FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Extended Image Classification

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software und Information Engineering

eingereicht von

Dipl.-Ing. Horst Gruber, MBA
Matrikelnummer 08330864

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Mag. rer. soc. oec. Stefan Ohrhallinger, PhD

Wien, 16. August 2019

Horst Gruber	Michael Wimmer

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Extended Image Classification

Iterative k-Means Image Classification

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software and Information Engineering

by

Dipl.-Ing. Horst Gruber, MBA
Registration Number 08330864

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Mag. rer. soc. oec. Stefan Ohrhallinger, PhD

Vienna, 16th August, 2019

Horst Gruber	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Horst Gruber, MBA

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 16. August 2019

Horst Gruber

Danksagung

Ich möchte mich an dieser Stelle bei meinen Betreuern Associate Prof. Dr.techn. Michael Wimmer und PhD. Stefan Ohrhallinger herzlich für ihre großartige Unterstützung und ihrem intensiven fachlichen Feedback während meiner Arbeit bedanken.

Acknowledgements

I would like to express my gratitude to my supervisors Associate Prof. Dr.techn. Michael Wimmer und PhD. Stefan Ohrhallinger for their great support and extensive feedback during my work on this thesis.

Kurzfassung

In dieser Thesis entwickeln wir ein Modell zur Bildklassifizierung mit einer sich während der Trainingsphase steigernden Klassifizierungsleistung. Das Modell verwendet für die Gewinnung der Bildmerkmale ein Convolutional Neuronales Netzwerk (CNN) und für die Clusterbildung einen k-Means Algorithmus. Die Leistungsoptimierung wird mittels optimierter Gewichtungsfaktoren erreicht, welche auf die Bildmerkmale angewandt werden. Die Optimierung der Gewichtungsfaktoren wird während einer Trainingsphase iterativ durchgeführt. Das Maß für die Anpassung der Gewichtungsfaktoren in einem Trainingsschritt steht in Relation zum Clustering-Beitrag eines im Trainingsschritt neu hinzugefügten Bildes.

Wir sehen als einen Vorteil unseres Konzeptes, dass keine Eingriffe in die inneren Strukturen der verwendeten Algorithmen für die Bildmerkmalgewinnung und deren Clustering notwendig sind, und daher bereits trainierte Modelle oder closed-source Modelle verwendet werden können. Einen weiteren Vorteil im Vergleich zur Batch-Struktur des Trainingsprozesses bei CNNs sehen wir in der Transparenz der schrittweisen Leistungsentwicklung unseres Modells während der Trainingsphase für jedes neu hinzukommende Bild. Dies ermöglicht eine flexible Kontrolle der Trainingsphase durch den Benutzer. Einen weiteren Vorteil sehen wir in der geringen Anzahl der zu optimierenden Parameter, was zu einer Reduktion der Bearbeitungszeit führt. Ein weiterer Vorteil ist die Klassifizierungsleistung unseres Modells, welche die Leistung des Referenzmodells ohne Bildmerkmalsoptimierung übertrifft.

Im Zuge unserer Arbeit haben wir eine Python Applikation entwickelt, in welcher unser Model implementiert wurde und die eine benutzerfreundliche Oberfläche bietet. Die Applikation ermöglicht eine einfache Konfiguration von Testfällen und bietet umfangreiche Auswertungsmöglichkeiten für jeden Trainingsschritt in grafischer und tabellarischer Form. Wir sehen diese Applikation als Ausgangspunkt für weitere Arbeiten zu diesem Thema.

Abstract

In this thesis, we developed an image classification model with improving classification performance over a training phase. The model is using a pre-trained convolutional neuronal network (CNN) for feature extraction and a k-means algorithm for clustering. Performance optimization is realized by optimized weight factors for the extracted feature values. The optimization of the weight factors is calculated iteratively during a training phase. The measure of the weight factor adoption in a training step is related to the ground-truth dependent clustering contribution of the newly added image feature.

We see as an advantage of our approach that the optimization requires no internal changes of the applied feature extraction and clustering algorithms, hence pre-trained models or closed-source implementations can be used. As a further advantage, we see the step-wise transparency of the performance development during the training phase for each newly added image as opposed to batch-based training for CNNs. This enables dynamic control of the training phase by the user. Another advantage is the small number of parameters to be optimized, which results in reduced processing time. A further advantage is the classification performance of our model that outperforms the reference model without feature weight optimization.

In the course of our work, we developed a Python application that implements our model and provides a user-friendly interface. It allows easy set-up of test cases and provides graphics and tables for a comprehensive evaluation on process steps level. We consider this application as a starting point for future work.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Image classification is one of the most important parts of digital image analysis. It is a process in computer vision for classifying an image according to its visual content. The process of image classification can be divided into feature extraction and a feature clustering phase. The results of the feature extraction are the image features that are used by the clustering process to classify the image. The image classification approaches can be broken down into supervised and unsupervised category, depending on the interaction between the user and the application during classification. Image classification has a broad application field, as there are for example automated image organization, image and face recognition on social networks, environment detection in autonomous driving concepts, medical diagnosis, and security applications.

There are many models for image classification with different approaches. The latest trend shows models with joint optimization of the feature extraction and the clustering algorithm. As an example, Yang et al. [YFSH17] propose a joint optimization approach applied to a model that combines a convolutional neuronal network (CNN) feature extraction and K-means clustering. Common to most of the models is a single learning phase, during which the model performance is optimized and integrated closed model architecture.

In contrast to a single learning phase approach, we wanted to address in our work the need for a user-controlled transparent training phase. The model should enable user interaction in any state of the training phase. Hence the model should provide performance and other relevant information in any process state to enable the user to control and adapt a currently processed training.

In the comparison with the common model approaches which are mainly highly sophisticated integrated solutions, like for example Yang et al. [XGF16], proposing deeply

embedded clustering (DEC), we want to reach modularity with our model architecture. It means that our model should be capable of using different available ready solutions for feature extraction and clustering without the necessity of intervention into the internal structure of the solution. This would make our model flexible for applying and combining different and arbitrary algorithms. Additionally, the model would be open for future third-party algorithm improvements and developments.

A further challenging task we want to address in our work is to keep the processing complexity low. This should prevent an unpractically long processing time. Finally, we have the goal to develop a model that shows a performance advantage in comparison to other state-of-the-art models.

1.2 Related Work

As mentioned above, there are different approaches to image classification. The most common state-of-the-art approaches for supervised image classification use deep convolutional neuronal networks (CNN). There were different approaches to specialized image classification model designs over the last decades. The latest development shows a trend to join the optimization of feature extraction and clustering, as follows:

Chang et al. [CWM+17] state that most of the existing methods ignore the conjunction between feature learning and clustering. They assume that the relationship between pairwise images is binary, hence their model approach is identifying the image pairs which should belong to the same cluster. The issue of unknown ground-truth similarities is solved by an alternating iterative adaptive learning algorithm which uses labeled samples for the CNN training. Hsu et al. [HL17] propose an iterative learning model that randomly selects sample images from the ImageNet[1] dataset and extracts their features, using a pre-trained CNN. These feature sets are initially assigned to cluster centroids. Then a mini-batch k-means clustering algorithm assigns cluster labels to their input samples. The samples are randomly selected from the set of all images until all images have been processed. The model simultaneously updates the parameters of the CNN and the centroids of the image clusters, based on stochastic gradient descent. Yang et al. [YFSH17] also propose a joint optimization approach applied to a model that combines CNN feature extraction and k-means clustering. Fard et al. [FTG18] propose the use of a deep k-means model for joint optimization of learning representations and clustering through stochastic gradient descent updates, representation, and k-means clustering losses. Yang et al. [XGF16] propose deeply embedded clustering (DEC), an algorithm that clusters a set of data points in a jointly optimized feature space, without ground-truth cluster membership labels.

Guérin et al. [GGTN17] provide a comprehensive evaluation of different combinations of CNNs for feature extraction and classic clustering algorithms as classification pipelines.

[1] ImageNet is a large visual database designed for use in visual object recognition software research. It contains more than 14 million hand-annotated images in more than 20,000 categories. http://www.image-net.org/

The results of their evaluation show that such combinations can compete with more sophisticated and tuned clustering methods.

The approach we propose in this thesis is not following the current trend of a joint optimization, instead, we are optimizing weight factors that we apply to extracted features for use in the clustering algorithm. This *interface* permits to plug-in modules for both feature extraction and clustering without altering their respective internals.

1.3 Scope of this Thesis

Aiming at a modular model architecture for using arbitrary feature extraction and clustering solutions required us to define the readily available solutions for feature extraction and clustering that we would use in our model for development and testing. The selected models had to meet the requirements of reliable and proven functionality, as well as an easy use in our Python[2] development and testing environment.

In our model architecture design, we had to cover the requirement for a step-wise transparent evaluation of performance improvement during training. For meeting this requirement, we defined the feature weighting at the interface of the feature extraction module and the clustering module as a proper architectural model solution.

A challenging task was to develop a stable algorithm for the feature weight optimization that would finally lead to a model performance advantage compared to a defined benchmark. We had to find a metric and an associated calculation algorithm for a step-wise feature weight optimization. The optimization algorithm had to consider the already learned weights from previous training steps and optimize them, based on the classification contribution delivered by the current training step. In that context, we had to avoid that the optimized feature weights converge to infinite values in case of a continuous single direction classification contribution in each training step. This would raise the problems of untraceable optimization and model instability. We solved this by defining a normalization of the measured values for feature weight adoption and the feature weight values in conjunction with the overall calculation algorithm.

After we had developed the calculation algorithm of our model, we faced the challenge to select the properly hidden layer for feature value output from the selected VGG16 CNN that we had selected as feature extraction solution for our testings. We aimed to use the output values of the layer as feature input values to the clustering process after weighting with the optimized weight factors.

By the requirement that the feature values contain spatial contextual information that is necessary to make a classification based on that criterion, we tested our model with each of the fully connected hidden layers that provide such information from the VGG16 CNN. When using classes defined by ImageNet that the VGG16 CNN was trained on,

[2]Python is an interpreted, high-level, general-purpose programming language, created by Guido van Rossum and first released in 1991 (see https://www.python.org/

we got very good performance results close to 100% that we supposed to be biased by the pre-trained status of the VGG16 CNN. Hence we doubted the relevance of the test results for a layer selection and we decided to carry out the tests with an image set with a manually defined ground-truth which the VGG16 CNN was not trained on. Based on the results of these tests, we finally selected a hidden VGG16 CNN layer that we used in our further tests.

For showing the performance of our model, we had to define a performance benchmark. Hence we defined a reference model for comparison that allowed us an evaluation of our model approach focused on weight factor optimization.

In our tests, we aimed to evaluate the performance of our model using image test sets with different subjective ground-truth. We carried out one test on an image set with subjectively similar ground-truth and for comparison one test with subjectively diverse ground-truth.

Lastly, we aimed to evaluate the performance of our model on larger image sets than used for the VGG16 layer selection, with ground-truth that the VGG16 CNN was not trained on. These tests should avoid a possible bias of the test results from the pre-trained status of the VGG16 CNN. We generated two different manually labeled image sets, each with a ground-truth the VGG16 CNN was not trained on and carried out the related tests.

