Chihuahuas v Muffins

Tobias Walker, Student-ID: 990443

Master of Data Science and Economics

Machine Learning, September 2023

1. Introduction

The objective of this paper is to construct a binary image classifier based on a dataset from Kaggle comprising 5917 pre-classified images categorized as either Chihuahuas or muffins. The dataset exhibits greater diversity compared to the famous example often used as an introductory example. It encompasses a wide range of images, including those of muffin dough and collages featuring tattoos, one of which depicts a dog in a tracksuit. The images underwent several pre-processing steps. Firstly, they were converted into the RGB format. Subsequently, pixel values were normalized to fall within the [0,1] range. Lastly, the images were resized to a uniform 256x256 pixel format, ensuring consistency in data dimensions. These steps were chosen to enhance the model's ability to discern relevant features. The dataset was partitioned into three distinct sets: a training set consisting of 3786 images, a validation set comprising 947 images, and a test set comprising 1184 images. The partitioning scheme adheres to the distribution provided by the data source.

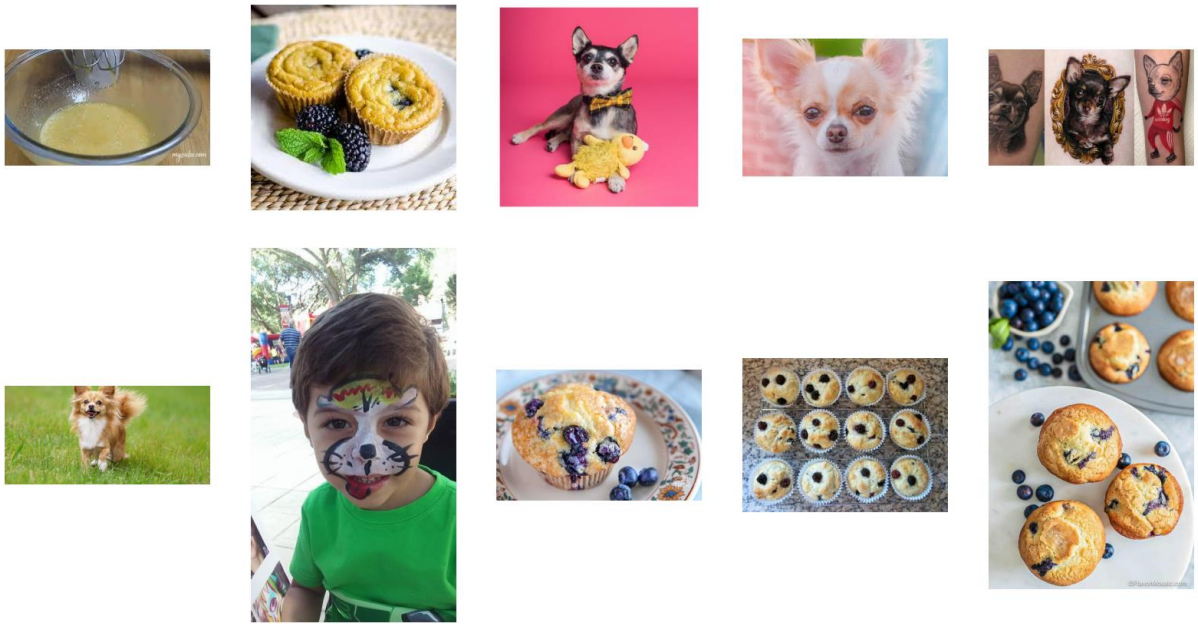

Figure 1: Random selection of images from the dataset (own illustration)

This paper adopts a systematic approach to investigate the evolution of Convolutional Neural Network (CNN) architectures. It begins with a straightforward baseline model and progressively explores the effects of architectural changes and solutions to common challenges. Model 1 serves as the baseline architecture. With its simple configuration, it establishes a reference point for performance evaluation. Model 2 introduces a more complex architecture. By doing so, we assess the impact of increased model complexity on classification performance. Recognizing overfitting issues in both the baseline and complex models, Model 3 implements countermeasures to combat overfitting. This includes the application of regularization techniques and hyperparameter tuning. On this model, 5-fold cross-validation is also performed. Lastly, to attain the highest accuracy, Model 4 employs transfer learning. This approach harnesses the InceptionV3 pre-trained neural network and fine-tunes the last layer on the dataset.

