196.580 |

| | | |
|--------|----------|---------|
| 0.0001 | 0.198891 | 229.917 |
|--------|----------|---------|

Table 7. Learning rate comparison for model 2

A learning rate of 0.001 yields the lowest zero-one loss value, indicating better predictive performance compared to the other learning rates. For the decreasing number of learning rates, the training time is increasing.

| Batch Size | Zero-One Loss | Time (s) |
|------------|---------------|----------|
| 16 | 0.228088 | 221.573 |
| 64 | 0.185048 | 184.967 |
| 128 | 0.200463 | 175.515 |

Table 8. Batch size comparison for model 2

The batch size 64 achieves the lowest average of zero-one loss. With the increasing number of batch sizes, the time is decreasing.

| Number of Epochs | Zero-One Loss | Time (s) |
|------------------|---------------|----------|
| 8 | 0.223837 | 107.098 |
| 16 | 0.183411 | 199.506 |
| 24 | 0.202446 | 283.345 |

Table 9. Number of epochs comparison for model 2

The best given average value for zero-one loss is for the number of 16. Naturally, for more epochs the training time is longer.

Based on the results for model 2, I draw the following conclusions regarding the impact of learning rate, batch size, and number of epochs on the zero-one loss and training time:

Among the three tested learning rates (0.01, 0.001, 0.0001) in model 2, a learning rate of 0.001 yields the lowest zero-one loss value. This indicates that a smaller learning rate allows the model to make more precise updates to the weights and results in better predictive performance.

The training time increases as the learning rate decreases. This is because smaller learning rates require more iterations to converge, leading to longer training times.

Model 2 shows that a batch size of 64 achieves the lowest average zero-one loss. This suggests that a moderate batch size can effectively capture the patterns in the data and lead to better predictive performance. Interestingly, the training time is relatively similar for batch sizes 64 and 128 in model 2. This may be due to the specific characteristics of the dataset or the model architecture used. Among the tested number of epochs in model 2, the lowest average zero-one loss value is achieved with 16 epochs. This indicates that increasing the number of epochs beyond 16 does not significantly contribute to further reducing the loss. As expected, the training time increases as the number of epochs increases. More epochs require more iterations to train the model, resulting in longer training times.

In summary, for model 2, a learning rate of 0.001, a batch size of 64, and around 16 epochs appear to be the optimal hyperparameter configurations for minimizing zero-one loss. These configurations lead to better predictive performance while also managing the training time efficiently. I

6.3. Model 3

For model 3 these are the calculated values:

| # | Epochs | Learning Rate | Batch Size | Cv zero-one loss | Training time for one cv iteration [s] |
|----|--------|---------------|------------|------------------|--|
| 1 | 8 | 0,01 | 16 | 0,859509725 | 75,79416437 |
| 2 | 16 | 0,01 | 16 | 0,575114227 | 192,8911489 |
| 3 | 24 | 0,01 | 16 | 0,708199495 | 254,7099952 |
| 4 | 8 | 0,01 | 64 | 0,458120966 | 77,18917809 |
| 5 | 16 | 0,01 | 64 | 0,369377553 | 142,1593581 |
| 6 | 24 | 0,01 | 64 | 0,238973439 | 242,7976195 |
| 7 | 8 | 0,01 | 128 | 0,247248328 | 75,85853934 |
| 8 | 16 | 0,01 | 128 | 0,174251497 | 141,0171684 |
| 9 | 24 | 0,01 | 128 | 0,192165756 | 201,6922361 |
| 10 | 8 | 0,001 | 16 | 0,208200490 | 137,04847 |
| 11 | 16 | 0,001 | 16 | 0,168173838 | 263,3265848 |
| 12 | 24 | 0,001 | 16 | 0,139602852 | 383,139579 |
| 13 | 8 | 0,001 | 64 | 0,201311231 | 112,7271017 |
| 14 | 16 | 0,001 | 64 | 0,139256859 | 203,1484431 |
| 15 | 24 | 0,001 | 64 | 0,137404335 | 323,3699697 |
| 16 | 8 | 0,001 | 128 | 0,207013774 | 81,84665523 |
| 17 | 16 | 0,001 | 128 | 0,152251077 | 183,1302708 |
| 18 | 24 | 0,001 | 128 | 0,131996775 | 257,1092425 |
| 19 | 8 | 0,0001 | 16 | 0,188935876 | 143,0799482 |
| 20 | 16 | 0,0001 | 16 | 0,154290366 | 263,2899737 |
| 21 | 24 | 0,0001 | 16 | 0,141807067 | 380,1441409 |
| 22 | 8 | 0,0001 | 64 | 0,223272252 | 143,2119673 |
| 23 | 16 | 0,0001 | 64 | 0,257241523 | 189,5568923 |
| 24 | 24 | 0,0001 | 64 | 0,160893631 | 240,9050732 |
| 25 | 8 | 0,0001 | 128 | 0,219712245 | 88,68867693 |
| 26 | 16 | 0,0001 | 128 | 0,195377505 | 154,5108853 |
| 27 | 24 | 0,0001 | 128 | 0,146094689 | 219,4244633 |

Table 10. Results for model 3

The best three hyperparameter configurations for minimizing zero-one loss are: 18, 15 and 14.