1.4 Outline

This thesis is structured in such a way that in Section 2 we cover the theory of the used VGG16 CNN feature extraction model and the used k-means clustering model. In Section 3, we explain our developed model in detail. This section also includes detailed descriptions of the implemented application user-interface for testing and evaluating, together with detailed data specification of the output interfaces (in Subsection 3.5). In Section 4, we describe the test parameters and show our results. For the correct model behavior testing, see Subsection 4.1. For the test results for the VGG16 layer selection, see Subsection 4.2. For the evaluation results of test cases with classes defined in ImageNet and pre-trained in the VGG16 CNN for subjectively similar and diverse classes, see Section 4.3. For the evaluation results of test cases with manually defined class labels that the VGG16 was not trained on. see Section 4.4. Finally, in Chapter 5, we discuss our results and what tasks are interesting for future work.

CHAPTER 2

Theory

2.1 VGG16 Convolutional Neuronal Network

For the image feature extraction, we use the VGG16 CNN model, as described by K. Simonyan and A. Zisserman from the Visual Geometry Group, from the University of Oxford in their paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" [SZ14]. The model was designed to classify images from the ImageNet

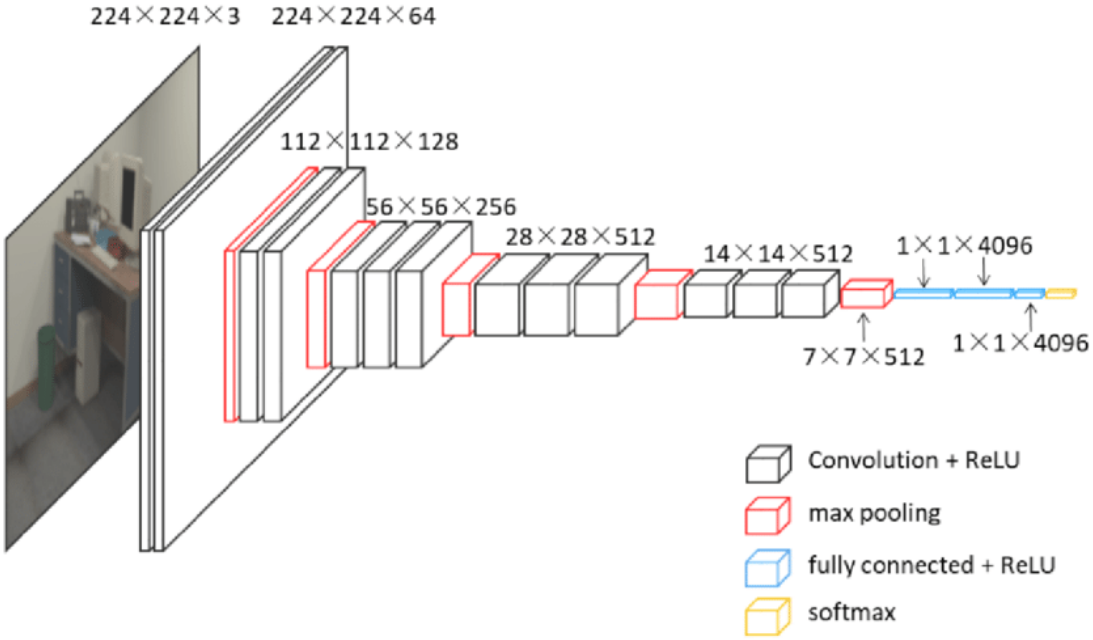


Figure 2.1: Architecture of VGG16.

database that contains over 15 million images belonging to roughly 22,000 categories. The network was trained to classify the images for 1,000 defined classes. The network uses RGB-images of size 224 x 224 x 3 as input. Rozsa et al. [RGB16] evaluated in their work a VGG16 top-1 accuracy error rate of 31.642% and a top-5 error rate of 11.556%.

The network model has 16 layers in total. The first 13 layers are convolutional layers and the following three layers are fully connected layers. A graphical representation of the architecture of the VGG16 CNN is shown in Figure 2.1[1] The VGG16 takes a 224 x 224

Layer (type)	Output Shape	Param #
input_9 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312

Table 2.1: VGG16 layers from Keras API as used in our model.

RGB image as input and pre-processes it by subtracting the mean RGB value from each

[1] source: Indoor Visual Positioning Aided by CNN-Based Image Retrieval: Training-Free, 3D Modeling-Free - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Architecture-of-VGG16_fig1_327060416 [accessed 11 Aug, 2019]

pixel. In the stack of convolutional layers, filters with the smallest possible receptive field ($3x3$) are used to capture the notion of left/right, up/done and center. The convolutional stride is fixed to 1 pixel. The spatial padding of the convolutional layer input is 1 pixel for $3x3$ convolutional layers. Max-pooling is performed over a $2x2$ pixel window, with stride 2. The stack of convolutional layers is followed by three fully connected layers. The first two have $4,096$ channels each, the third performs 1000-way ILSVRC[2] classification and thus contains $1,000$ channels (one for each class) (as described by Simonyan et al. [SZ14]).

In our model implementation, we are using the Keras API[3] VGG16 implementation. The VGG16 layers we use in our model to generate the image features for the k-means segmentation can be found in Table 2.1. The RGB images values are the input values to the *InputLayer* layer and the detected feature values we process further on are the output values of the *fc2* layer.

2.2 K-Means Clustering

The k-means method is a commonly used clustering technique that provides the advantages of speed and simplicity. It seeks to minimize the average squared distance between points within the same cluster.

An example of the algorithm process steps is shown in Figure 2.2 for two clusters. In the first step, the center points of the clusters are randomly selected from the data points. In further steps, the algorithm seeks to minimize the average squared distance between points within the same cluster.

In our model we are using the Keras API in-built k-means algorithm[4], implemented in the *KMeans*[5] class to generate the cluster centroids.

The Keras API in-built k-means algorithm is described in the paper by Arthur et al. [AV07] and is called k-means++. In their algorithm approach, they choose the centers at random from the data points, but weigh the data points according to their squared distance squared from the closest already chosen center. They analyzed the performance of their k-means++algorithm and showed that the k-means++ algorithm outperforms Lloyd's k-means algorithm [Llo82] that begins with k arbitrary *centers*, randomly chosen from the data points.

[2]The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a benchmark in object category classification and detection on hundreds of object categories and millions of images. The challenge has been run annually from 2010 to present, attracting participation from more than fifty institutions (see Russakofsy et al. [RDS+15]).

[3]Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. https://keras.io/

[4]for algorithm code see https://github.com/scikit-learn/scikit-learn/blob/1495f6924/sklearn /cluster/k_means_.py#L772

[5]The *Kmeans* class (see https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html), in sklearn.cluster package, from scikit learn, provides an in-built clustering algorithms.

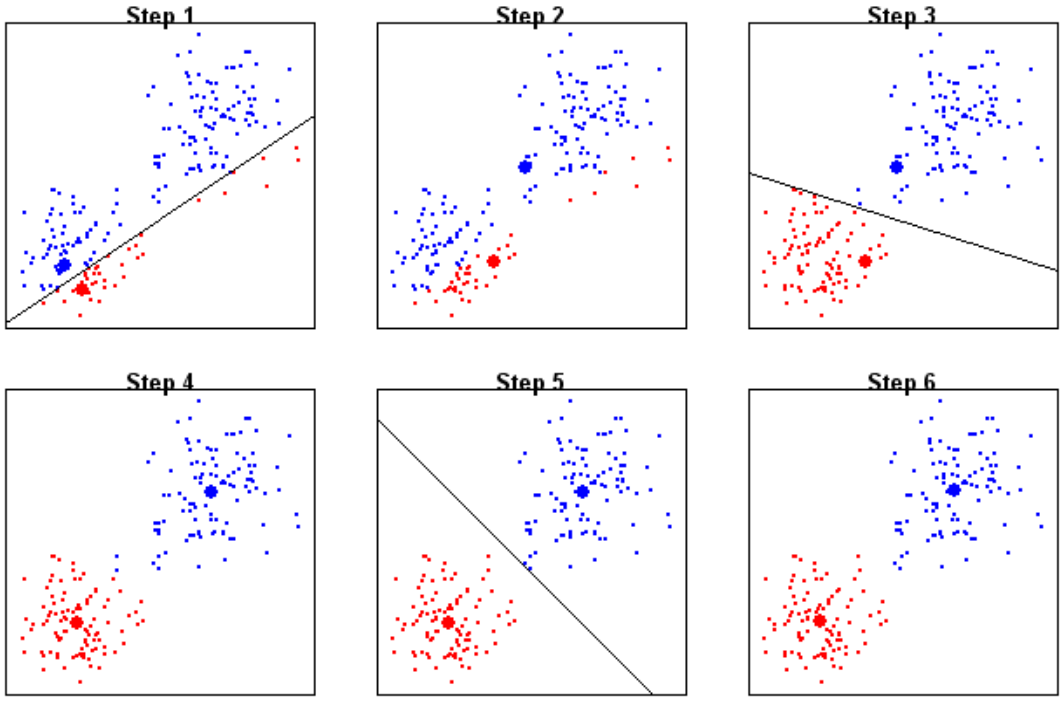


Figure 2.2: Example of k-means algorithm steps.

CHAPTER 3

Method

3.1 Goal and Process

There are various approaches regarding image classification according to objects shown in the image. This work aims to evaluate a model for image classification based on using a pre-trained VGG16 CNN for image feature extraction and the k-means algorithm for classification, such that the feature input in the k-means clustering is weighted with calculated feature weights to foster a better class separation.

A prerequisite for the further evaluation and optimization of the image classification is that all images are labeled with their ground-truth, i.e an image specific information that tells what object is shown in the image.

In comparison to a normal k-means clustering approach, we apply in our model weight factors on each image feature value before using it in the k-means clustering algorithm. We are calculating the optimized weight factors using a training phase iteratively in each training step.

The goal of using feature weights that are initialized with value 1 is to consider those features more that support a correct classification and those that are preventing a correct classification less. Consequently, the classification process should deliver better results by applying optimized weight factors.

In our training phase, we add a new image in each training step. The change to the feature weighting factors, resulting from the previous training step, is calculated on a metric how good this image is supporting a correct classification related to the result of the previous training step.

For this, we calculate an overlapping factor for each centroid of the k-means clustering of the previous training step per feature. This calculation is based on the distances of the image feature to the centroids. Thus, distances which support the classification are

assigned a positive sign, and distances which prevent such classification, are given a negative sign. The result of the calculation value per feature is normalized to [0..2]. At the end of each training iteration step, the mean value of the feature result value and the feature weight factor is calculated and set as the new feature weight factor.

As image base, we use ImageNet data sets. The k-means clustering algorithm for the images takes the features of the image as input that were detected by a CNN for that image. In our model, we use the pre-trained VGG16 CNN for feature extraction. We tested the use of the output values of these hidden layers: *flatten*, *fc1* and *fc2*. The test of the different layers showed a significant improvement of classification performance by using *fc1*-layer instead of *flatten* and *fc2*-layer instead of *fc1*. Especially in the case of newly defined classes that the VGG16 CNN was not trained on, the classification performance of k-means was significantly better when using *fc2*-layer than the *flatten* layer. Based on these test results, we decided to use the *fc2*-layer in our model.

3.2 Implementation

The initialization, training, and evaluation phases are implemented in an object-oriented Python application. For the development, we were using the Python Version 3.7 and for application development the PyCharm API version Professional 2019.1.

