Exercise 3
Topic 2: Deep Learning for Images

184.702 Machine Learning (VU 3,0) 2023S

Group 34

July 31, 2023

1 Introduction

This exercise deals with image classification as fundamental task in the field of computer vision. Image classification is explored using diferent approaches: baseline models, MLP, CNN (from scratch) and transfer learning. The pupular highlevel API keras, based on the tensorflow framework is used for building models. The datasets, used in this exercise are "CIFAR-10" and "10 Monkey Faces". CIFAR-10 is widely used as a benchmark dataset containing 60,000 small images divided into 10 classes. The 10 Monkey Faces dataset consits of 10 different monkey species with similarities and fine-grained differences between the classes.

2 Datasets

The CIFAR-10 and the "10 Monkey Species" datasets differ in the types of images they contain and the specific classification task they address. While the CIFAR-10 is a more general dataset with diverse object classes, the 10 Monkey Faces dataset focuses on distinguishing between visually similar monkey species.
Both datasets are quite balanced but differ in size and resolution. This requires different training-strategies including data augmentation and transfer learning.

CIFAR-10

The CIFAR-10[1] dataset consists of 60,000 images with a 32x32 resolution and 3 color channels. The data is labeled in 10 classes representing specific objects or categories: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck. Each class is balanced with 6,000 images per class. The comparably low resolution makes it partly challenging for the human eye to recognize the images.

10 Monkey Species

The 10 Monkey Species dataset from kaggle[2] consists of about 1,642 images in the format 400x300 or larger with 3 color changels of ten different monkey species. It is a highly specific dataset with fine-grained differences between the classes that requires human expert knowledge to distiguish between mantled howler, patas monkey, bald uakari, japanese macaque, pygmy marmoset, white headed capuchin, silvery marmoset, common squirrel monkey, black headed night monkey and nilgiri langur.

[1]https://www.cs.toronto.edu/ kriz/cifar.html

[2]https://www.kaggle.com/datasets/slothkong/10-monkey-species?resource=download

3 Methodology

This Section inlcudes overall information about preprocessing, used libraries and KPIs.

3.1 Data Collection and Preprocessing

CIFAR-10:
Because of its popularity it would have been possible to download the CIFAR-10 dataset via tensorflow.keras.datasets.cifar10 and split it directly into training and test data in the required 32x32x3 resolution using the method `load_data()`.
For this exercise, the data was downloaded via https://www.cs.toronto.edu/ kriz/cifar-10-python.tar.gz, double unzipped and saved in a "data" folder in the project directory.
The data downloaded via the manual process consists of 6 batches of numpy.ndarrays with 10000 x 3072 entries per batch. Which means 10000 samples and 3072 bits, describing the same-sized pictures. Like shown in figure 1, the origin data is a row vector per picture which has to be reshaped and transposed to get a displayable form.

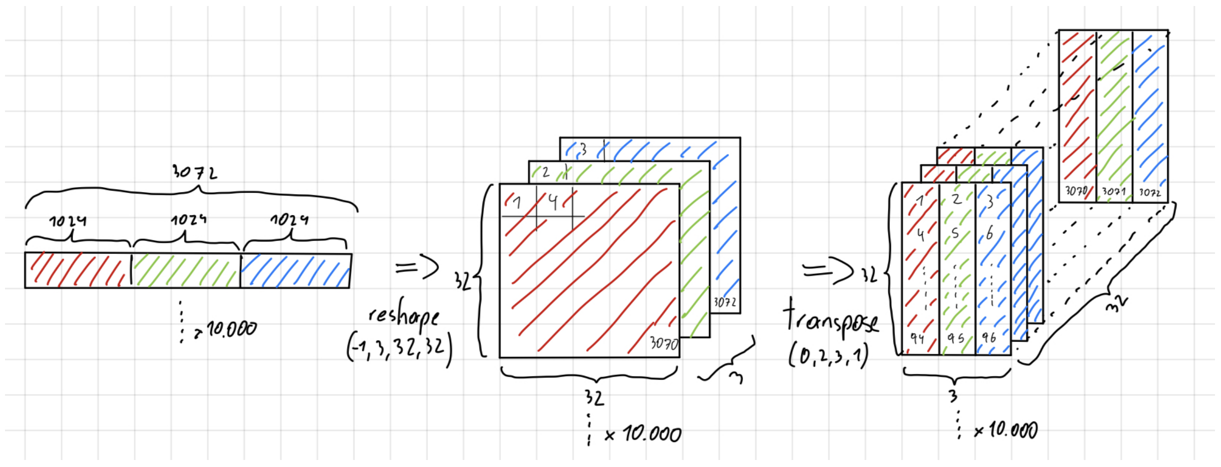


Figure 1: reshaping and transforming to 32x32x3

5 batches are provided for training/validation and 1 batch is for testing. The reshaped and transposed data is used directly for MLP and CNN. Because of the quantity of 5000 training images per class, no data augmentation is used. For usage in transfer learning models, the images were rescaled to an 192x192x3 (6 times size).

10 Monkey Species:
The "10 Monkey Species" dataset consists of 1370 labeled train images and 272 validation images with an image size of 400x300x3 and larger. The 10 classes are evenly balanced. The data was downloaded from https://www.kaggle.com/datasets/slothkong/10-monkey-species?resource=download and unzipped. On one hand, for the baseline models, the images were resized to a (400x300) dimension. On the other hand, for the neural network models, the images were reshaped to (224x224x3) using "flow_from_directory" method "ImageDataGenerator" class from tensorflow.keras.preprocessing.image. A bigger size would have led to more trainable parameters and a longer computation time (without GPU usage).

3.2 Models and feature extraction:

This section deals with the different used models for image classification. Due to the differences in the datasets regarding size and classification specifics, different approaches were tested including baseline methods to basic Convolutional Neural Networks and tranfer learning. In the initial phase of this project, various feature extraction techniques were employed to capture image characteristics. These "traditional" feature extraction methods were then used to train and test models using conventional methods similar to those applied in handling tabular data. These models will serve as the baseline for further evaluation and analysis.