2. Arcitecture guidelines and theoretical concepts

The models built in this study (excluding the transfer learning model) adhere to a consistent construction guide. This guide consists of several key steps: Initially, a series of convolutional blocks is employed. Each block comprises a convolutional layer followed by a max-pooling layer. These convolutional blocks play a pivotal role in feature extraction from the input data. Following the convolutional blocks, a flattening layer is introduced to transition from the spatially organized data in the convolutional layers to a format suitable for fully connected layers. Two dense layers follow the flattening layer. The first dense layer captures interconnections among the features extracted from the convolutional blocks. This enables the model to learn complex representations of the data. The second dense layer is responsible for making the final binary classification decision.

A convolutional layer is the fundamental component within a CNN. It contains a collection of learnable filters, represented as 2D matrices, which systematically slide over the input data. At each position, these filters, fine-tuned during the training process, perform a mathematical operation called convolution. In essence, convolution involves taking the element-wise product between the filter and the corresponding segment of the input data it currently overlays. This step is followed by the summation of these products to produce a single value for each position. Subsequently, this computed value undergoes activation through a function, which, in this context, is the Rectified Linear Unit (ReLU):

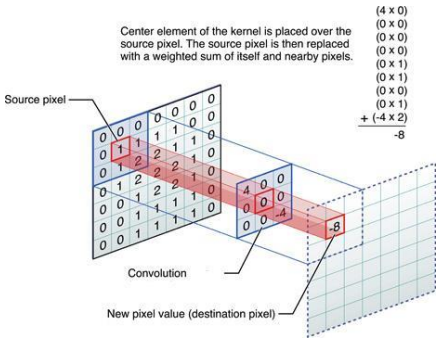

Figure 2: Simple example of convolution (external source)

$$ReLU(x) = \max(0, x).$$

The resultant values obtained for each position collectively constitute the output map. The primary objective of a convolutional layer is to extract distinctive features from the input data, such as edges, textures, or patterns. In this project, the size of the convolutional filters is established as 3x3, and to ensure that the output map maintains the same dimensions as the input data, padding is configured as 'same'. The number of filters employed varies across the Models.

A max-pooling layer is employed to reduce the dimensions of the output from a convolutional layer. It operates independently on each feature map and retains only the maximum value within a small window while discarding the rest. This approach reduces computational load and helps the network concentrate on the most critical features. In this study, the size of the window is set to 2x2.

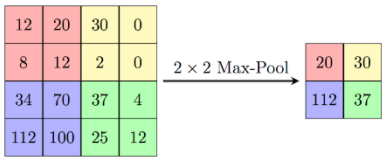

Figure 3: Simple example of max-pooling (external source)

A flatten layer transforms the multi-dimensional output from the preceding layer, in this paper, this is always a max-pooling layer, into a one-dimensional vector. This step is necessary to transition from a convolutional layer to a fully connected layer.

A dense layer, also known as a fully connected layer, is a type of neural network layer where each neuron is connected to every neuron in the preceding layer. These connections have associated weights that determine the strength and sign of the connection. These weights are adjusted during the learning process. Each neuron's output undergoes an activation function. In each model constructed in this paper, two dense layers are employed. The first one uses a ReLU activation function and has a model-specific number of neurons, aiming to capture relationships and interactions between

the extracted features. The second dense layer consists of only one neuron with a Sigmoid activation function. The Sigmoid function is defined as:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}}$$

and returns a value between 0 and 1. This layer's purpose is to predict the class to which the input belongs.

A total of three measures are implemented against overfitting. An early stop condition based on validation loss is applied in every model. Data augmentation and dropout are utilized only in the third model. The early stop condition monitors a specified metric and halts the training process if the metric fails to improve for a certain number of consecutive iterations. Additionally, the option to restore the best weights, i.e., the weights that resulted in the best metric value, is enabled in every model. Dropout layers serve as a regularization technique that randomly set a fraction of the neuron's output from the previous layer to zero during each training iteration. This encourages the network to learn more robust and independent features, as it cannot rely on specific neurons always being present. Dropout is applied after convolutional and dense layers. The data augmentation process is not a traditional layer but rather a set of operations applied to the input before feeding it into the network. These operations can include rotation, scaling, cropping, flipping images, and zooming, among others. They are performed before every training iteration on the training data, enhancing the model's ability to generalize by exposing it to a broader range of variations in the training data.