The application process parameters are

- the set of defined classes D that the images should be assigned to,
- the number n_I of images per class for the initialization phase before running the training phase,
- the number n_T of images per class for the training phase ,
- the number n_E of images per class for the evaluation phase,
- the order of images in the training phase according to their class relation (alternating or non-alternating),
- the name of the pre-trained VGG16 hidden layer for image feature extraction,
- an optional set F_i of image features for test purpose and
- an optional set D_m of classes that are used as ground-truth for the processed list of images

The general goals are to increase the correct class assignment percentage over the training iteration steps and the improved performance of the classification result of the evaluation phase, using the optimized weight factors. For a quantitative evaluation of the performance improvement, each training phase step result is compared to a reference

model result. The reference model is the same process but without feature weight optimization, i.e. the feature weights stay constantly 1. For a user evaluation, the application gives graphical and tabular result outputs.

The graphical evaluation output is given in the form of plots that show performance development over the training iteration steps. The plots contain a graph using weight factor optimization and a graph for the reference model with constant weight factors for comparison. Additionally, a graph in the same plot shows the standard deviation development of the weight factor differences between each step as a qualitative measure of weight factor optimization. When the standard deviation becomes constant, this is a sign that the feature weight factor optimization process has become stable.

As a final performance measure of our model with k-means clustering using optimized weight factors, in comparison to the reference model with k-means clustering without feature weighting, an evaluation phase is applied. The evaluation phase uses a set of images that were not used in the initialization and training phase before. The result of the evaluation phase is the number of correctly classified images, relative to the total number of images. The evaluation is processed for our model and the reference model. These result values are shown as constant dashed graphs in the plot.

As further graphical result output, the processed images are shown separately for the training and evaluation phase, labeled with their ground-truth and framed with a color that indicates the classification result of our model and that of the reference model.

In addition to the graphical result output, the numerical performance values in each training iteration step, as well as the evaluation phase results, are saved as *.csv* files.

For special test purposes, the application can be started with a manually defined feature set F_i for the images and/or with a set of image related defined ground-truth D_m for the images to be processed. In case the test is executed with manually defined feature lists, an additional *.csv* file is generated with detailed information to each training step.

3.3 Application Initialization Process

3.3.1 ImageNet

We are using the ImageNet database as an image source. ImageNet provides the images not by itself, but via URLs to third-party sources. The application downloads the necessary images according to the provided application parameters n_I, n_T, n_E and saves them locally for further processing. This improves the run-time of processes that use image categories that were already used by a previous process. Before downloading images, the application checks the number of already available images of selected classes, and only downloads the number of missing images.

ImageNet is not the owner of the images. Therefore, there are a great number of image-related URLs that are not valid. This circumstance is handled by the application and the user does not have to take care of it. In case that the total number of requested images

is not available, no error occurs and the process is executed only with the number of available downloaded images.

3.3.2 VGG16

For the image feature extraction, we use the pre-trained VGG16 CNN implementation from Keras API. Before processing, each image is resized to $height = 224$ and $width = 224$. In our work, we tested the result of taking the output values of three different hidden layers - the *flatten* layer with 25088 features, the first fully connected *fc1* layer with 4096 features and the second fully connected layer *fc2*, also with 4096 features. The result of the tests showed that the performance of the k-means clustering with feature weight optimization, as well as without, improved by using the *fc1* layer instead of using the *flatten* layer, and also improved by using the *fc2* layer instead of using the *fc1* layer (see Section 4.2).

3.3.3 Initialization

A schematic overview of the model process pipeline is shown in Figure3.1.

Before the training phase starts, a set of images I_p is initialized with n_I images per class in D. The weight factors are set to 1. The images are not taken from the set of training images so that the initialization phase is not reducing the defined training phase scope n_T. For each image of this initial set, the features are calculated by the VGG16 CNN.

3.4 Application Training Process

3.4.1 k-Means and Class-Cluster Relations

The training phase is an iterative process that optimizes the images feature weights in each step. The number of process steps is defined by the number of images in the queue of images to be processed. Initially, this queue contains n_T images per class in D for training in total. The process order of the image queue can be defined as sequential per class or alternating on the classes in round-robin.

At the beginning of each training phase step, k-means clustering is executed with an image set I_p. For this, each feature f_i of each image i is multiplied with the feature weight factor w_f. f_i is the feature f in the feature set F of an image i. w_f is the weight factor for feature f. As a parameter, the k-means clustering gets the number of clusters that it should separate. This number is equivalent to the number of defined classes n_c. In the first training step, I_p contains the images from the initialization phase and all w_f are set to 1. The result of the k-means clustering is a set C of n_c centroids c_i that represent the clusters.

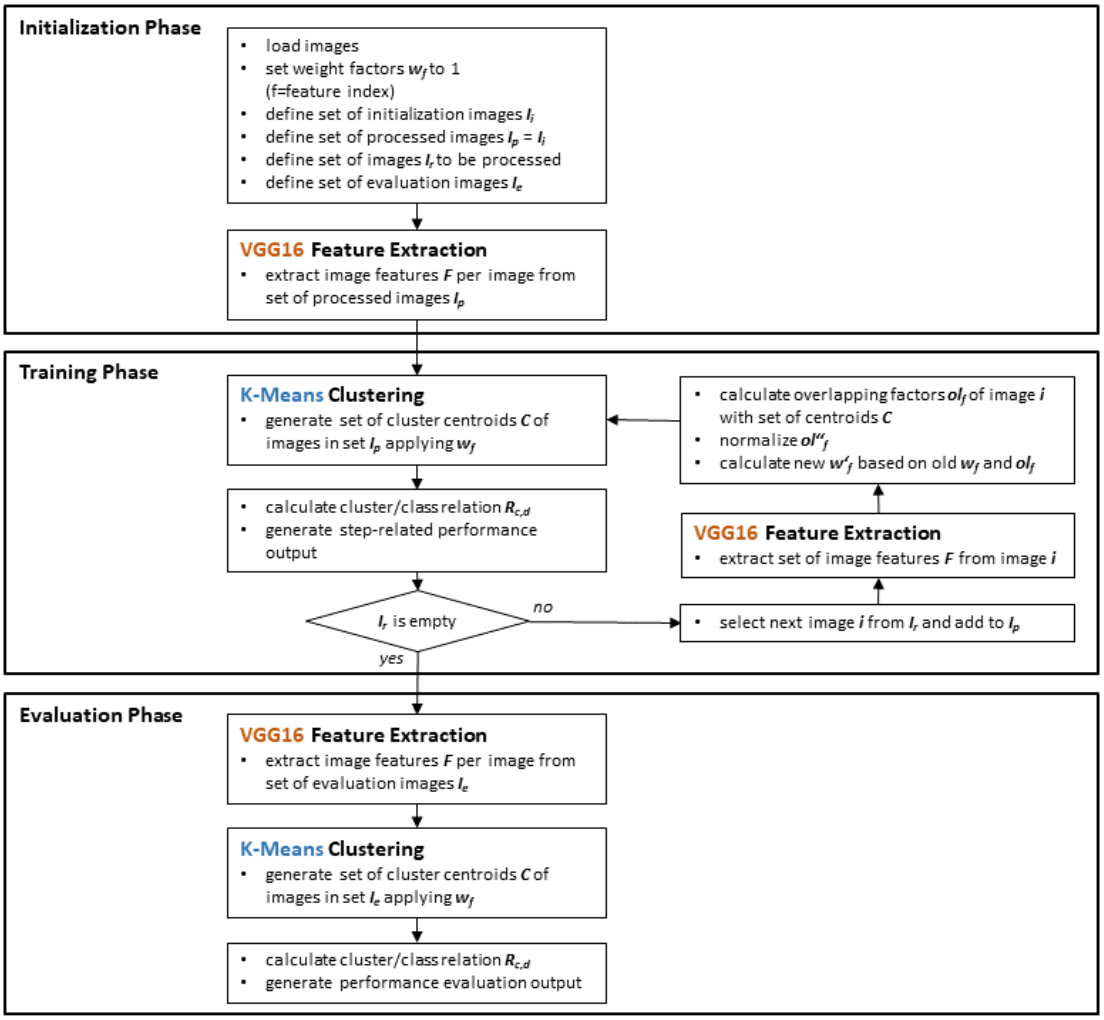


Figure 3.1: Overview of the model process pipeline.

3.4.2 Cluster-Class Assignment

After the k-means clustering run, the calculated centroids (i.e. clusters) are not yet assigned to any class. An optimal cluster-class assignment that maximizes the total number of correctly assigned images, has to be found. Thus, an optimized relation of defined classes to the k-means clusters is calculated iteratively, based on the result set C of the k-means clustering.

In this calculation process, the number of correctly assigned images for a cluster c_i of C is calculated, assuming that each cluster is representing a particular class d of D. Because each cluster can represent each class, the number of correctly assigned images is calculated for each possible cluster/class combination, which yields a square matrix. Based on this matrix, the effective cluster/class relation is iteratively computed. This

process starts with the cluster/class relation represented by the largest matrix element. After selecting the related cluster/class relation, the row and column of this element are eliminated from the matrix. This step is repeated for the reduced matrix, until all rows and columns have been eliminated. Then, all classes d of D are in a mutual exclusive relation with a cluster c_i of C and the optimized class-cluster relations $R_{c_i,d}$ are defined.

3.4.3 New Image Class Assignment

Then the first image from the queue of remaining training images I_r is assigned to the selected image i. The image is processed by the VGG16 CNN, to detect the feature values f_i of the image i. Then the centroid c_{true} of the selected image ground-truth is determined according the cluster-class assignment $R_{c_i,d}$. In addition, the feature standard deviations $\sigma_{f,c_{true}}$ are calculated for the images assigned to c_{true}.

3.4.4 Overlapping Factor Calculation

In the next training step phase, the overlapping factors o_{f,c_i} are calculated. The calculation is done separately for each feature f.

o_{f,c_i} is a measure of how much the feature value f_i of the selected image i_s is separated from a centroid c_i. The more the feature value f_i supports a separation, the more positive the raw overlapping factor o_{f,c_i} becomes. On the other hand, the more it prevents a separation, the more negative it becomes. There is a neutral separation contribution (i.e. neither making separation better nor making separation worse) of feature f in relation to centroid c_i in case of $o_{f,c_i} = 0$.

For the calculation of o_{f,c_i} the distances of the selected image feature value f_i to the feature value $f_{c_{true}}$ of centroid c_{true} and to the feature value f_{c_i} of centroid c_i are used. These distances are divided by the centroid's standard deviations $\sigma_{f,c_{true}}$ and σ_f, c_i plus 1, for getting distances relative to the feature standard deviation of the centroid.

There are three cases in the raw overlapping factor calculation:

$$\mathbf{c_i = c_{true}}: \qquad o_{f,c_i} \text{ is not calculated} \tag{3.1}$$

$$\mathbf{f_{c_{true}} < f_{c_i}}: \qquad o_{f,c_i} = \frac{f_{c_i} - f_{i_s}}{(\sigma_f, c_i + 1)} + \frac{f_{c_{true}} - f_{i_s}}{(\sigma_{f,c_{true}} + 1)} \tag{3.2}$$

$$\mathbf{f_{c_{true}} \geq f_{c_i}}: \qquad o_{f,c_i} = \frac{f_{i_s} - f_{c_i}}{(\sigma_f, c_i + 1)} + \frac{f_{i_s} - f_{c_{true}}}{(\sigma_{f,c_{true}} + 1)} \tag{3.3}$$

3.4.5 Overlapping Factor Standardization and Transformation

After calculating all overlapping factors o_{f,c_i} for a feature, the normalized overlapping factors o'_{f,c_i} results within the interval $[-1..1]$. This is achieved by dividing each o_{f,c_i} of

a feature f by the maximum of the absolute values of all factors for the feature. The normalization transforms the value 0 (=neutral separation contribution) of o_{f,c_i} to value 0 (=neutral separation contribution) of o'_{f,c_i}.

$$o'_{f,c_i} = \frac{o_{f,c_i}}{\max\limits_{c_i}(|o_{f,c_i}|)} \tag{3.4}$$

Then the total feature overlapping factor o'_f is calculated by summing all scaled o'_{f,c_i} per feature, and dividing the sum by the total number of centroids minus 1. The subtraction of 1 is required because the overlapping factor for centroid case $c_i = c_{true}$ is not calculated and therefore not included.

$$o'_f = \frac{\sum\limits_{c_i \setminus c_{true}} (|o'_{f,c_i}|)}{|C| - 1} \tag{3.5}$$

At the end of the overlapping factor calculation phase, the feature overlapping factor o'_f is transformed from the interval $[-1..+1]$ to an overlapping factor o''_f, within the interval $[0..2]$, by adding 1.

$$o''_f = o'_f + 1 \tag{3.6}$$

This transforms for example the lower bound value -1 of o'_f to 0 in o''_f, the value for no overlap 0 of o'_f to 1 in o''_f, and the upper bound 1 of o'_f to 2 in o''_f.