CIFAR-10:

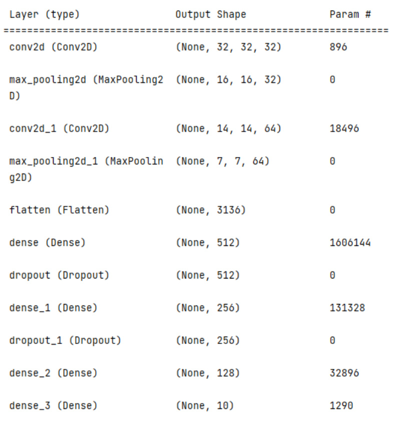
Baseline:
In the initial step, a straightforward feature representation based on image color histograms was employed. Each image was represented using three channels (red - R, green - G, blue - B), and separate histograms were created for each channel. As the ultimate feature representation, these one-dimensional arrays were concatenated to form a combined representation. Using the extracted features, three different classifiers were implemented: k-NN (k-nearest neighbors), SVM (support vector machine), and MLP (multi-layer perceptron). These classifiers were trained and evaluated to make predictions based on the feature representations obtained from the image color histograms. Despite RGB being a widely used color representation in various applications, the HSV color space offers a perceptually more uniform and intuitive way of understanding and controlling colors. The Euclidean distance between colors in the HSV space better aligns with human perception of color differences. In the HSV color space, the H - hue component represents the color itself, S - saturation represents the purity or intensity of the color, and V - value represents the brightness or lightness of the color. Therefore, we opted to create a feature representation of images by concatenating histograms corresponding to the H, S, and V channels. As an additional feature extraction technique, we chose to explore the application of the Sobel filter for edge detection, with the aim of identifying boundaries between different objects or regions in an image. The Sobel filter operates by convolving an image with small kernels or matrices (usually 3x3) in both the horizontal and vertical directions. These kernels are designed to estimate the gradient of the image intensity at each pixel. The Sobel operator highlights edges in an image by computing the intensity gradient in both the x (horizontal) and y (vertical) directions, and then combining them to determine the overall edge strength. Sobel filter typically operates on grayscale images (1 channel) and we decided to experiment with this approach by flattening the 2D edges representation of image to a 1D feature vector. Despite the potential limitations of flattening the 2D data, we considered it worth exploring its applicability in our feature extraction process.

As a more sophisticated, yet still "traditional" approach to feature extraction, we implemented SIFT (Scale-Invariant Feature Transform) and subsequently applied the visual bag of words technique, as suggested in the project's description. SIFT is an algorithm that identifies keypoints and computes their corresponding descriptors. Keypoints represent points of interest in an image, and descriptors capture the local features by extracting small, fixed-size (128) patches (converted into vectors) around each keypoint, describing the surrounding area. Here, we will provide a concise explanation of the feature extraction process and the concept of the visual bag of words. As mentioned earlier, each image in the training dataset is transformed into a set of descriptors using the SIFT, resulting in an array of size m x 128 (m - number of keypoints). The subsequent step involves creating a visual vocabulary. First, all the descriptors are grouped into a predefined number of clusters using the K-means algorithm. These K centroids, known as "visual words", collectively constitute the visual vocabulary. This vocabulary effectively captures the distinct local patterns present in the dataset. Next, each descriptor in every image from the training dataset is assigned to the nearest visual word (centroid) from the visual vocabulary, determined by their Euclidean distance. This assignment yields a histogram representation of the image, with each bin in the histogram corresponding to a visual word from the vocabulary. The value of each bin indicates the frequency of occurrence of the corresponding visual word in the image. Once the images are represented as histograms of visual words, a classifier is trained using these histogram representations. Classifiers that we trained in this part are again: k-NN, SVM, and MLP.

Regarding Neural Networks, MLP and CNN architectures with different trainable parametersizes were tested for the image classification. Regarding nomenclature, an "S", "M" or "L" for small, medium or large is attached to MLP or CNN.

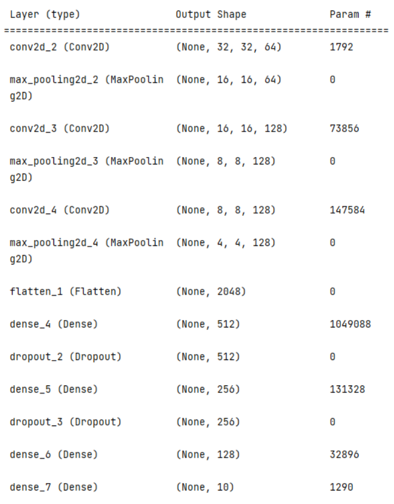
Multilayer perceptron:
For exercise purpose the keras subclassing API was used for creating the MLP models for the CIFAR-10 dataset. This API is, compared to the sequential and the functional API, not as straightforward and comprehensible and not as easily cloneable and shareable but allows involving loops, varying shape and other dynamic behaviour. Therefore a MLP-Class was created, that takes a list with neurons per layer (e.g. [500, 500, 500] for 3 hidden layers with each 500 neurons) and the number of output classes as input. MLP_S and MLP_M consist of a 3 hidden layers (each with 500 neurons for small and 1000 neurons for medium) between a Flatten layer and the output layer. MLP_L consists of 3 hidden layers with 3000, 3000, and 1000 neurons between a Flatten layer and the output layer. The number of output classes is 10 for each model due to the 10 different labels.
In total, this results in 2,042,510 trainable parameters for the small model, 5,085,010 trainable parameters for the medium model and 21,233,010 trainable parameters for the large model.

Convolutional Neural Networks:
CNNs emerged from the study of the brain´s visual cortex and consist basically of so called convolutional blocks and a prediction blocks. The basic idea of the convolutinal block is to learn the network recognizing different shapes and forms. The complexity of this form is increasing with deeper convolutional layers. Therefore multiple, so called "filters" are trained in each convolutional layer to represent different shapes (e.g. vertical and horicontal lines) and forms (e.g. edges). This is done by "scanning" the channels of the previous or input layer with a set of filters. After convolution, the features are pooled by a so called pooling layer to reduce the size of the layers to the most essential. This convolution and pooling can be done multiple times. After the convolution block a so called prediction block (basically a MLP or a Global Average Pooling Layer combined with a Flatten layer and a output layer) is used for predicting the classes.
In the process of several undocumented experiments, 3 models (small medium and large) were also selected for the exercise. figures 2, 3 and 4 show the different architectures of the own created CNNs. figure 2 shows a simple CNN with 2 convolution blocks (32 and 64 filters) and an attached MLP with 3 hidden layers. The total number of trainable parameters is 1,038,922.
The medium CNN has 3 convolutional blocks and 3 hidden layers in the prediction block. In total it has 2,641,674 trainable parameters The large CNN, displayed in 4 has 3 convolutional layers also attached to each a MaxPooling layer. The attached prediction block MLP has 3 Hidden Layers and 2 50% Dropout layers. In total there are 2,633,994 trainable parameters for the large CNN model.