The optimizer employed in this project is the Adaptive Moment Estimation (Adam) optimizer. Adam is chosen for its fast convergence, light computational load, and memory efficiency.

The models are evaluated using two metrics: loss and accuracy. Loss measures the disparity between the predicted value and the actual label. In this paper, two types of loss functions are employed. In general, binary cross-entropy

$$l(y, \hat{y}) = -[y * \log(\hat{y}) + (1 - y) * \log(1 - \hat{y})]$$

is used to train the models. Binary cross-entropy considers the predicted value $\hat{y}$ (which can range from 0 to 1, while the true labels y are either 0 or 1) and penalizes the model more severely when it makes predictions that deviate significantly from the true labels. For 5-fold cross-validation, the zero-one loss is utilized. The zero-one loss function is defined as follows:

$$l(y, \hat{y}) = \begin{cases} 1 \; if \; y \neq C(\hat{y}) \\ 0 \; if \; y = \; C(\hat{y}) \end{cases} with \; C(\hat{y}) = \begin{cases} 1 \; if \; \hat{y} \geq 0.5 \\ 0 \; if \; \hat{y} < 0.5 \end{cases}$$

The second evaluation metric employed in this paper is accuracy. Accuracy calculates the ratio of correctly classified samples to the total number of samples. Since the dataset lacks a dominant class, accuracy, in conjunction with loss, serves as a suitable metric for assessing predictive quality.

3. Model 1: Baseline model

The first model serves as the foundation upon which subsequent models are built. It consists of two convolutional blocks. In the initial block, the input data is in the format of 256x256x3, and it undergoes a convolutional layer with 32 filters. Following this, a max-pooling layer reduces the image dimensions to 128x128. The second block takes as input the resulting data format from the first block, which is 128x128x32, and applies 64 filters. The subsequent max-pooling layer reduces the dimensions to an output format of 64x64x64. The data is then flattened into a one-dimensional vector of size $64^3 = 262{,}144$. This vector is used as input in the first dense layer, which consists of 128 neurons and

employs a ReLU activation function. The resulting 128 values are then reduced to a single value, which is sent through a Sigmoid activation function to determine the predicted value.

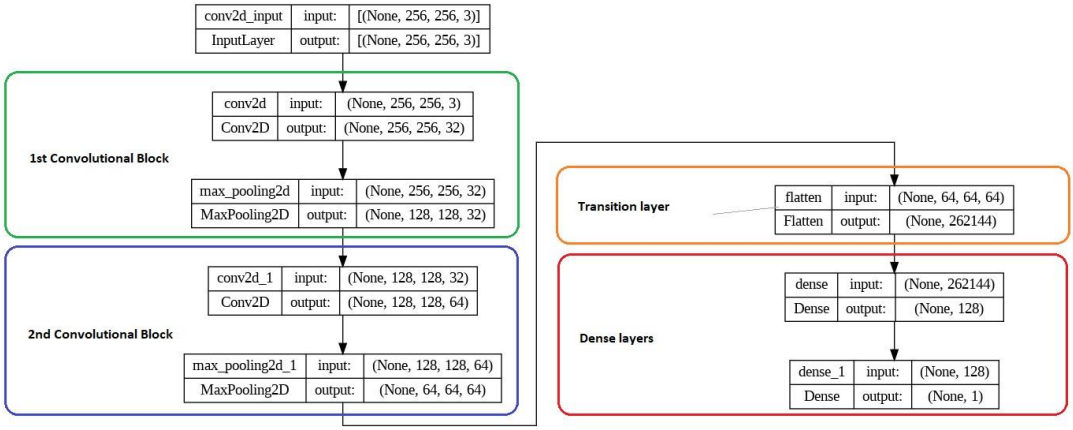

Figure 4: Set-up of the baseline model (own illustration)

The model is trained with an early stop condition that halts training and restores the best weights if the validation loss doesn't improve for three consecutive iterations. The training set combines the training and validation sets, while the validation set corresponds to the test set. The learning rate is set to 0.001, and the batch size is 32.