3.4.6 Calculation of new Feature Weight Factors

Finally, the new feature weight factors w'_f are calculated by the means of the old feature weight factors w_f and the feature overlapping factor o''_f. We calculate the mean value because the new feature weight factor w'_f should reflect the learned weight, represented by the old feature weight factor w_f from previous training steps, and the clustering contribution of the newly selected image feature f.

$$w'_f = \frac{w_f + o''_f}{2} \tag{3.7}$$

After all feature weights have been updated, the selected image is removed from the queue of remaining images for the training phase I_r and appended to the set of images already processed in the training phase I_p. Then, in case of a non-empty queue of remaining images, the process continues with the next step.

3.4.7 Reference Process

In parallel to the training phase, a reference model is executed. The reference model processes the same data by the same calculation algorithms and in the same sequence as the training phase. The only difference between training and reference model is that there is no feature weight optimization executed in the reference model. This means that the feature weighting factors for the reference k-means clustering for all features remain constantly 1.

3.5 Application Output

3.5.1 Graphical Output

The application shows the stepwise performance development during the training phases in graphical form. There are three plots (see example in Figure 3.2) that are updated continuously after each training phase step. The headline of the figure contains the total number of images used for initialization, training and evaluation phase. Furthermore, it contains information about image order and the VGG16 hidden layer output used for image features. In the first plot, the percentage of total correctly assigned images

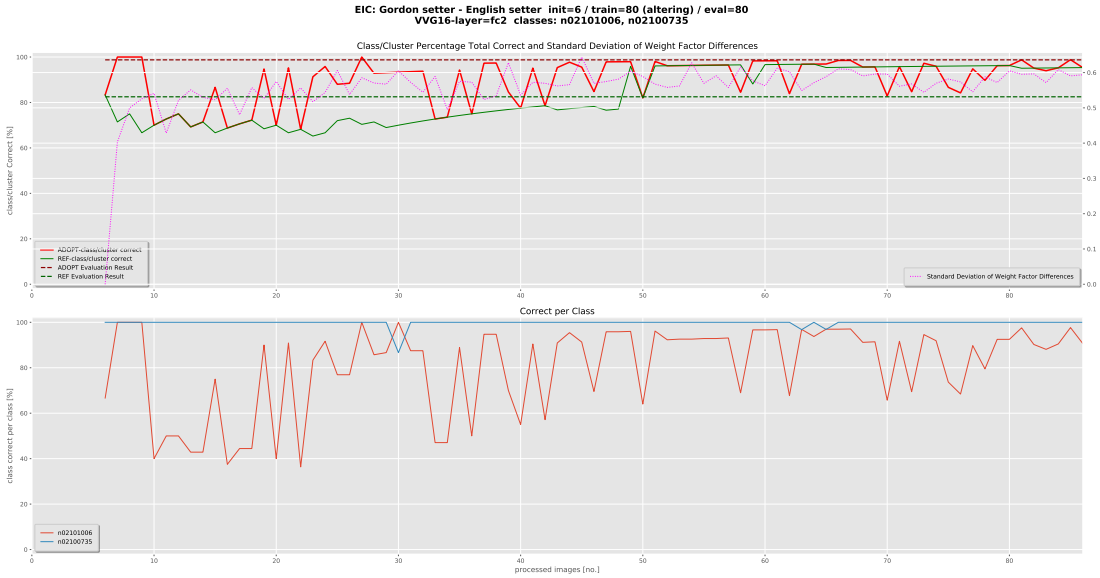


Figure 3.2: Example for the graphical output plots. (first plot: red graph=ADOPT-class/cluster correct, green graph=REF-class/cluster correct, dark red dashed graph=ADOPT evaluation result, dark green dashed graph=REF evaluation result, magenta dotted graph=standard deviation of weight factor differences; second plot: red graph=first class label, blue graph=second class label)

relative to all processed images at each training phase step are shown, separately for the process with feature weight optimization (red graph *ADOPT-class/cluster correct*) and

the reference model (green graph *REF-class/cluster correct*). Also, the development of the standard deviation of the differences of the feature weight factors w_f of the actual and previous step is shown (magenta dotted graph *standard deviation of weight factor differences*) with y-axis on the right side. The development of this value is a measure of the stability of the weight factor optimization process. For a stable process, this value should be nearly constant or the volatility and level should be as low as possible. The x-axis represents the number of processed images. For images processed at the beginning in the initialization phase, no graph values are shown. The graphs start with the first step of the training phase.

The second plot shows the percentage of correctly assigned images per class relative to the sum of all images currently assigned to that class. These graphs are only shown for the process with feature weight optimization. With this plot, the class-specific detection rates can be analyzed.

After the training phase, an evaluation phase is executed. This evaluation phase uses a new set I_e of n_E images per class defined in D that were not used in the initialization and training phase. In the evaluation phase, the image features of each image in I_e are detected by the VGG16 CNN and clustered by the k-means clustering algorithm. The k-means clustering is processed once with using the optimized feature weights from the training phase, and once with constant feature weights 1 as reference.

The results of the evaluation phase are shown as constant dashed graphs in the first plot (see Figure 3.2). They represent the evaluation result as a percentage of total correctly assigned images relative to all processed images in I_e, separately for the evaluation phase that uses optimized feature weights (dark red dashed graph *ADOPT evaluation result*) and for the reference evaluation phase that uses constant feature weights (dark green dashed graph *REF-evaluation result*).

3.5.2 Image-related Output

After the evaluation phase, two additional windows are shown by the application. In the first window (example see Figure 3.3) all images of the initialization and training phase are shown. In the second window (example see Figure 3.4) the images of the evaluation phase are shown. Images that were correctly assigned by both models (i.e. with and without k-means feature weight optimization) according to their ground-truth are shown at the top of the window, framed green. Below these, the images which only correctly assigned by the model with k-means feature weight optimization are shown, framed blue. Below these, the images, which only correctly assigned by the reference model without k-means feature weight optimization are listed and framed orange. At the bottom of the window, the images that are not correctly assigned by both models are shown and framed red. All images are textually labeled with their ground-truth.

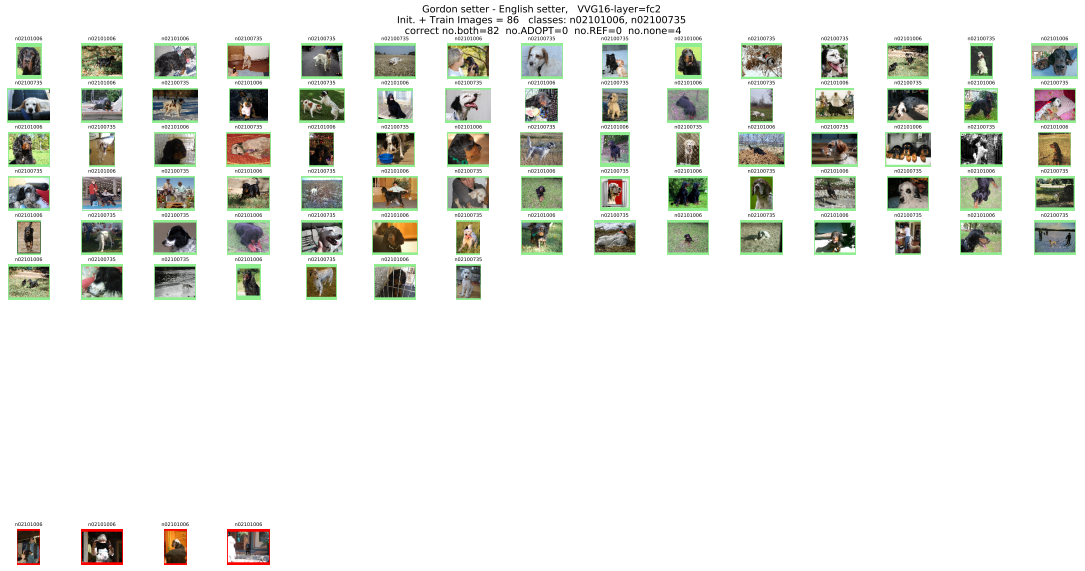


Figure 3.3: Example for the graphical output of training phase images, marked with final classification result.

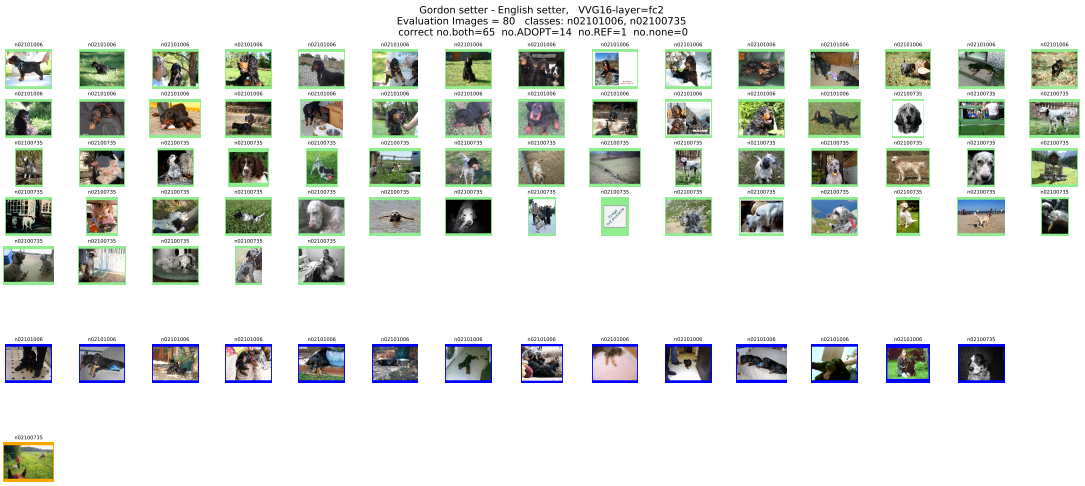


Figure 3.4: Example for the graphical output of evaluation phase images, marked with classification result.

3.5.3 Tabular Output

The results of each training phase iteration step, as well as the evaluation results, are saved as `.CSV` tables in the application subfolder `.\output`.

One file with name starting with *iter_table* contains the following data for each training phase step in columns:

- *step* ... step number (counting also initialization images as separate steps)
- *total-adapt* ... percentage of total correctly assigned images of process with k-means weight factor change
- *total-ref* ... percentage of total correctly assigned images of reference model without k-means weight factor change
- *wf-min* ... minimum of weight factors
- *wf-max* ... maximum of weight factors
- *wf-mean* ... mean value of weight factors
- *wf-std-dev* ... standard deviation of weight factors
- *std-wf-diff* ... standard deviation of weight factor differences from actual step to previous step

Additionally, the percentage of total correctly assigned images for the process with k-means weight factor optimization and the reference model are listed per class and per cluster in the following columns. These class and cluster related outputs can be activated or deactivated via application start parameters.