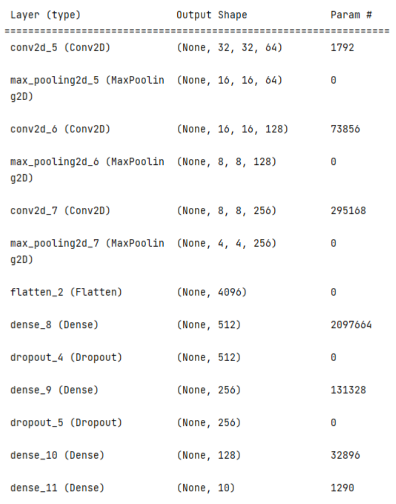
```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 64)          0
 g2D)

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 512)               1606144

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 128)               32896

 dense_3 (Dense)             (None, 10)                1290
```

Figure 2: CNN_S



```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 32, 32, 64)        1792

 max_pooling2d_2 (MaxPoolin  (None, 16, 16, 64)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 16, 16, 128)       73856

 max_pooling2d_3 (MaxPoolin  (None, 8, 8, 128)         0
 g2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_4 (MaxPoolin  (None, 4, 4, 128)         0
 g2D)

 flatten_1 (Flatten)         (None, 2048)              0

 dense_4 (Dense)             (None, 512)               1049088

 dropout_2 (Dropout)         (None, 512)               0

 dense_5 (Dense)             (None, 256)               131328

 dropout_3 (Dropout)         (None, 256)               0

 dense_6 (Dense)             (None, 128)               32896

 dense_7 (Dense)             (None, 10)                1290
```

Figure 3: CNN_M



```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 32, 32, 64)        1792

 max_pooling2d_5 (MaxPoolin  (None, 16, 16, 64)        0
 g2D)

 conv2d_6 (Conv2D)           (None, 16, 16, 128)       73856

 max_pooling2d_6 (MaxPoolin  (None, 8, 8, 128)         0
 g2D)

 conv2d_7 (Conv2D)           (None, 8, 8, 256)         295168

 max_pooling2d_7 (MaxPoolin  (None, 4, 4, 256)         0
 g2D)

 flatten_2 (Flatten)         (None, 4096)              0

 dense_8 (Dense)             (None, 512)               2097664

 dropout_4 (Dropout)         (None, 512)               0

 dense_9 (Dense)             (None, 256)               131328

 dropout_5 (Dropout)         (None, 256)               0

 dense_10 (Dense)            (None, 128)               32896

 dense_11 (Dense)            (None, 10)                1290
```

Figure 4: CNN_L

An even larger CNN with an additional convolution layer with 128 filters before the last convolution layer with 256 filters was also tested but led to no further improvement. For simplicity reasons, this XL model will not be investigated further in this document.

Transfer learning models:
Since the classes of the CIFAR-10 dataset are also included in the requested classes of modern image classification competitions, sophisticated state of the art models can be used for classifying the images. For this dataset, the Inception-v3 model, developed by Google, is used. The Model requires images with minimum 75x75x3 resolution. Therefore, the CIFAR-10 images had to be rescaled as described in chapter 3.1.
For transfer learning, the Inception-v3 model was imported without top layers as base model and attached to a prediction block consisting of a flatten layer and an MLP with [512, 256, 128, 10] layers with 2 dropout layers in between to reduce overfitting. This results in a model with 16,943,242 trainable and 21,802,784 non-trainable parameters.

10 Monkey Species:
Baseline:
Essentially, identical feature extraction techniques (with the exception of not using the Sobel filter), were employed, and the identical models were trained as in the previous dataset.

CNN from scratch:
Due to the compareable small dataset size, the influence of data augmentation was tested on CNN_L model architecture from the CIFAR-10 dataset.

CNN transfer learning:
Main focus of the 10 Monkey Species was on transfer learning. Inception-v3 and VGG16 had been tested. The models were imported without top layers. MLP or a Global Average Pooling Layer attached to a 10 neuron output layer were tested as different methods for Inception-v3. Since VGG16 was only used for comparison purposes, only the more promising configuration with Global Average Pooling Layer was used for VGG16.

3.3 Model training

This section describes the training process and the different refinements per dataset and model class. For all neural network models, a batch size of 32 and early stopping with a patience of 5 or 10 and checkpoints for saving and restoring weights were used callback methods during the training process.

CIFAR-10:
Baseline:
As previously mentioned, as baseline for this task, we used multiple "traditional" feature extraction techniques whose output then served as input to our classifiers. For example, first we trained our models separately on color histograms created by RGB and HSV. Then the mega histogram created using SIFT technique was used as input for our kNN, SVM and MLP. In addition, due to unsatisfactory results, a feature combination was also created and used to train the models.

MLP and CNN from scratch:
After importing and rescaling the data, the train data was splitted into 90% train data and 10 % validation data by using the train_test_split class from sklearn. Categorical crossentropy was used as loss function because the targets were one-hot-encoded. Early stopping with a patience of 10 was used to stop out the train process and restore the best weights for each model.

CNN transfer learning:
As described in 3.2, the top layers from the Inception model were excluded for the base model. There-

fore, as usual with transfer learning, only the newly attached prediction block was fitted during training. No fine tuning of the last base model layers were done.

10 Monkey Species:
baseline:
For the monkey dataset, we followed a similiar approach as for the CIFAR dataset. The kNN, SVM and MLP classifiers were trained using the output of "traditional" feature extraction techniques.

CNN from scratch:
For the 10 Monkey Species dataset, an explicit validation dataset was not used. Instead, the validation results, which are also used for the training stop, are used as test results. Focus in this exercise was on the differencies made by data augmentation for one CNN model (CNN_L from the CIFAR-10 dataset), made from scratch (with 103,297,290 trainable parameters because of the higher resolution compared to the CIFAR-10 dataset). Therefore two different training pipelines, using the ImageDataGenerator class form tensorflow.keras.preprocessing.image library were created as input for the "model.fit" method. While one pipeline did not use any data augmentation technique, the other used the following:

- rotation range = 40
- width shift range, height shift range and zoom range = 0.2
- horizontal flip = True

CNN transfer learning:
The transfer learning models used data augmentation as described in the paragraph before. Focus was on the differencies in fitting the prediction block (either MLP or Global Average Pooling Layer before output layer) with and without fine tuning the last layers of the base model. For the Inception-v3 base model, 4 models were created. Two of them with MLP as prediction block and 2 with a Global Average Pooling Layer before the output layer. Each 2 models prediction blocks were trained with "Adam" solver one time till early stopping callback stopped the process. The respective other 2 models were trained trained for 15 epochs with "Adam" solver. Afterwards the top 5 layers from the base model were set trainable and it was then trained until stop by early stopping callback with "Stochastic Gradient Descent" solver, learning rate of 0.1 and momentum of 0.9.

3.4 Model evaluation

Because of the large number of models created and investigated, and the refinements in the training process, an equally extensive evaluation is omitted. Therefore, only the accuracy as metric for model evaluation is used.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ Observations} \tag{1}$$

True Positives (TP)	number of correctly predicted positive instances
False Positives (FP)	number of incorrectly predicted positive instances
True Negatives (TN)	number of correctly predicted negative instances
False Negatives (FN)	number of incorrectly predicted negative instances

Comparison and interpretation of results:
Due to the large number of models created and examined and the subtleties in the training process, the finest differences in the results of similar models are not discussed in detail because explanatory attempts with scientific standard would be difficult within the scope of this exercise. The confusion matrices are only given for the most expressive models. Regardless of the results, the models are compared in terms of training and validation loss and accuracy.