The baseline model achieves its best performance after the second iteration, with a test loss of 0.4167 and a test accuracy of 84.46%. The model might have a problem with overfitting, as its training accuracy after the fifth iteration is already at 98.8%, while the test accuracy remains constant at around 85%.

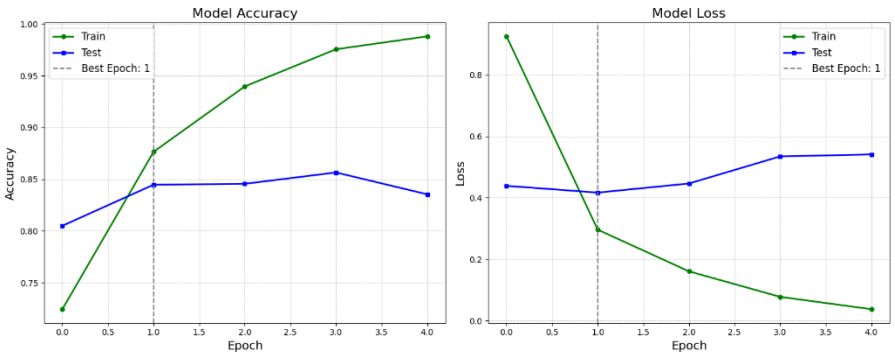

Figure 5: Development of the accuracy and the loss in the baseline model (own illustration)

4. Model 2: A more complex model

In this model, the impacts of a more complex architecture are explored. This increased complexity is achieved by introducing an additional convolutional block, increasing the number of filters in the convolutional layers, and expanding the number of neurons in the first dense layer. In contrast to the baseline model, which had 32 filters in the first convolutional layer, 64 filters in the second convolutional layer, and 128 neurons in the initial dense layer, the more complex model adopts 64, 128, and 256 filters in its convolutional layers, with 256 neurons in its first dense layer.

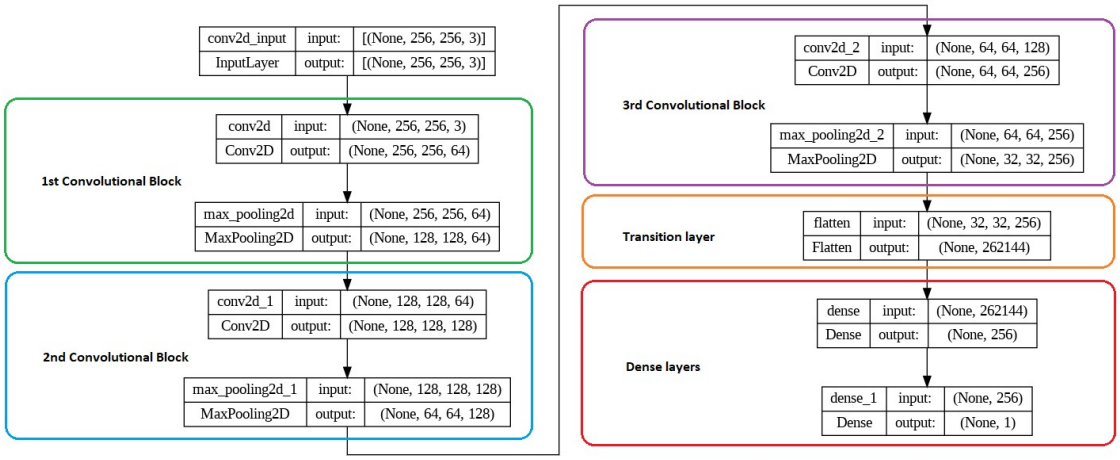

Figure 6: Set-up of the more complex model (own illustration)

The early stopping mechanism is set to activate after three consecutive iterations without improvement. The learning rate is fixed at 0.001, and the batch size is set to 32. The training dataset comprises both the training and validation datasets, with the test dataset serving as the validation set.

The training process concluded due to the early stopping condition after the 7th iteration, achieving its best performance during the 4th iteration. In this iteration, the model achieved a validation loss of 0.3265 and a validation accuracy of 86.73%. Remarkably, the more complex model displayed improvements in both loss and accuracy compared to the baseline model. This enhancement suggests that the complex model possesses an improved ability to generalize.