The second file, with a name, starts with *classes_table*, contains the following data in columns:

- *class* ... class name
- *train* ... percentage of class related total correctly assigned images of training phase with k-means weight factor change
- *train-ref* ... percentage of class related total correctly assigned images of reference training phase without k-means weight factor change
- *eval* ... percentage of class related total correctly assigned images of evaluation phase with k-means weight factor change
- *eval-ref* ... percentage of class related total correctly assigned images of reference evaluation phase without k-means weight factor change

As last row of this file, the total percentage values are added.

In case of given images feature values as application parameter, an additional `.CSV` file is generated. The name of the file starts with *test* and contains the following data columns:

- *step* ... the step number
- the feature values $if - f$ of the actual added new image (f = feature index)
- the calculated overlapping factors o''_f named $ol - f$ (f = feature index)
- the calculated new feature weight factor w_f named $wf - f$ after the training step
- *tot-corr* ... the total number of correctly assigned images by the process with feature weight optimization
- *tot-in-class* ... the total number of assigned images by the training phase with feature weight optimization
- *tot-corr-ref* ... the total number of correctly assigned images by the reference model
- *tot-in-class-ref* ... the total number of assigned images by the reference model

CHAPTER 4

Results

In case of given images feature values as application parameter, an additional `.CSV` file is generated. The name of the file starts with *test* and contains the following data columns. In our work, we carried out several tests to evaluate the correct functioning and performance of our model. We carried out three tests to evaluate the correct functioning of our model (see Section 4.1). We carried out three tests on three hidden VGG16 layers, to determine the hidden VGG16 layer that we used in our model for our further classification performance tests (see Section 4.2). Two classification performance tests were carried out with different ground-truth, each containing two classes that were defined in ImageNet and which the VGG16 was trained on (see Section 4.3). Further two classification performance tests were carried out with different ground-truth, each with two classes that were not defined in ImageNet and which the VGG16 was not trained on. In these two test cases, we labeled the images manually (see Section 4.4).

The development of the initialization and training phase is shown for the test cases in Figures 4.1, 4.3, 4.5, 4.7, 4.9, 4.11 and 4.15. The first plot of the Figures shows the percentage of total correctly classified images for each process iteration step (left y-axis). The red graph represents the performance of the process with feature weight optimization, the magenta dotted graph the standard deviation of the weight factor differences from step to step (right y-axis) and the green graph the performance of the reference model. The dashed horizontal graphs indicate the percentage of correctly classified images in the evaluation phase. The red dashed graph represents the performance of the process with feature weight optimization, the green dashed graph the performance of the reference model.

The second plot in Figures 4.1, 4.3, 4.5, 4.7, 4.9, 4.11 and 4.15 shows the correctly classified images of the process with feature weight optimization for each defined class in each iteration step of the initialization and training phase.

The classification results of the evaluation phase are shown for each test case per image in Figures 4.2, 4.4, 4.6, 4.8, 4.9, 4.14 and 4.16. Images that were classified by both models correctly are framed green, images that were only correctly classified by the process with feature weight optimization are framed blue, images that are only correctly classified by the reference model are framed orange and images that are classified by neither of the processes correctly are framed red.

4.1 Model Verification

The evaluation of test data was a very important task in our work. Based on the evaluation phase results the model behavior could be evaluated and adapted.

For each test, the set for the training phase included 10 images (step no. 5-14). Before the training phase, an initialization phase with four images was processed (step no. 1-4) without weight factor change.

4.1.1 Class Separation of Clearly Separable Test Data

This test aims to show a convergence of the weight factors development to the maximum possible weighting value, given a perfect separable set of labeled images.

To make the behavior of the model transparent, this test is executed on a small set of images. Also, to make feature weight development traceable, we were not using the results of the VGG16 image feature extraction for k-means clustering, instead we defined the image feature sets manually. Each image feature set consists of four manually defined feature values that are either 0 or 1. We defined the two separable classes A and B for the manual image labeling. Each image was labeled either with class A, with assigned feature set $[1, 0, 1, 0]$ or class B, with assigned feature set $[0, 1, 0, 1]$ (see Table 4.1). The weighting factor value interval of the model is [0..2]. Therefore, the expected weight factors should development over the training iteration steps should converge to 2

The test results are shown for each process step in Table 4.2. The table shows that the process result meets the expectations.

4.1.2 Ignore Classification Neutral Features

This test aims to show that the weight of an image feature that behaves neutral in the classification process remains constant.

The test data was similar to the above test in Section 4.1.1, but we changed the second feature value in each feature set to 2 (see Table 4.3). This means that the second feature value is constant and is not contributing to the class separation. The weighting factor value interval of the model is [0..2]. Therefore, we expected that the weight factor development over the training iteration steps for the weight factors *wf-0*, *wf-2* and *wf-3* converges to 2 and the second weight factor *wf-1* constantly remains 1.

img.no.	class	feature set
1	A	[1, 0, 1, 0]
2	A	[1, 0, 1, 0]
3	B	[0, 1, 0, 1]
4	B	[0, 1, 0, 1]
5	A	[1, 0, 1, 0]
6	B	[0, 1, 0, 1]
7	A	[1, 0, 1, 0]
8	B	[0, 1, 0, 1]
9	A	[1, 0, 1, 0]
10	B	[0, 1, 0, 1]
11	A	[1, 0, 1, 0]
12	B	[0, 1, 0, 1]
13	A	[1, 0, 1, 0]
14	B	[0, 1, 0, 1]

Table 4.1: Clearly separable test data.

step	if-0	if-1	if-2	if-3	ol-0	ol-1	ol-2	ol-3	wf-0	wf-1	wf-2	wf-3	tot-corr
1	1	0	1	0	1	1	1	1	1	1	1	1	
2	1	0	1	0	1	1	1	1	1	1	1	1	
3	0	1	0	1	1	1	1	1	1	1	1	1	
4	0	1	0	1	1	1	1	1	1	1	1	1	4
5	1	0	1	0	2	2	2	2	1.5	1.5	1.5	1.5	5
6	0	1	0	1	2	2	2	2	1.75	1.75	1.75	1.75	6
7	1	0	1	0	2	2	2	2	1.875	1.875	1.875	1.875	7
8	0	1	0	1	2	2	2	2	1.938	1.938	1.938	1.938	8
9	1	0	1	0	2	2	2	2	1.969	1.969	1.969	1.969	9
10	0	1	0	1	2	2	2	2	1.984	1.984	1.984	1.984	10
11	1	0	1	0	2	2	2	2	1.992	1.992	1.992	1.992	11
12	0	1	0	1	2	2	2	2	1.996	1.996	1.996	1.996	12
13	1	0	1	0	2	2	2	2	1.998	1.998	1.998	1.998	13
14	0	1	0	1	2	2	2	2	1.999	1.999	1.999	1.999	14

Table 4.2: Test iteration steps results of clearly separable images (if=image feature 1 to 4, ol=overlapping factor 1 to 4, wf=weight factor 1 to 4, tot-corr=total correctly assigned images).

The test results are shown for each process step in Table 4.4. The table shows that the process result meets the expectations and that the weight factor *wf-1* for the second feature constantly remains 1.

img.no.	class	featur set
1	A	[1, 2, 1, 0]
2	A	[1, 2, 1, 0]
3	B	[0, 2, 0, 1]
4	B	[0, 2, 0, 1]
5	A	[1, 2, 1, 0]
6	B	[0, 2, 0, 1]
7	A	[1, 2, 1, 0]
8	B	[0, 2, 0, 1]
9	A	[1, 2, 1, 0]
10	B	[0, 2, 0, 1]
11	A	[1, 2, 1, 0]
12	B	[0, 2, 0, 1]
13	A	[1, 2, 1, 0]
14	B	[0, 2, 0, 1]
15	A	[1, 2, 1, 0]
16	A	[1, 2, 1, 0]
17	B	[0, 2, 0, 1]
18	B	[0, 2, 0, 1]

Table 4.3: Test data with classification-neutral feature 2.

4.1.3 Filter Classification-Opposing Features from Test Data

This test aims to show the filtering of a particular feature in case it is opposing a classification by all other perfectly separable features.

The test data was similar to the above test in Section 4.1.1, but we changed the second feature value so that an image in the process sequence has the opposite value than the last image with the same ground-truth (see Table 4.5).

The weighting factor value interval of the model is [0..2]. Therefore, we expected that the weight factor development over the training iteration steps for the weight factors *wf-0*, *wf-2* and *wf-3* converges to 2 and the second weight factor *wf-1* decreases.

The test results are shown for each process step in Table 4.6. The table shows that the process result meets the expectations that the weight factor *wf-1* for the second feature is decreasing in comparison to the other weight factors. In step seven and 11 the overlapping factor *ol-1* is 2, which means perfect separation contribution. This is because the test data represents not a constantly opposing second feature in each training step relative to the already processed images.

step	if-0	if-1	if-2	if-3	ol-0	ol-1	ol-2	ol-3	wf-0	wf-1	wf-2	wf-3	tot-corr
1	1	2	1	0	1	1	1	1	1	1	1	1	
2	1	2	1	0	1	1	1	1	1	1	1	1	
3	0	2	0	1	1	1	1	1	1	1	1	1	
4	0	2	0	1	1	1	1	1	1	1	1	1	4
5	1	2	1	0	2	1	2	2	1.5	1	1.5	1.5	5
6	0	2	0	1	2	1	2	2	1.75	1	1.75	1.75	6
7	1	2	1	0	2	1	2	2	1.875	1	1.875	1.875	7
8	0	2	0	1	2	1	2	2	1.938	1	1.938	1.938	8
9	1	2	1	0	2	1	2	2	1.969	1	1.969	1.969	9
10	0	2	0	1	2	1	2	2	1.984	1	1.984	1.984	10
11	1	2	1	0	2	1	2	2	1.992	1	1.992	1.992	11
12	0	2	0	1	2	1	2	2	1.996	1	1.996	1.996	12
13	1	2	1	0	2	1	2	2	1.998	1	1.998	1.998	13
14	0	2	0	1	2	1	2	2	1.999	1	1.999	1.999	14

Table 4.4: Test iteration steps results of separable images with second feature neutral (if=image feature, ol=overlapping factor, wf=weight factor, tot-corr=total correctly assigned images).

img.no.	class	feature set
1	A	[1, 0, 1, 0]
2	A	[1, 1, 1, 0]
3	B	[0, 0, 0, 1]
4	B	[0, 1, 0, 1]
5	A	[1, 0, 1, 0]
6	B	[0, 0, 0, 1]
7	A	[1, 1, 1, 0]
8	B	[0, 1, 0, 1]
9	A	[1, 0, 1, 0]
10	B	[0, 0, 0, 1]
11	A	[1, 1, 1, 0]
12	B	[0, 1, 0, 1]
13	A	[1, 0, 1, 0]
14	B	[0, 0, 0, 1]

Table 4.5: Test data with opposing feature 2.

step	if-0	if-1	if-2	if-3	ol-0	ol-1	ol-2	ol-3	wf-0	wf-1	wf-2	wf-3	tot-corr
1	1	0	1	0	1	1	1	1	1	1	1	1	
2	1	1	1	0	1	1	1	1	1	1	1	1	
3	0	0	0	1	1	1	1	1	1	1	1	1	
4	0	1	0	1	1	1	1	1	1	1	1	1	4
5	1	0	1	0	2	0	2	2	1.5	0.5	1.5	1.5	5
6	0	0	0	1	2	0	2	2	1.75	0.25	1.75	1.75	6
7	1	1	1	0	2	2	2	2	1.875	1.125	1.875	1.875	7
8	0	1	0	1	2	0	2	2	1.938	0.563	1.938	1.938	8
9	1	0	1	0	2	0	2	2	1.969	0.281	1.969	1.969	9
10	0	0	0	1	2	0	2	2	1.984	0.141	1.984	1.984	10
11	1	1	1	0	2	2	2	2	1.992	1.07	1.992	1.992	11
12	0	1	0	1	2	0	2	2	1.996	0.535	1.996	1.996	12
13	1	0	1	0	2	0	2	2	1.998	0.268	1.998	1.998	13
14	0	0	0	1	2	0	2	2	1.999	0.134	1.999	1.999	14

Table 4.6: Test iteration steps results with second feature opposing (if=image feature, ol=overlapping factor, wf=weight factor, tot-corr=total correctly assigned images).