Conclusion: Finally, the report concludes with a summary of the models' performance and the insights gained from the analysis. The limitations of the study and potential areas for future research are also discussed.

4 Results

This chapter includes the findings and results from the analysed datasets. It is separated into subsections for each dataset It is referred to the models, described in chapter 3.2 and to the processes and settings from chapter 3.1

4.1 CIFAR-10

For the CIFAR-10 dataset, beside the baseline models, basic Multi Layer Percepron (MLP) and Convolutional Neural Networks (CNN) architectures as well as a transfer learning method were tested.

BASE models
The baseline models served as a good technique to get to know and explore our data. However, as expected, the results were not very satisfactory. What was very interesting is that one would expect the SVM and MLP,as powerful models, to have the best results, however, they were both overun by kNN which is a much simpler approach.
As the initial traditional feature extraction method, RGB histograms were generated using the "calculate_color_histogram" function, and Figure 5, illustrates an example from the dataset. The figure and the presence of yellow feathers on the bird confirm that in the RGB color system, yellow is represented as (255, 255, 0), indicating full R and G intensity, with no B intensity. It is worth noting that the images in this dataset were encoded as uint8 (unsigned 8-bit integers), which explains the presence of values ranging from 0-255 on the histograms. The process of extracting RGB histograms and concatenating them on the training dataset required approximately 16 minutes, while on the testing dataset, it took approximately 3 minutes. Additionally, the functions "calculate_histogram_hsv" and "sobel_filter" were implemented to extract HSV color histograms and edge information (multiple visualizations can be found in the notebook). The effectiveness of these features was evaluated using a k-NN classifier. Although there was an expectation of improved performance with HSV compared to RGB, the results did not show a significant improvement. As a result, this part of the assignment will primarily concentrate on RGB color histograms and classifiers trained using these features.

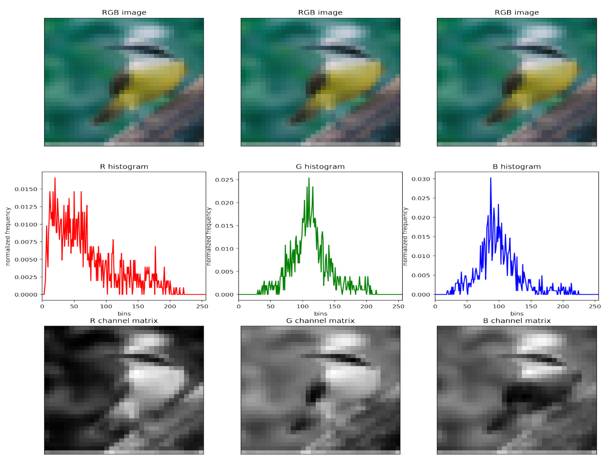


Figure 5: RGB color histograms for one dataset example

The following table gives information on accuracy of the different classifiers. The table presents results obtained with k-NN using k=3, SVM with a regularization parameter c=0.8, and MLP composed of 4 layers. The data was based on the training/test split predefined by the dataset authors (with the training set further divided into training and validation sets during MLP training). It's important to note that hold-out or cross-validation methods were not utilized, and the metrics provided are not the final evaluation due to this approach. Despite k-NN being the simplest classifier with instant training (as it does not require learning any parameters), surprisingly, it outperforms other more complex and challenging-to-train classifiers. Nevertheless, the achieved accuracies are not considered satisfactory. The features used are too basic and don't effectively describe the image content, which is why we even expected a low accuracy. Since the accuracy on testing set is really low, confusion matrices (that could be seen in notebook), show significant deviations from the diagonal, no specific missmatch-patterns could be observed that could be further discussed.

Table 1: classifier's accuracy (features: RGB color histograms)

classifier	k-NN		SVM		MLP	
	train set	test set	train set	test set	train set	test set
accuracy	0.5625	0.1038	0.2197	0.1017	0.8040	0.0994

A more advanced feature extraction method using histograms of visual words was implemented. To begin, we created vocabularies with a specified number of clusters for K-means using the "generate_vocabulary" function. We opted to construct dictionaries with 10 and 50 centroids, representing the main patterns found in the training images. This allowed us to observe how this parameter affected the subsequent classification. Subsequently, the "histogram_creation" function was utilized to generate the final features for the training images. The entire process took 46 minutes for 10 centroids and 51 minutes for 50 centroids. In this section of the exercise, we trained the same models as before. However, during training, we encountered instances where SIFT couldn't detect any keypoints in certain images from the dataset. As a result, these examples were ignored during the training process. In the testing phase, these images were labeled as "not classified." To ensure proper feature extraction and classification of testing images, the "test_model" function was used, which calls the "recognize" function. Additionally, we conducted additional testing on the training set through the "test_model_train_set" function, as there was no need to perform time-consuming feature extraction again. The Table 2 presents the classifier's accuracy for features represented by histograms of visual words. For the case of 10 clusters, we used k=17, and for 50 clusters, k=11 was utilized, as these settings resulted in higher accuracy on the testing set. In the case of SVM, c=0.8 was used for both 10 and 50 clusters. For MLP, the first case had 5 layers, while the second case had 7 layers. Additionally, we experimented with training k-NN on a combination of features represented as a single feature vector, consisting of the V histogram (providing information about brightness) and a histogram of visual words with 50 clusters. The goal was to enhance accuracy on the test set. Although there was a slight improvement, with accuracy reaching 0.195 for k=15, we didn't consider this as a significant enhancement. Therefore, we did not incorporate this combined feature approach in the other models.

Table 2: classifier's accuracy (features: histograms of visual words)

classifier	no_clusters = 10		no_clusters = 50	
	train set	test set	train set	test set
k-NN	0.2960	0.1894	0.3151	0.1863
SVM	0.1089	0.1083	0.1198	0.1183
MLP	0.2246	0.2220	0.3417	0.2504

Figure 6 demonstrates that the confusion matrices for the best model obtained in this part of the assignment do not exhibit any clear patterns.