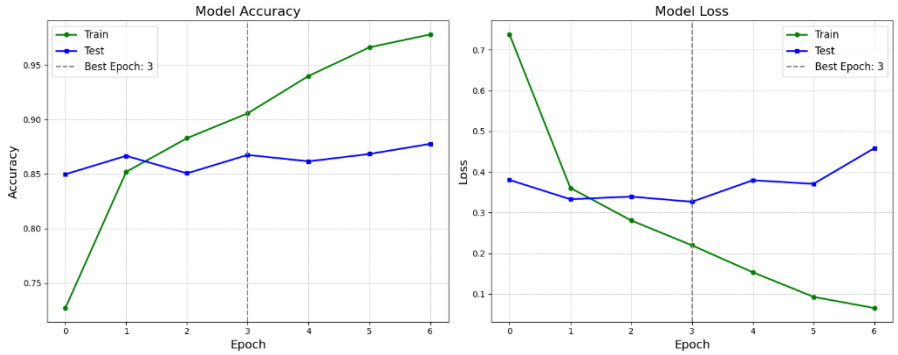

Figure 7: Development of the accuracy and the loss in the more complex model (own illustration)

This observation is reinforced by the fact that, while the baseline model attained a training accuracy of 98.85% in the 5th iteration, the more complex model achieved a training accuracy of 93.98% in the same iteration. This outcome, somewhat unexpected given the conventional association of increased complexity with a higher risk of overfitting, implies that the architecture's complexity may have facilitated the model in capturing deeper underlying patterns and extracting more relevant features from the dataset before eventually encountering overfitting issues. In contrast, the baseline model may have primarily focused on the obvious features present in the training set. Therefore, the heightened complexity allowed the model to develop a more comprehensive understanding of the dataset, resulting in improved performance on the unseen validation set, particularly in the early iterations and before eventually also overfitting.

5. Model 3: Introducing measures against overfitting

To address the issue of overfitting, a set of countermeasures is introduced. These countermeasures include the addition of a data augmentation layer at the beginning of the architecture, as well as dropout layers after each max-pooling layer and after the initial dense layer. Furthermore, hyperparameters tuning has been applied to improve the model's predictive performance.

5.1 Adding data augmentation and dropout layers to the more complex model

In the data augmentation layer, random horizontal and vertical flips, rotations of up to 5%, and zooming in height and width by up to 5% are applied before each iteration over the training set. These operations might alter image sizes, so the input and output shapes are set to 'None' in the model representation (Figure 8). Additionally, dropout layers with a dropout rate of 10% are introduced after each max-pooling layer and after the 1st dense layer.