4.2 Evaluations of VGG16 Output Layers Usability

In our development process, we had to choose the VGG16 hidden layer that we use as an output layer for the image features in our further work.

We considered as possible hidden layers the *flatten*, the *fc1* and the *fc2* layers (see Section 2.1). These layers are the fully connected layers of the VGG16 CNN, hence their features contain context information which is required for spatial context related image classification. We tested the performance of our model with ImageNet classes, which led to very good results using *fc2* layer. We decided to make additional evaluations, using image ground-truths that are not defined in the ImageNet database. We considered that this approach delivers results that are less biased by the VGG16 CNN than those which were trained on the ImageNet classes.

The following VGG16 layer performance tests were executed with a manually labeled image set from ImageNet, showing missiles, with labels *Starting* and *Non Starting*. For a detailed description, on how we processed evaluation cases with manually labeled images, see Section 4.4.

In each test case, we used 1 images for the initialization phase, 80 images for the training phase and 20 images for the evaluation phase from each class.

Based on the results of the following described test case evaluations, we selected the *fc2* layer output for image features in our model.

4.2.1 VGG16 Output Layer *flatten*

For this test we used the output values of the hidden VGG16 *flatten* layer as image feature values for the following k-means clustering algorithm and feature weight optimization calculations.

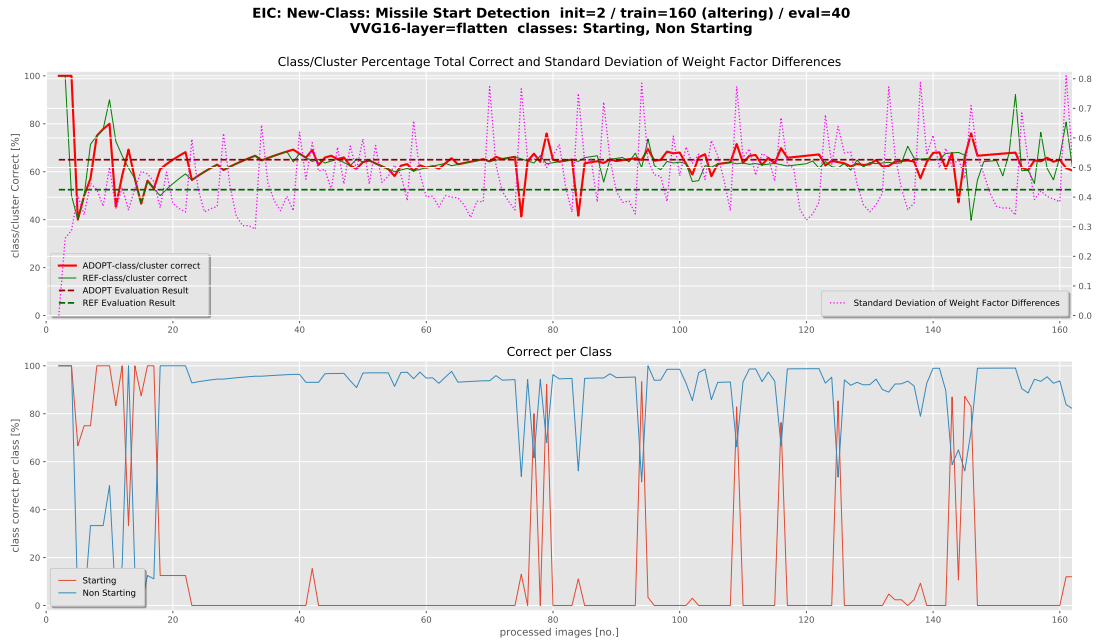


Figure 4.1: Classification training and evaluation performance with feature values from hidden VGG16 *flatten* layer outputs (first plot: red graph=ADOPT-class/cluster correct, green graph=REF-class/cluster correct, dark red dashed graph=ADOPT evaluation result, dark green dashed graph=REF evaluation result, magenta dotted graph=standard deviation of weight factor differences; second plot: red graph=Starting, blue graph=Non Starting).

There is a high percentage of correct classification for images with *Non Starting* ground-truth and only for some steps a correct classification of images with *Starting* ground-truth (see the second plot in Figure 4.1).

The evaluation result for the process with feature weight optimization was about 65% and for the reference model, it was about 53% (see the first plot in Figure 4.1). This means that the performance of the reference model is close to random (50%) classification and that of the process with feature weight optimization only 15% above, which is an unsatisfying performance. The image-related evaluation results are shown in Figure 4.2.

Even though the performance of the processed is significantly better than that of the reference model, the absolute evaluation performance values are only a little above

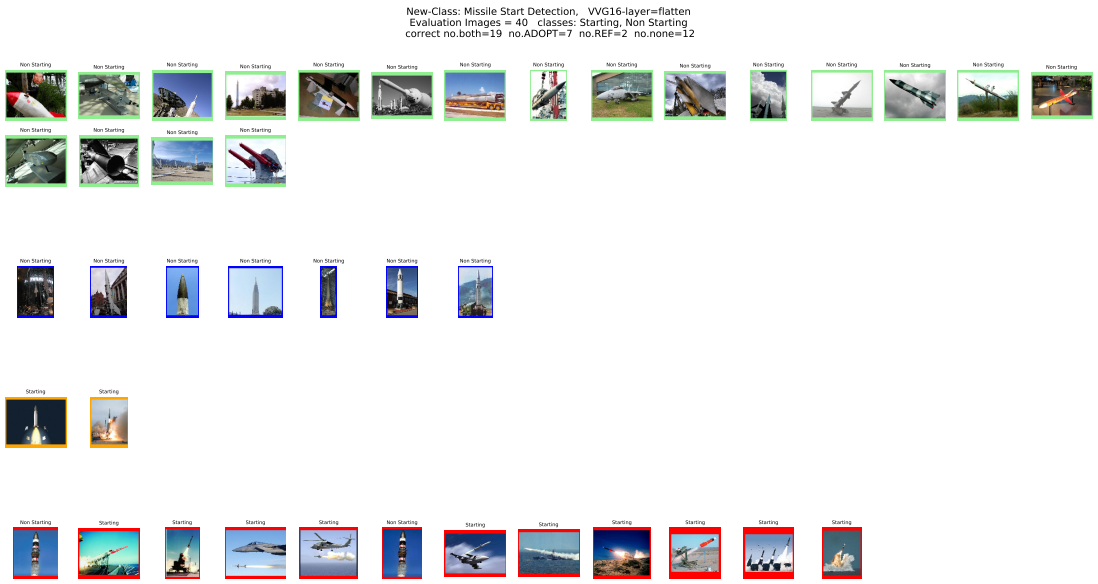


Figure 4.2: Image related performance of evaluation phase with feature values from hidden VGG16 *flatten* layer outputs.

random classification. Consequently, the hidden VGG16 *flatten* layer output is not a useful source for the feature values for our model.

4.2.2 VGG16 Output Layer *fc1*

For this test we used the output values of the hidden VGG16 *fc1* layer as image feature values for the following k-means clustering algorithm and feature weight optimization calculations.

There is a high percentage of correct classification for images with *Non Starting* ground-truth and a significant increase of periods in number and duration with correct classification of images with *Starting* ground-truth (see the second plot in Figure 4.3). In comparison to the usage of the *flatten* layer, this is a significant performance increase, but only for limited periods.

The evaluation result for the process with feature weight optimization was about 60% and for the reference model, it was about 57% (see the first plot in Figure 4.3). In comparison to the results of the evaluation phase with the *flatten* layer, this means a performance decrease of the process with feature optimization and slightly better performance of the reference model. Considering the performance development volatility, the evaluation results are owed a bad point, when the training phase stopped. Never the less, this unpredictable periodicity and its consequences make the result unsatisfactory. The image-related evaluation results are shown in Figure 4.4.

Even though the performance of the processed is in some periods significantly better

Figure 4.3: Classification training and evaluation performance with feature values from hidden VGG16 *fc1* layer outputs. (first plot: red graph=ADOPT-class/cluster correct, green graph=REF-class/cluster correct, dark red dashed graph=ADOPT evaluation result, dark green dashed graph=REF evaluation result, magenta dotted graph=standard deviation of weight factor differences; second plot: red graph=Starting, blue graph=Non Starting)

compared with the use of *flatten* layer, the unpredictable periodicity of performance makes the VGG16 *fc1* layer output not a useful source for the feature values for our model.

4.2.3 VGG16 Output Layer *fc2*

For this test we used the output values of the hidden VGG16 *fc2* layer as image feature values for the following k-means clustering algorithm and feature weight optimization calculations.

There is a high percentage of correct classification for images with *Non Starting* ground-truth and a significant increase of the duration of the periods with correct classification of images with *Starting* ground-truth (see the second plot in Figure 4.5).

The evaluation result for the process with feature weight optimization was about 89% and for the reference model, it was about 83% (see the first plot in Figure 4.5). In comparison to the results of the evaluation phase with the *flatten* and *fc1* layer, this

Figure 4.4: Image related performance of evaluation phase with feature values from hidden VGG16 *fc1* layer outputs.

means a significant evaluation performance increase. The image-related evaluation results are shown in Figure 4.6.

In comparison to the use of the *flatten* and *fc1* layer, the number of correctly classified images is not only higher but contains also a significantly higher amount of *Starting* ground-truth images.

The results of our three VGG16 hidden layer tests correspond to the test results of the same VGG16 layers in the paper of Guerin et al. [GGTN17]. We expected these results because the context of a fully connected layer output increases the closer the layer is to the end of the VGG16 process pipeline. The reason for this, from the architectural aspect, is that the tested layers are fully connected, the last layer represents the final class assignment, and the training of the pre-trained VGG16 CNN uses backpropagation. Consequently, the information context, related to the classes to be extracted, increases for layers at a later position in processing order. This increase of contextual information in the layer output fosters a correct classification by the k-means algorithm. That is why the test results are better for the *fc2* layer than those for the *fc1* layer, and the *fc1* layer result better then that of the *flatten* layer.

As a result, we selected the hidden VGG16 *fc2* layer output as the source for image feature values for all of our analysis and evaluation in our work.