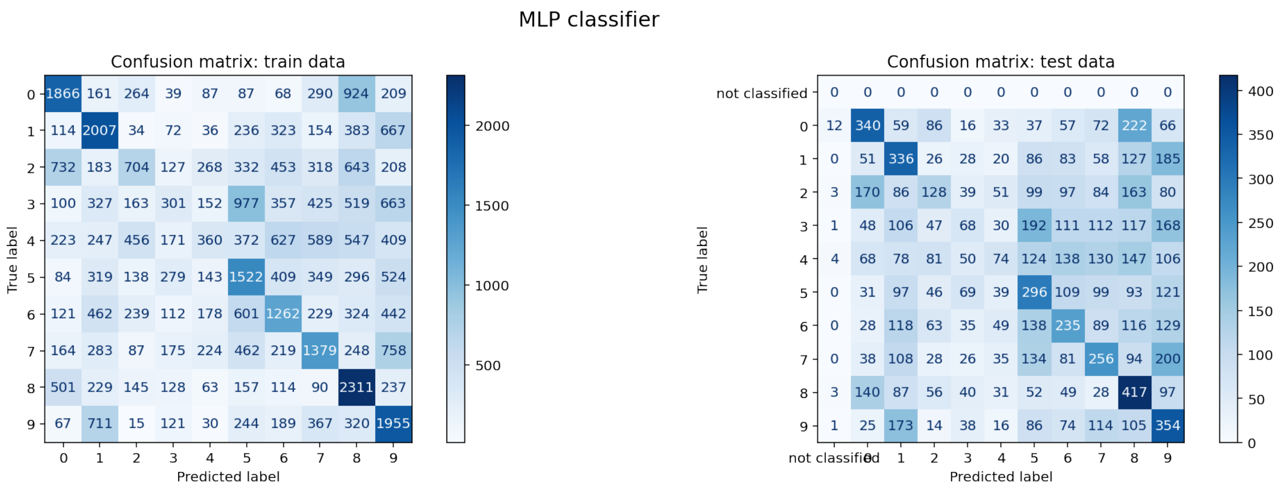


Figure 6: confusion matrices, 50 clusters, MLP classifier

Tensorflow neural network models

Table 3 shows the test accuracy of the different tensorflow models described in chapter 3.2. The results show clearly that the CNN models perform much better than the MLP models although the CNN models with a maximum of 2,633,994 trainable parameters (CNN_L) has only 12.4 % of the size of the largest tested MLP model with 21,233,010 trainable parameters.

Table 3: Accuracies of the tensorflow neural network models

Algorithm	Accuracy test set
MLP_S	47.7 %
MLP_M	48.3 %
MLP_L	47.9 %
CNN_S	69.7 %
CNN_M	70.8 %
CNN_L	72.9 %
Inception transfer learning with MLP	82.3 %

Figure 7 and figure 8 show the strengths and weaknesses of the best CNN model from scratch compared to the transfer learning model, based on Inception v3. It can be seen that both models have weaknesses in the dog/cat distinction. The transfer learning model had comparatively greater problems in distinguishing between deer and horse. 121 horse images were classified wrong as deer which is the second largest error after dogs classified as cats (160). Ad hoc, however, this can also be explained by the modest image quality.

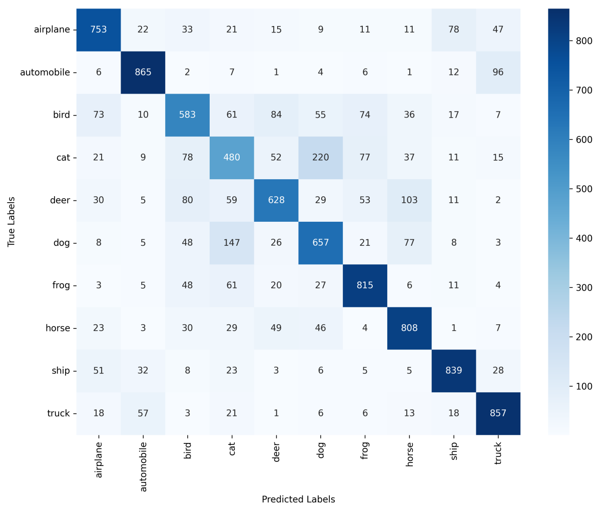


Figure 7: Confusion matrix of the large CNN model

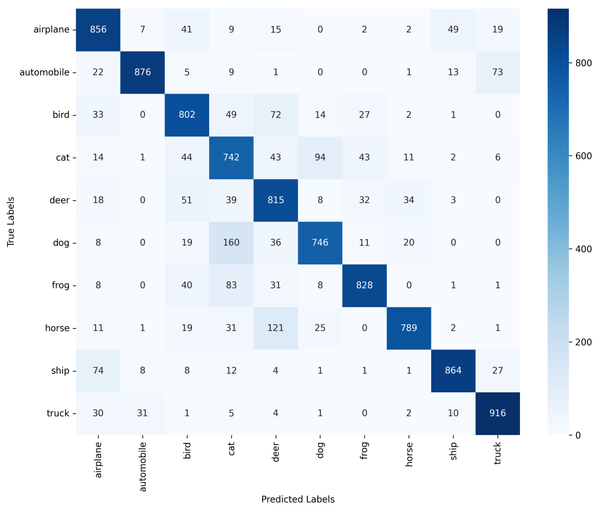


Figure 8: Confusion matrix of the transfer learneing Inception model

Figure 9 and 10 show the loss and accuracy over the train epochs of the large CNN and the Inception transfer learning model. The CNN model was stopped out after 17 epochs and at the transfer learning model, the train process was stopped out 10 epochs after the lowest validation loss at epoch 23. After this epochs, there was only improvement in training and none in validation what indicates overfitting. Without GPU performance, one epoch for the transfer learning model needed about 15 to 20 minutes calculation time, which led to the most time consuming training for this exercise with more than 12 hours calculation time.

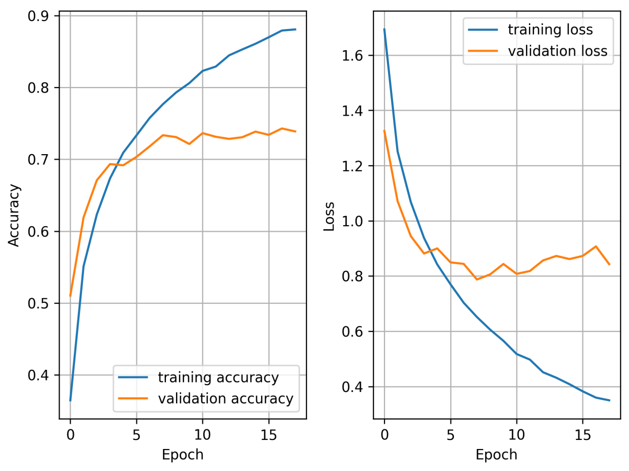


Figure 9: loss and accuracy curves from train and validation of the large CNN model

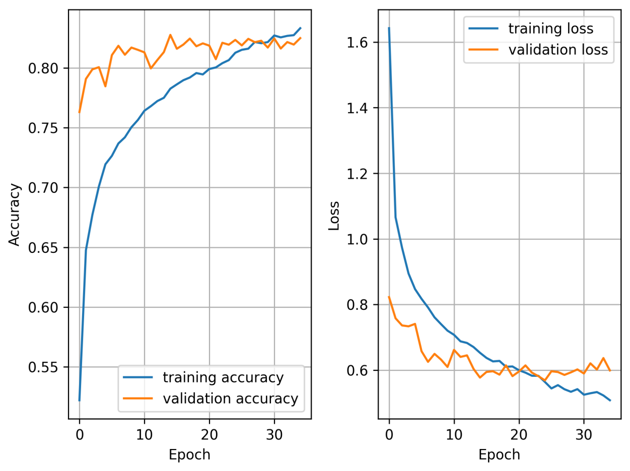


Figure 10: loss and accuracy curves from train and validation of the transfer learning model

4.2 10 Monkey Species

For the neural network models from this dataset, focus was on training rather than model selection. Nevertheless, with VGG16 a second well-known model architecture was added for transfer learning to the research.