The average values for model 3 are:

| Learning Rate | Zero-One Loss | Time (s) |
|---------------|---------------|----------|
| 0.01 | 0.424773 | 156.012 |
| 0.001 | 0.165023 | 216.094 |
| 0.0001 | 0.187514 | 202.535 |

Table 11. Learning rate comparison for model 3

A learning rate of 0.001 achieves the lowest zero-one loss value, indicating better predictive performance compared to the other learning rates. It also exhibits a shorter training time compared to the learning rate of 0.0001.

| Batch Size | Zero-One Loss | Time (s) |
|------------|---------------|----------|
| 16 | 0.349315 | 232.603 |
| 64 | 0.242872 | 186.118 |
| 128 | 0.185124 | 155.920 |

Table 12. Batch size comparison for model 3

Batch size 128 ensures the lowest training error with the lowest training time.

| Number of Epochs | Zero-One Loss | Time (s) |
|------------------|---------------|----------|
| 8 | 0.312592 | 103.938 |
| 16 | 0.242815 | 192.559 |
| 24 | 0.221904 | 278.144 |

Table 13. Number of epochs comparison for model 3

In this model with a high number of epochs the zero-one loss is decreasing and time is increasing.

For this model I draw the following conclusions: Among the three tested learning rates (0.01, 0.001, 0.0001) in model 3, a learning rate of 0.001 achieves the lowest zero-one loss value. This suggests that a smaller learning rate allows the model to make more precise updates to the weights and improve the predictive performance. The training time for the model is shorter with a learning rate of 0.001 compared to 0.0001, indicating that a moderate learning rate can achieve better performance without significantly increasing the training time. Model 3 shows that a batch size of 128 yields the lowest average zero-one loss. This indicates that using larger batches helps the model capture the data patterns more effectively and leads to improved predictive performance. The training time decreases

as the batch size decreases. This is because smaller batch sizes require more iterations to process the entire dataset, leading to longer training times.

Among the tested number of epochs in model 3, increasing the number of epochs generally leads to lower zero-one loss values. This suggests that allowing the model to train for more epochs improves its ability to learn and make better predictions. The training time increases as the number of epochs increases. This is expected since more epochs require more iterations to train the model, resulting in longer training times.

In summary, for model 3, a learning rate of 0.001, a batch size of 128, and around 24 epochs appear to be the optimal hyperparameter configurations for minimizing zero-one loss. These configurations lead to better predictive performance while managing the training time reasonably well.

6.4. Model comparison

By comparing the results among the three models and analyzing the tables, the following general observations can be made:

Learning Rate:

- Across all models, a learning rate of 0.001 achieved the lowest zero-one loss values compared to 0.01 and 0.0001. This indicates that a moderate learning rate allows the model to find a better balance between convergence and avoiding getting stuck in local optima.
- The training times for different learning rates varied among the models but did not follow a consistent pattern. Therefore, the impact of learning rate on training time seems to be model-dependent.

Batch Size:

- There is no clear trend in the zero-one loss values based on the batch size. The performance varies across the models and depends on the specific combination of other hyperparameters.
- The training times tend to decrease as the batch size increases. This can be attributed to the more efficient parallel processing of larger batches.

Number of Epochs:

- The zero-one loss values generally decrease as the number of epochs increases. This indicates that allowing the model to train for more iterations improves its performance and ability to learn complex patterns in the data.
- However, longer training times are associated with a higher number of epochs. This is expected as more epochs require more iterations of forward and backward passes during training.

To compare performance with different models I calculated average loss and training time.

| Model | 1 | 2 | 3 |
|---------------|----------|----------|----------|
| Zero-one loss | 0,217346 | 0,197010 | 0,259104 |
| Time (s) | 92,185 | 198,377 | 191,547 |

Table 14. Average values of loss and training time for 3 models

- Model 2 achieved the lowest zero-one loss values across different hyperparameter settings and the best overall result with loss equals to 0,125916.

- Model 3 also performed well, achieving competitive results with a slightly higher zero-one loss than Model 2.
- Model 1 generally had higher zero-one loss values compared to the other two models. However, it still demonstrated reasonable performance and could potentially benefit from further optimization.

Overall, the choice of the model architecture plays a crucial role in the performance of the neural network. The results suggest that Model 2 might be the most effective and efficient choice among the three models tested.

6.5. Image size

All models were tested on images resized to a dimension of 32x32 pixels. However, I also wanted to explore the impact of scaling the images to a larger size, specifically 64x64 pixels to check if scaling the images to a larger size improves the models' classification performance. To investigate this, I decided to select the hyperparameters and model configuration that yielded the lowest zero-one loss values in the previous experiments: model2, learning rate = 0,001 and batch size = 64. I run code for 8, 16 and 24 epochs. Results are in the following table.

| # | Epochs | Learning Rate | Batch Size | Cv zero-one loss | Training time for one cv iteration [s] |
|---|--------|---------------|------------|------------------|--|
| 1 | 8 | 0,001 | 64 | 0,142137 | 261,719 |
| 2 | 16 | 0,001 | 64 | 0,120339 | 540,888 |
| 3 | 24 | 0,001 | 64 | 0,135884 | 791,965 |

Table 15. Results for 64x64 image

Results are quite better than for 32x32 images, but considering training time for one iteration, it is not worth using a bigger size of images for this model.

7. Reproducibility and Code Availability

In this project, the code was developed using Google Colab, a cloud-based platform that allows for easy sharing and collaboration on Jupyter notebooks. Google Colab provides a convenient environment for executing code, as it offers pre-installed libraries and resources necessary for machine learning tasks. Additionally, the only requirement to run the code is to have Kaggle credentials for downloading the dataset. By providing these credentials, other researchers can readily reproduce experiments and obtain the same dataset used in the project.

I declare that this material, which I now submit for assessment, is entirely my/our 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/our 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/us or any other person for assessment on this or any other course of study.