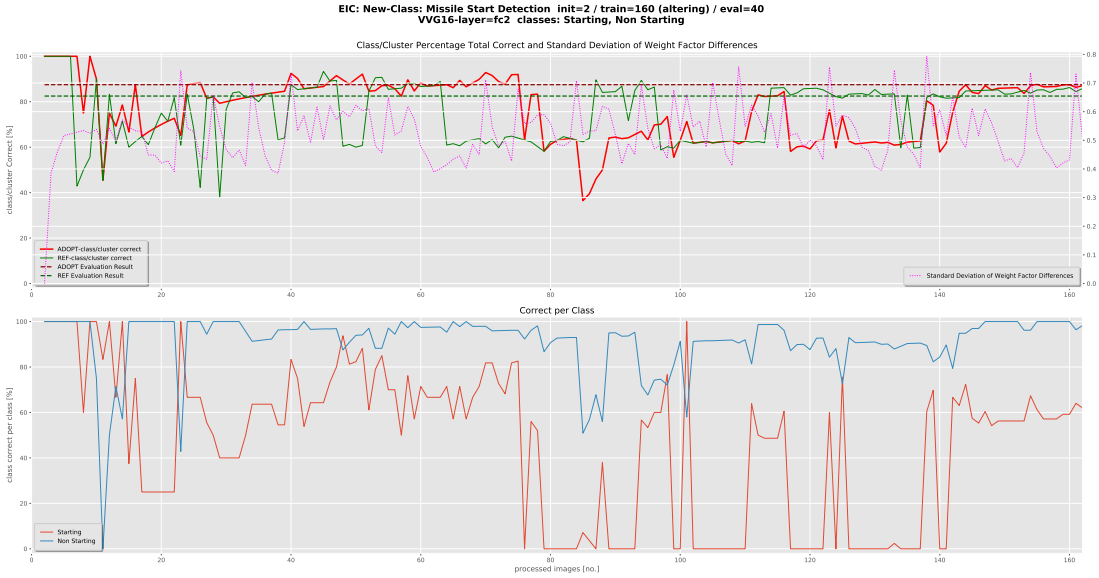


Figure 4.5: Classification training and evaluation performance with feature values from hidden VGG16 *fc2* layer outputs. (first plot: red graph=ADOPT-class/cluster correct, green graph=REF-class/cluster correct, dark red dashed graph=ADOPT evaluation result, dark green dashed graph=REF evaluation result, magenta dotted graph=standard deviation of weight factor differences; second plot: red graph=Starting, blue graph=Non Starting)

4.3 Evaluations on ImageNet Classes

With the following tests, we aim to evaluate the classification performance of our model on two subjectively similar classes in comparison to two subjectively diverse classes.

For the test cases, we used images from ImageNet and classes defined in ImageNet. We processed two different test cases: For the first test case, we used subjectively divergent images, showing motor scooters and dogs. For the second test case, we used subjectively similar images, showing dogs of two different races.

We selected a large number of images to also make visible long term performance effects. In each test case, we used 5 images for the initialization phase, 300 images for the training phase and 50 images for the evaluation phase from each class.

4.3.1 Test Case: Intuitive Diverse Classes

In this test case, we analyzed the results of our model classifying performance on subjectively diverse classes. For this, we selected from the ImageNet database the two categories *scooter* with class id=*n03791053* and *Gordon setter* with class id=*n02101006*. We expected a very good classification performance of our model, given by the subjective diversity of these two classes.

Figure 4.6: Image related performance of evaluation phase with feature values from hidden VGG16 *fc2* layer outputs.

The model with weight optimization showed volatility from roughly 75% to 100% in the starting phase of the training phase (see the first plot in Figure 4.7). The reference model performed in the same phase atroughly 75%. After this phase, both models showed a performance close to 100%. The model with weight optimization showed performance volatility from 96% to 100% that diminished significantly after 350 processed images and stabilized close to 100%.

The evaluation performance result for the process with feature weight optimization was 100% and that of the reference model was also 100%. This means that the evaluation performance of both models is ideal. The image-related evaluation results are shown in Figure 4.8.

There is a constant 100% correct classification for images of the *scooter* ground-truth (see second plot in Figure 4.7). The performance volatility of the *Gordon setter* ground-truth classification is responsible for the variation of the total classification performance.

The min-, max-, mean- and standard deviation values of the feature weight factors that are limited to the interval of [0..2] are listed in Table 4.7. The mean value of 1.243 indicates a majority of correctly classified features.

The results of the subjectively diverse classes test show perfectly performing models. The process with feature weight optimization and the reference model reached an ideal evaluation performance of 100%.

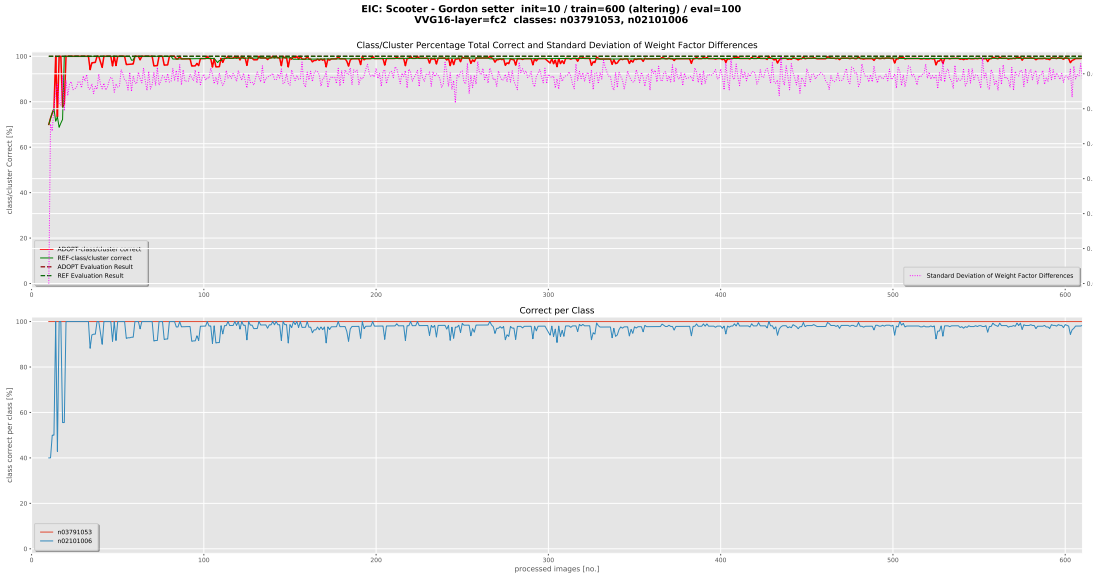


Figure 4.7: Classification training and evaluation performance for subjectively diverse classes *scooter* and *Gordon setter* (first plot: red graph=ADOPT-class/cluster correct, green graph=REF-class/cluster correct, dark red dashed graph=ADOPT evaluation result, dark green dashed graph=REF evaluation result, magenta dotted graph=standard deviation of weight factor differences; second plot: red graph=n03791053, blue graph=n02101006).

var	value
min	0.0397
max	2.0000
mean	1.2430
std.dev.	0.4598

Table 4.7: Feature weigh factor min-, max-, mean- and standard deviation values for subjectively diverse classes *scooter* and *Gordon setter*.

4.3.2 Test Case: Intuitive Similar Classes

In this test case, we analyzed the results of our model classifying performance on subjectively similar classes. For this, we selected from the ImageNet database the two categories *English setter* with class id=*n02100735* and *Gordon setter* with class id=*n02101006*. We expected a worse result than that, using subjectively diverse classes (see Section 4.3.1), given by the subjective similarity of these two classes.

The model with weight optimization showed volatility from roughly 75% to 95% in the starting phase of the training phase (see the first plot in Figure 4.9). The reference model

Figure 4.8: Image related performance of evaluation phase for subjectively diverse classes *scooter* and *Gordon setter*.

showed in the same phase performance volatility of roughly 70% to 80%. The average performance from process step 350 until processing end, was about 95.1% for the process with feature optimization and about 92.7% for the reference model. The related standard deviations were 3.7% and 2.0%.

The evaluation performance result for the process with feature weight optimization was 99% and that of the reference model was 95%. This result shows a clear performance advantage of the process with feature weight optimization in comparison to the reference model. The image-related evaluation results are shown in Figure 4.10.

There is a nearly constant correct classification, close to 100%, for images of the *English setter* ground-truth (see the second plot in Figure 4.9). The performance volatility of the *Gordon setter* ground-truth classification was high at the beginning and diminished with continuing training phase.

The min-, max-, mean- and standard deviation values of the feature weight factors that are limited to the interval of [0..2] are listed in Table 4.8. The mean value of 1.24 indicates a majority of correctly classified features.

var	value
min	0.0105
max	1.9998
mean	1.2402
std.dev.	0.4197

Table 4.8: Feature weigh factor min-, max-, mean- and standard deviation values for subjectively similar classes *English setter* and *Gordon setter*.

The results of the subjectively similar classes test show good performing models. The process with feature weight optimization shows a correct classification percentage of 99%

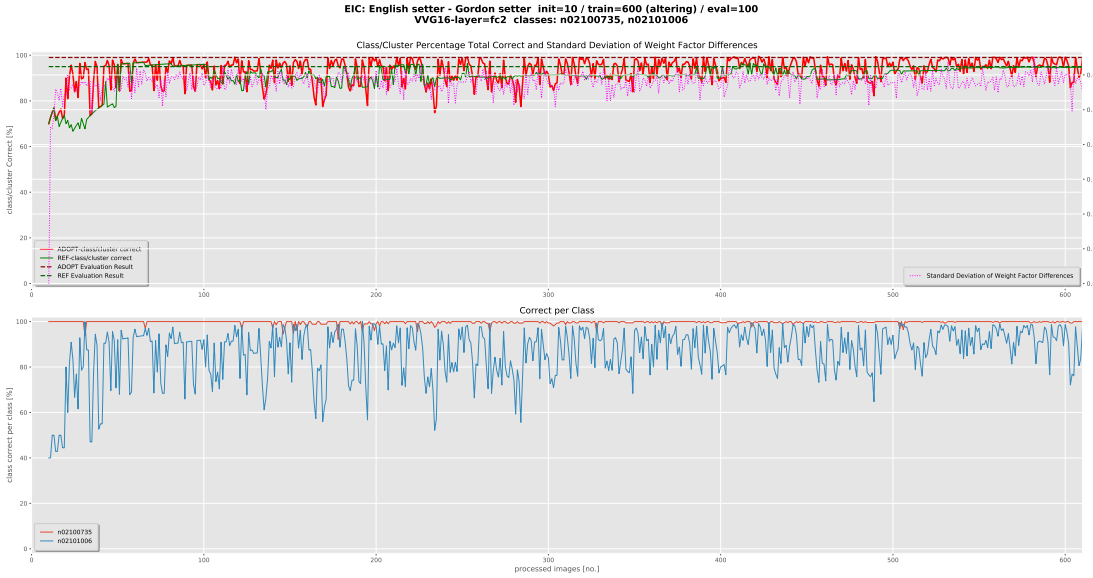


Figure 4.9: Classification training and evaluation performance for subjectively similar classes *English setter* and *Gordon setter* (first plot: red graph=ADOPT-class/cluster correct, green graph=REF-class/cluster correct, dark red dashed graph=ADOPT evaluation result, dark green dashed graph=REF evaluation result, magenta dotted graph=standard deviation of weight factor differences; second plot: red graph=n02100735, blue graph=n02101006).

and outperforms the reference model by with 4%.

4.4 Evaluations of Manually Labeled Images with New Classes

With the following tests, we aimed to evaluate the performance of our model with a ground-truth that the VGG16 CNN was not trained on. We solely aim to discover general performance level differences to the tests with ground-truth that the VGG16 was trained on. Analyzing the reasons for the differences is an interesting task for future work.