BASE models:
Similar to the previous dataset, for simple feature representation, RGB and HSV color histograms were created. Figure 11 displays the RGB histograms for an example image from the dataset. Unlike the previous dataset, the pixels in images from this dataset were encoded as float values, which is why the histograms contain values ranging from 0 to 1. Once again, we evaluated the effectiveness of both

features using k-NN as the classifier. However, as both RGB and HSV features demonstrated similar performance, we made the decision to focus solely on RGB for further analysis.

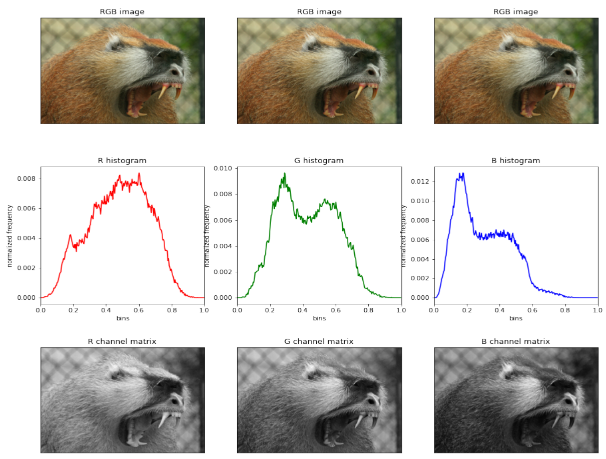


Figure 11: RGB color histograms for one dataset example

The table 4 provides information on the accuracy of different classifiers. The results are obtained using k-NN with k=7, SVM with a regularization parameter c=0.8, and MLP with 6 layers.

Table 4: classifier's accuracy (features: RGB color histograms)

classifier	k-NN		SVM		MLP	
	train set	test set	train set	test set	train set	test set
accuracy	0.46624	0.09559	0.51551	0.11765	0.63780	0.07720

For the SVM classifier, which performed the best in this phase, the analysis of the confusion matrix reveals that the training matrix is predominantly diagonal, with most misclassifications of class 9 being predicted as class 5. However, for the confusion matrix of the test data, due to the low accuracy, a clear diagonal pattern is not observed. Instead, classes 4 to 7 are frequently predicted as class 1, while classes 8 and 9 are predicted as class 2. Regarding the MLP, overfitting is evident in the model, but less complex networks did not yield satisfactory accuracy even on the training set. The accuracy on the test set is considerably lower in comparison to the accuracy on the validation set.

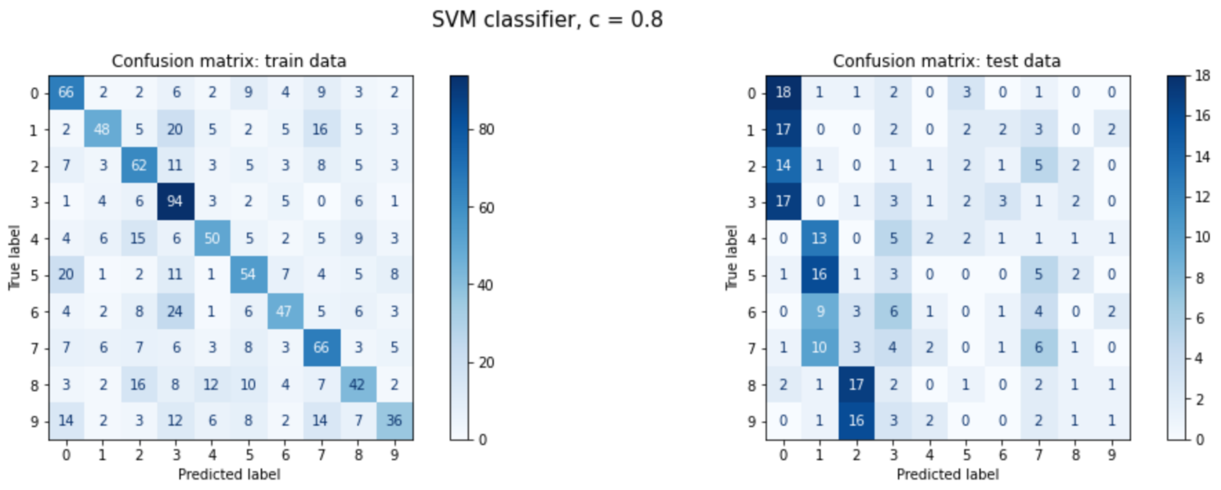


Figure 12: Confusion Matrix of the SVM model

A more sophisticated feature extraction approach using histograms of visual words, employing a vocabulary with 10 clusters was adopted. The training process for this method took around 10 minutes.

Upon plotting histograms of three images from three randomly selected classes (as illustrated in the notebook), we observed that histograms of images belonging to the same class were not exactly identical but appeared to be slightly more similar compared to the previous dataset. This promising observation gave us hope for improved classification quality in the subsequent analysis.

The Table 5 gives information on accuracy of the different classifiers. The table presents results obtained with k-NN using k=17, SVM with a regularization parameter c=0.8, and MLP composed of 5 layers. In this phase, the k-NN classifier showed the best performance. Analyzing the confusion matrix, we observed that it is predominantly diagonal, with certain misclassifications. Notably, the highest number of misclassifications occurred when class 5 was predicted as class 0. Regarding the confusion matrix of the test data, the most frequent misclassifications occurred as follows: class 5 was predicted as 6, class 1 as 4, class 7 as 0, class 6 as 3, class 0 as 6...