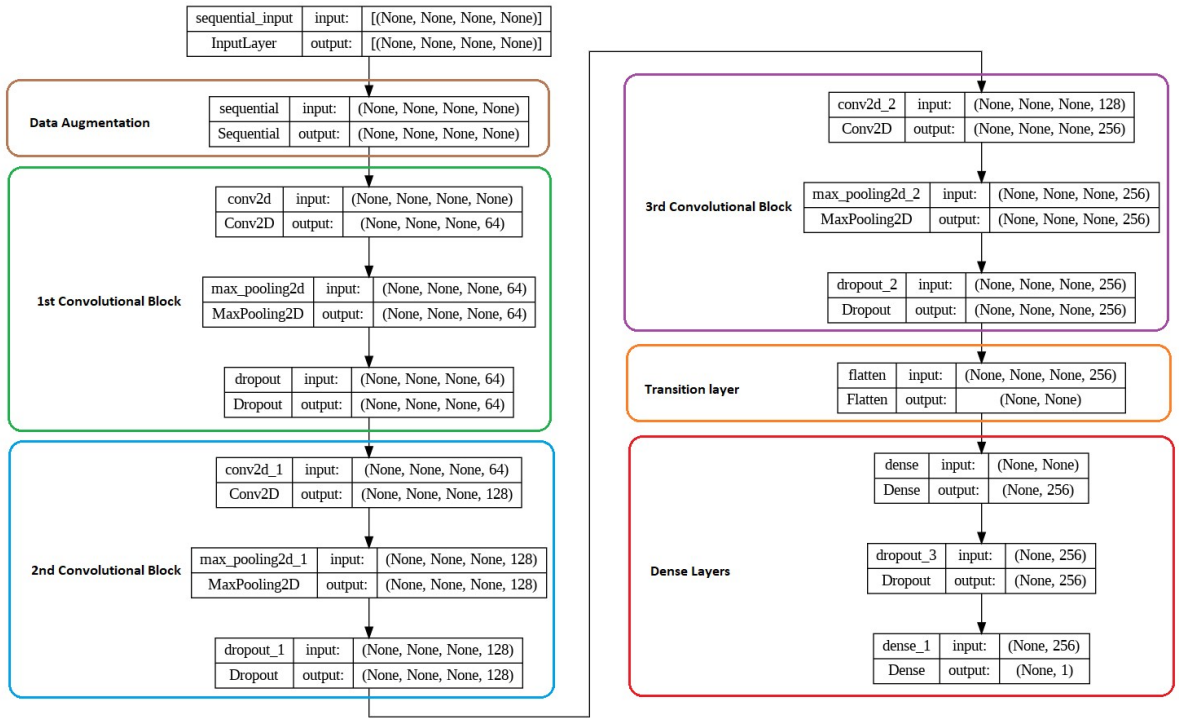

Figure 8: Set-up of the more complex model with countermeasures against overfitting (own illustration)

The early stopping mechanism with a patience of five in this model halted training after the 29th iteration, with the best iteration being the 24th. It achieved a validation loss of 0.2233 and a validation accuracy of 91.13%. This approach, which involves training the model on slightly modified images and preventing over-reliance on specific neurons or features, effectively mitigates the overfitting issue. This is evident in Figure 9, where the test metrics align more closely with the training metrics.

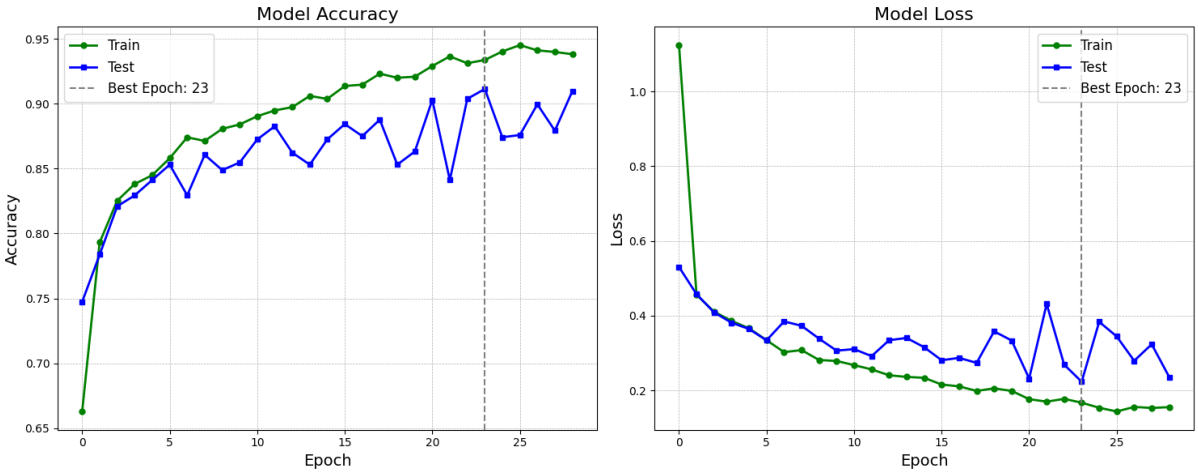

Figure 9: Development of the accuracy and the loss in the more complex model with countermeasures against overfitting (own illustration)

The fluctuations in validation accuracy and loss, as depicted in Figure 9 after epoch 17, might be attributed to a combination of a too-high learning rate and too aggressive data augmentation or dropout rates. These fluctuations could potentially undermine the model's ability to generalize beyond the validation set. Therefore, in the hyperparameter tuning phase, a crucial goal should be to attain a more stable validation performance. This can be accomplished by implementing cross-validation in conjunction with an early stop condition that does not restore the best weights. However, due to resource limitations, this paper was unable to incorporate cross-validation into the hyperparameter tuning process. The resulting model therefore might also struggle with stability issues.