Table 5: classifier sccuracy (features: histograms of visual words)

no_clusters = 10		
classifier	train set	test set
k-NN	0.3714	0.3272
SVM	0.1496	0.1544
MLP	0.1201	0.1103

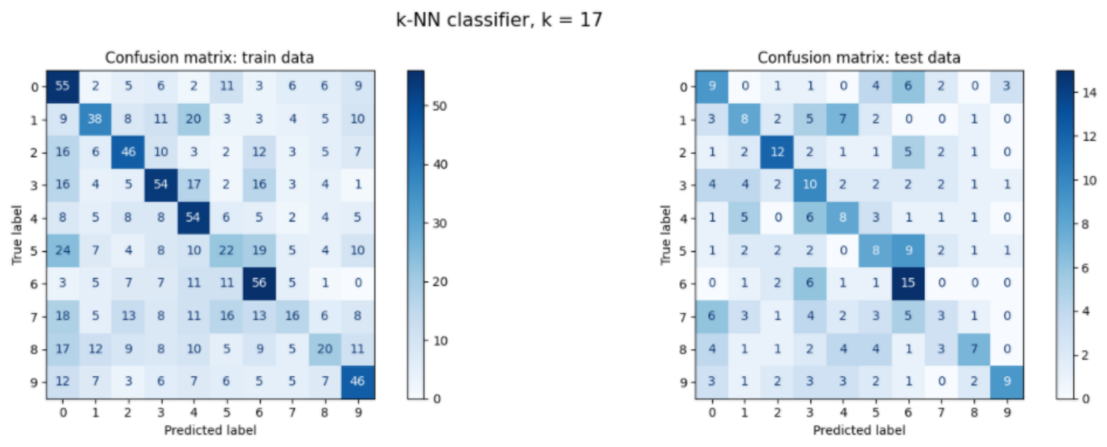


Figure 13: Confusion Matrix of the k-NN model

CNN models:
As described earlier, the main focus of this experiment was not model selection, but the impact of data augmentation on a basic CNN structure with 103,297,290 trainable parameters from an convolution block with 3 convolution layers [64, 128 256] and a prediction block with a 3-layer MLP before the output block. The results in table 7 and the training curves in figures 14 and 15 show large differences in training and accuracy of the two models. The model, which was trained without data augmentation, was aborted after 18 epochs due to the early stopping callback. This led to an accuracy of 40.81 % with the lowest loss weights from epoch 8. The model trained with data augmentation training data was trained for 85 epochs and reached an accuracy of 72.1 (weights from epoch 75) %. Although the early stop without data augmentation could also have occurred by chance, the results show the great advantages offered by this technique.

Table 6: Accuracies of the CNN models

Algorithm	Accuracy test set
CNN_L	40.8 %
CNN_L with data augmentation	72.1 %

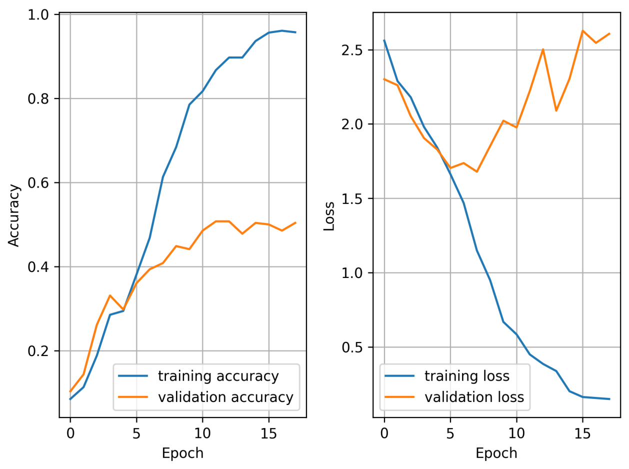


Figure 14: loss and accuracy curves from train and validation of the large CNN model

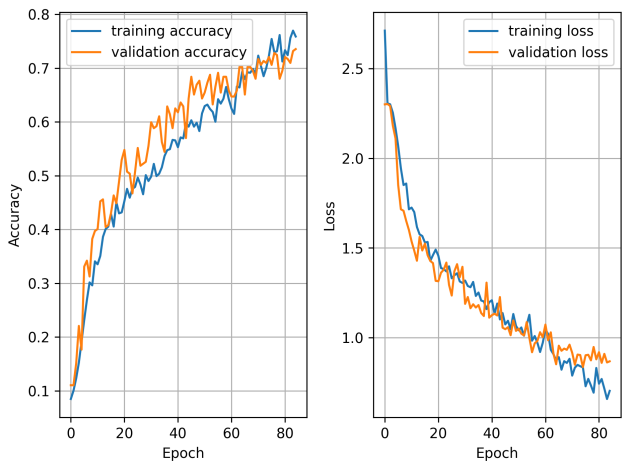


Figure 15: loss and accuracy curves from train and validation of the CNN Model with data augmentation

Transfer learning model:
In this section, the impact of different prediction blocks of transfer learning models was investigated. Base for each model was Inception-v3 with weights form 'Imagenet' dataset. Based on this, prediction blocks based on MLP or Global Average Pooling were added as described in chapter 3.2. For each of two models, only the prediction block was trained. In the other two models, the last 5 layers of the base model were then fine-tuned together with the prediction block after 15 epochs.

Table 7: Accuracies of the CNN models

Algorithm	Accuracy test set
inception_MLP1	96.7 %
inception_MLP2	86.0 %
inception_GAP1	98.2 %
inception_GAP2	97.8 %
VGG16_GAP1	68.0 %

The incredibly high accuracy of 98.2 % can be explained by the weights of the pre-trained base model. The used 'Imagenet' dataset already includes different monkey species. In this case, the transfer learning only led to a further specialization of the model.

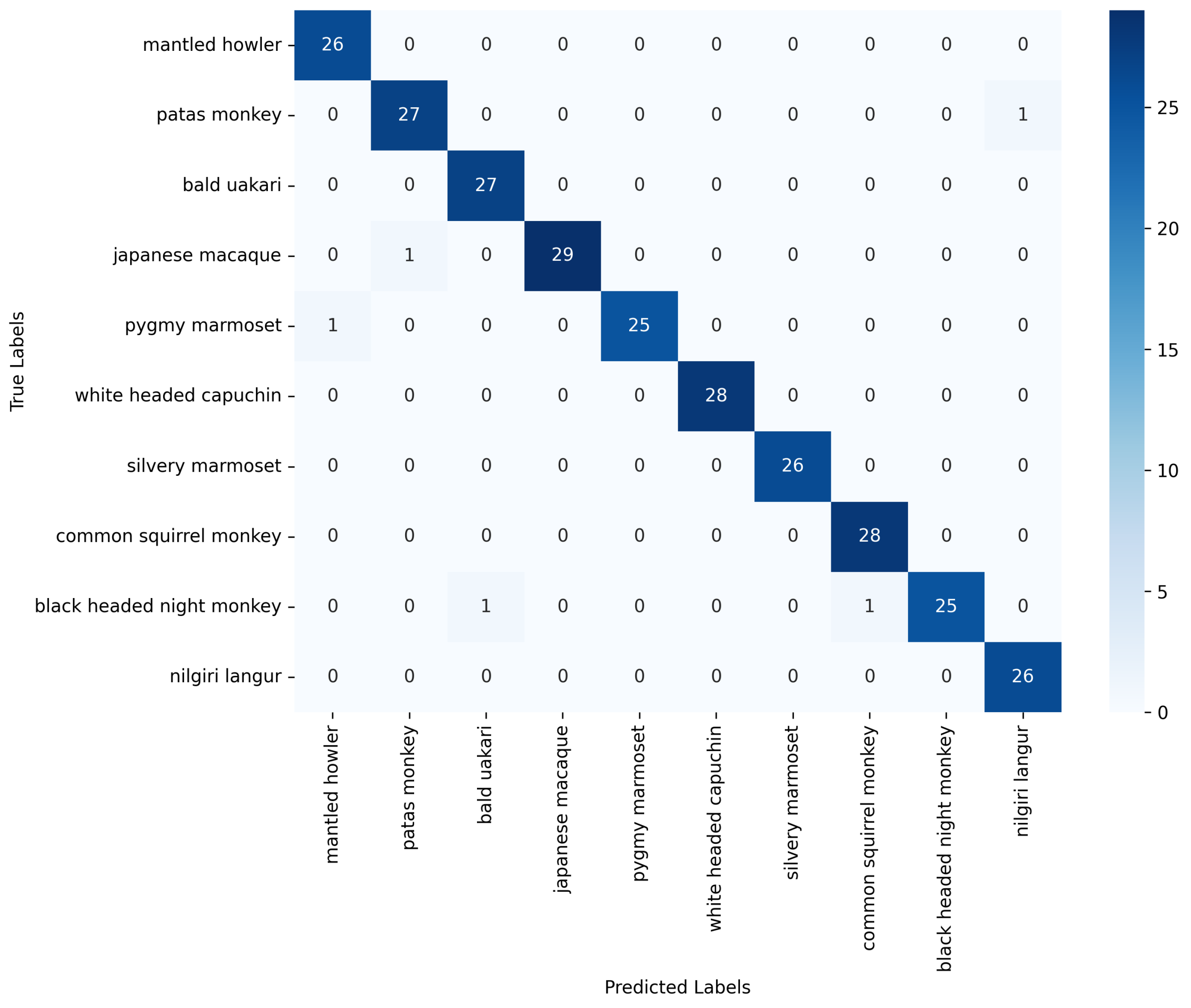


Figure 16: Confusion Matrix of the transfer learning model with Interception-v3 as base model and weights from the 'Imagenet' dataset and Global Average Pooling Layer

In transfer learning, those models that were not fine-tuned performed better than those with additional training run. This could be explained by the hyperparameters 'learning rate' and 'momentum', which were not optimized by hyperparameter tuning, but were taken from a comparable example in the literature[3]. At least within 50 epochs, the VGG16 architecture could not achieve better results than the comparatively simple CNN described above, which was also trained with data augmentation. Even if the training was stopped early (max 50 epochs), the convergence time was comparatively long. Due to the excellent results with Imagenet, no need was seen to continue the training with VGG16.

[3]Géron A., 2023, Hands-On Machine learning with Scikit-Learn, Keras and TensorFlow. ISBN: 978-1-098-12597-4

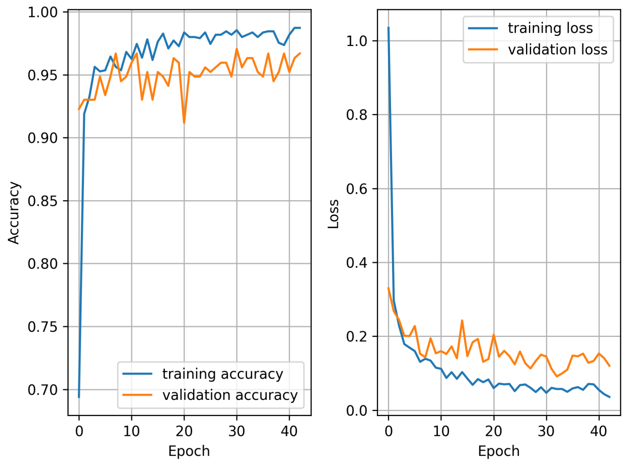


Figure 17: Loss and accuracy curves from train and validation of the transfer learning model with VGG16 as base model and Global Average Pooling Layer

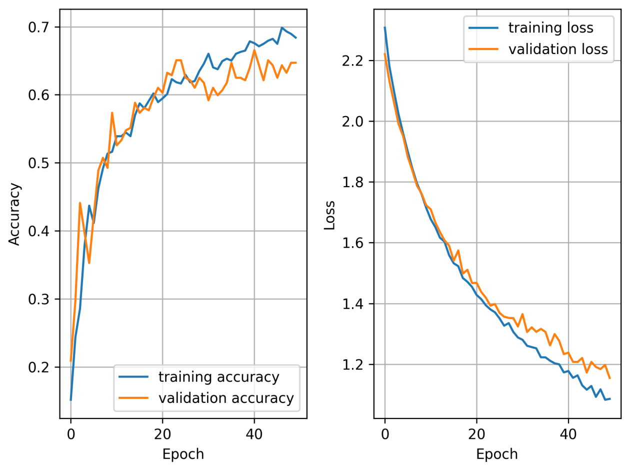


Figure 18: Loss and accuracy curves from train and validation of the transfer learning model with VGG16 as base model and Global Average Pooling Layer

5 Conclusio and Discussion

In the first part of the project, traditional feature extraction methods such as color histograms based on RGB and HSV were used as input to multiple classifiers, namely kNN, SVM and MLP. Moreover, a more advanced feature extraction technique such as SIFT was used to detect and describe key points in our images. These techniques were useful to get to know our data better and interesting to experiment with. However, it was clear from the results that they were extremely lacking in performance, especially when compared with more advanced methods such as CNN, which have direct access to raw image pixels and can automatically learn the features during training process.

Regarding the architecture of CNNs, the number of convolutions is limited due to the input size because of the size reduction while pooling. Tests with deeper CNNs and the CIFAR-10 dataset led to an non-converging results because of the size of the size of the feature maps in the last convolution layer.

In terms of the "history" output from the tensorflow training process and the loss and accuracy curves generated from it, the picture between train and valid is deceptive. While train error is computed using a running mean during the epoch, the validation error is computed at the end of each epoch. For an accurate comparison, the training curve should therefore be shifted to the right by half an epoch

Many transfer learning models are freely available and easily adaptable. Using the 10 Monkey Faces dataset, this work demonstrated that with comparatively little programming and computational effort, an accuracy greater than 98 % could be achieved in classifying 10 different monkey species within less than 50 training epochs. 4 out of 272 test images were incorrectly classified. The fact that the process was presumably simplified with the pre-trained weights of the "Imagenet" dataset was a coincidence, but shows the necessity of clever model and weight selection.