5.2 Hyperparameter-tuning of the more complex model with countermeasures

The tuning of the hyperparameters unfolded as follows: a grid of potential hyperparameter values (as outlined in Table 1) was presented to the searcher, which randomly selected 15 combinations. These combinations were subsequently trained on the training set with an early stop condition employing a patience of five and restoring the best weights. After each training iteration, the model was evaluated on the validation set. Following the completion of all 15 models, the one with the highest validation accuracy was chosen. The selected model was then retrained on the combined training and validation datasets and assessed on the test set.

Table 1: Grid of possible hyperparameter values given to the searcher (own illustration)

Hyperparameter		**Possible values**
Data augmentation	Rotation	0, 0.02, 0.04, 0.06, 0.08, 0.1
	Zoom	0, 0.02, 0.04, 0.06, 0.08, 0.1
Convolutional layers	Filters in 1st layer	32, 64, 128
	Filters in 2nd layer	64, 128, 256
	Filters in 3rd layer	128, 256, 512
Dense layer	Neurons in 1st layer	64, 128, 256
Dropout	Rate after max-pooling layer	0 to 0.2 in steps of 0.02
	Rate after 1st dense layer	0 to 0.2 in steps of 0.02

Optimizer	Learning rate	0.005, 0.001, 0.0005, 0.0001

The model with the highest accuracy utilized a rotation value of 4%, a zoom value of 10%, 128 filters in the 1st convolutional layer, 128 in the 2nd, and 512 in the 3rd layer. It employed a dropout rate of 14% after each max-pooling layer and recommended 256 neurons with a dropout rate of 16% for the 1st dense layer. The optimal learning rate was determined to be 0.0005. With these hyperparameters, the model achieved a test loss of 0.2106 and an accuracy of 91.81%. This represents a slight improvement compared to the previous model without fine-tuned hyperparameters. Like the previous model, this one also grapples with performance fluctuations. Given that the hyperparameter tuning process only considered a single validation set, these instabilities are not unexpected. If the instabilities remain also after a tuned model with cross-validation, they might result from the architecture. In this case, the general architecture set-up might have to be reviewed.

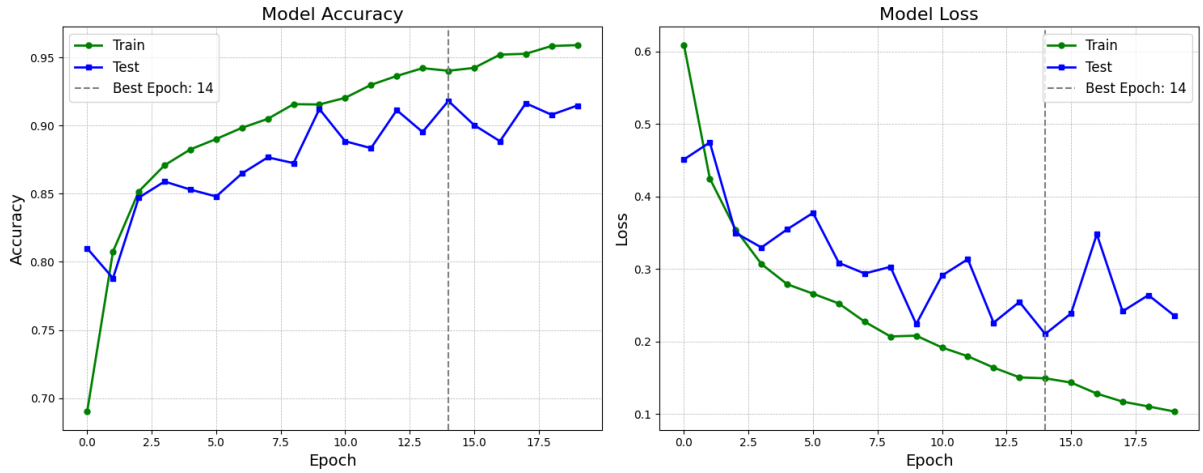

Figure 10: Accuracy and loss of the hyperparameter tuned model (own illustration)

5.3 Cross validation on the hyperparameter tuned model

To conduct a comprehensive assessment of the tuned model's performance, a 5-fold cross-validation was carried out. The entire dataset was divided into five equally sized parts, and the model was then trained on four parts whiles being validated on the remaining one. This process was repeated five times, ensuring that each fold served as the validation set once. To account for and potentially highlight the instabilities, the number of epochs was set to 15, and no early stop condition was imposed. The model was trained using a batch size of 32.

The validation accuracy across the five folds was as follows: [88.26%, 88.94%, 88.42%, 90.03%, 89.35%], resulting in an average accuracy of 89%. The zero-one loss, calculated as 1 - validation accuracy, amounted to 11%. This is significantly lower than the accuracy achieved by the model optimized solely based on one validation set, underscoring the challenges posed by instability and the importance of conducting a thorough model evaluation that includes cross-validation. Often, a balance must be struck between stability and achieving peak validation accuracy.

6. Model 4: Transfer learning

For the final model, transfer learning is leveraged. The InceptionV3 model, pre-trained on the extensive ImageNet database with millions of labels spanning thousands of categories, is employed. To adapt the InceptionV3 model for the specific task, its dense layers are removed and substituted with a flatten layer followed by a dense layer containing a single neuron and a Sigmoid activation function. The

weights of the original InceptionV3 model layers remain fixed throughout the training process, meaning that only the weights of the newly added dense layer are fine-tuned. The model is trained with a learning rate of 0.001 and an early stop condition with a patience of 3, restoring the best weights.

The best validation loss is attained during the 5th iteration, recording a loss of 0.0806 and achieving a remarkable test accuracy of 99.24%. This performance significantly surpasses that of the models trained solely on the Chihuahua v Muffin dataset. Pre-trained models like InceptionV3 or the VGG models often present an excellent option for image classification tasks with small and general datasets, showcasing their capability to leverage prior knowledge for improved performance.

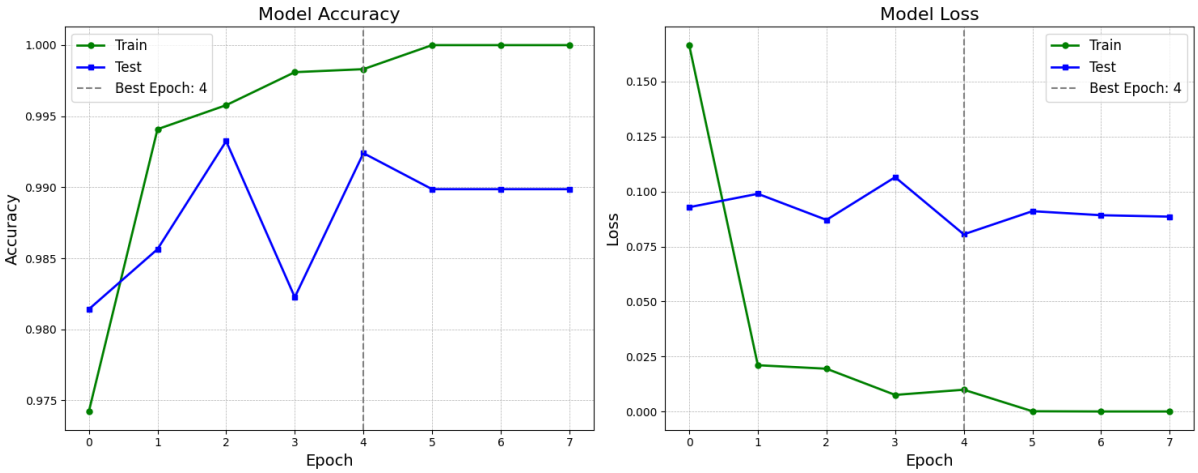

Figure 11: Accuracy and loss of the transfer learning model (own illustration)

7. Conclusion

Even relatively simple models can produce remarkable results on limited datasets. However, the pursuit of the optimal model surpasses the mere pursuit of the highest validation accuracy. Several other crucial metrics and factors must be considered, including addressing overfitting and model stability. Overfitting arises when a model learns to fit the training data too closely, resulting in poor generalization to unseen data. Model stability is essential for real-world applications, as it ensures consistent performance outside of the dataset, making the model more reliable. For datasets containing somewhat general images, it is often helpful to explore the possibility of using pre-trained models. These models have been trained on vast and diverse datasets, enabling them to extract essential features that may be challenging to capture with models trained exclusively on small datasets. By carefully considering these factors and striking a balance between them, one can identify a model that not only achieves high accuracy but also exhibits robust performance, making it suitable for practical use beyond the dataset.

External sources:

Figure 2: https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411

Figure 3: https://paperswithcode.com/method/max-pooling