For the following test cases we used images from ImageNet and manually assigned a ground-truth that is not defined in ImageNet and the VGG16 CNN is not trained on. We processed two different test cases: In the first test case, we used images showing missiles that we labeled with ground-truth *Starting* in case the missile was starting, otherwise with ground-truth *Non Starting*. In the second test case, we used images showing refrigerators that we labeled with ground-truth *Open* in case the refrigerator was open, otherwise with ground-truth *Not Open*.

For a reasonable test scenario size, we labeled 506 images manually for each test case. Furthermore, we adapted the training and evaluation application, to process manual

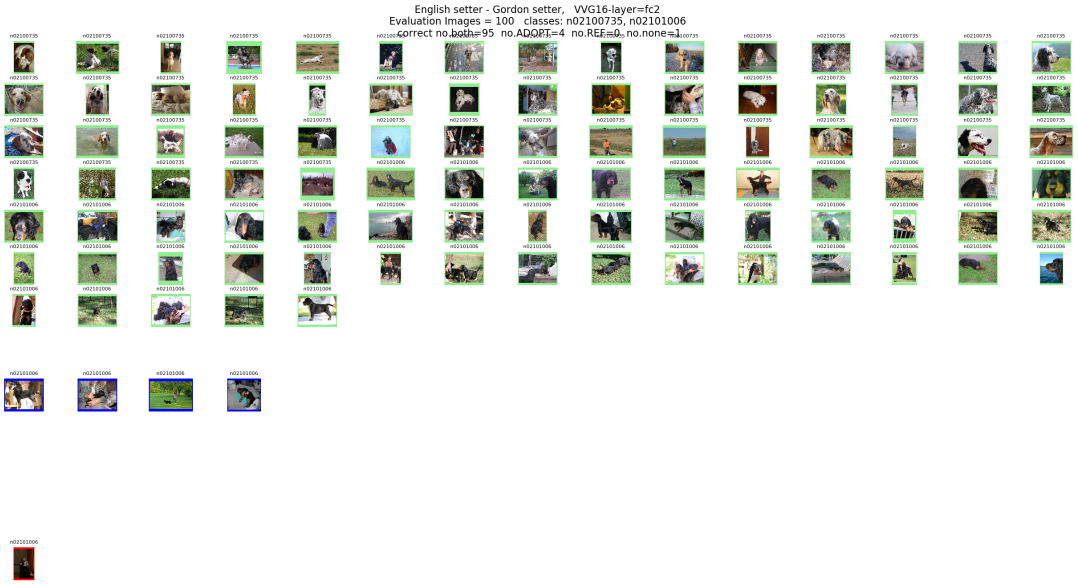


Figure 4.10: Image related performance of evaluation phase for subjectively similar classes *English setter* and *Gordon setter*.

ground-truth. In each test case, we assigned 3 images per class (in total 6) to the initialization phase, 200 images per class (in total 400) to the training phase and 50 images per class (in total 100) to the evaluation phase.

4.4.1 Test Case: Manual Ground-Truth on Missile Images

For this test case, we selected images from the ImageNet database with the category *missiles* with class id=*n03773504*. We downloaded 506 images of this category and labeled each image manually with the ground-truth *Starting* in case the missile shown was starting, otherwise with ground-truth *Non Starting*. Consequently, we got an ImageNet independent classified image set with two classes.

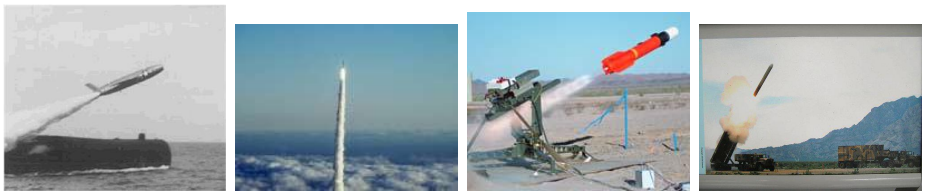
The model with weight optimization showed in the starting phase of the training phase high volatility (see the first plot in Figure 4.11). The reference model performed constantly low in the same phase. The average performance from process step 108 to step 340 of the model with feature optimization was about 84.0% and that of the reference model was about 79.0%. The related standard deviations were 4.78% and 4.53%.

After process step 340 of the process with feature weight optimization, unexpected reductions occurred in process steps 341, 386 to 387 and 404. The performance was reduced after one to two process steps after these occurrences. The images that were processed in these steps are shown in Figure 4.12 and are all with ground-truth *Starting*. Four images with ground-truth *Starting* that were correctly classified are shown as examples in Figure 4.13. A further deeper analysis of the reasons for these performance

Figure 4.11: Classification training and evaluation performance for missile images labeled with classes *Starting* and *Non Starting* (first plot: red graph=ADOPT-class/cluster correct, green graph=REF-class/cluster correct, dark red dashed graph=ADOPT evaluation result, dark green dashed graph=REF evaluation result, magenta dotted graph=standard deviation of weight factor differences; second plot: red graph=Starting, blue graph=Non Starting).

reductions would be an important task for future work.

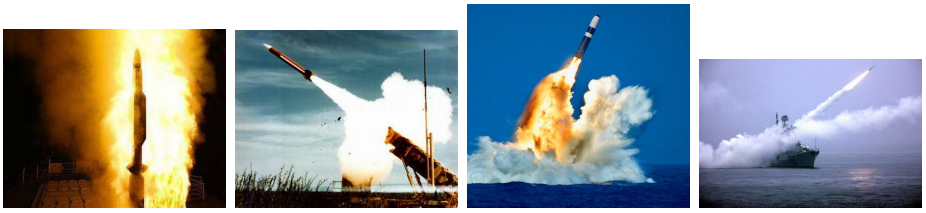
The performance evaluation result for the process with feature weight optimization was about 72% and that for the reference model was also about 72%. The image-related evaluation results are shown in Figure 4.14. These results show a clear performance reduction in comparison to the performance reached with classes that are already defined in ImageNet and that the VGG16 CNN is trained on (see Section 4.3).



(a) Image at process step 341 (b) Image at process step 386 (c) Image at process step 387 (d) Image at process step 404

Figure 4.12: Images processed at process steps with significant performance break in.

There is a balanced performance of the correct classification between both classes on a high level, except at points of unexpected performance reduction (see the second plot in



(a) Image at process step 1 (b) Image at process step 15 (c) Image at process step 184 (d) Image at process step 345

Figure 4.13: Examples for correct classified images with ground-truth *Starting*.

Figure 4.11). The average performance from process step 108 to 340 for ground-truth *Starting* was about 80.8% and for ground-truth *Non Starting* about 85.3%.

The min-, max-, mean- and standard deviation values of the feature weight factors that are limited to the interval of [0..2] are listed in Table 4.9. The mean value of 0.9153 indicates a smaller number of correctly classified features than not correctly classified features.

var	value
min	0.0013
max	1.9957
mean	0.9153
std.dev.	0.4719

Table 4.9: Feature weigh factor min-, max-, mean- and standard deviation values for missile images labeled with classes *Starting* and *Non Starting*.

The results of this test show a well-performing model, except at the three points in the process, when there was a significant short term performance reduction.

4.4.2 Test Case: Manual Ground-Truth on Refrigerator Images

For this test case we selected images from the ImageNet database with the category *refrigerator* with class id=*n04070727*. We downloaded 506 images of this category and labeled each image manually with the ground-truth *Open* in case the refrigerator is shown in an open state, otherwise with ground-truth *Not Open*. Consequently, we got an ImageNet independent classified image set with two classes.

There is also a short starting phase as in the other tests but with less volatility (see the first plot in Figure 4.15). After a starting phase, the performance of the process with weight optimization and the reference model are staying on a quite constant performance level, whereas the process with weight optimization shows higher volatility. The average performance from process step 20 to 402 of the process with feature optimization was

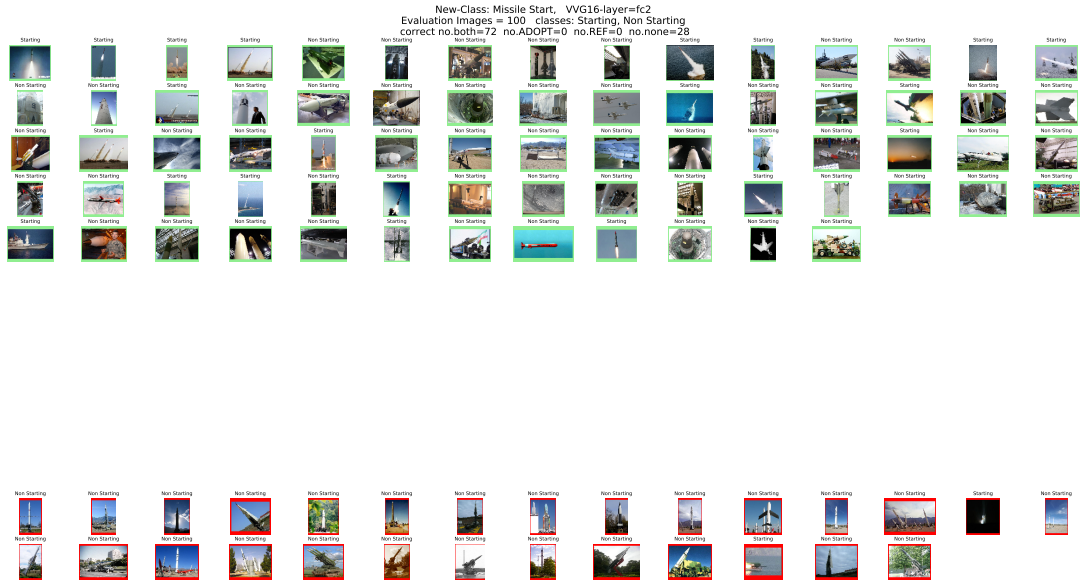


Figure 4.14: Image related performance of evaluation phase for missile images labeled with classes *Starting* and *Non Starting*.

about 77.5.0% and that of the reference model was about 76.0%. The related standard deviations were 2.53% and 1.66%.

The performance evaluation result for the process with feature weight optimization was about 91% and that of the reference model was about 89%. The image-related evaluation results are shown in Figure 4.16. These results lie roughly 8% under those of the test cases with ImageNet defined classes (see Section 4.3) but can be considered as very good.

There is a nearly constant correct classification close to 100% for images with ground-truth *Not Open* (see second plot in Figure 4.15). The average classification performance of images with ground-truth *Open* was about 61.1% with a standard deviation of about 4.8%.

The min-, max-, mean- and standard deviation values of the feature weight factors that are generally within the interval of [0.0..2.0] are listed in Table 4.10. The mean value of 1.0945 indicates a larger number of correctly classified features than not correctly classified features.

Even though the test showed lower performance rates than the tests with sclasses defined in ImageNet that the VGG16 CNN was trained on, the expected performance decrease is still acceptable. Also, the performance of our model with weight optimization was better or at least equal to the performance of the reference model.

Figure 4.15: Classification training and evaluation performance for refrigerator images labeled with classes *Open* and *Not Open* (first plot: red graph=ADOPT-class/cluster correct, green graph=REF-class/cluster correct, dark red dashed graph=ADOPT evaluation result, dark green dashed graph=REF evaluation result, magenta dotted graph=standard deviation of weight factor differences; second plot: red graph=Not Open, blue graph=Open).

var	value
min	0.0014
max	1.9999
mean	1.0945
std.dev.	0.5121

Table 4.10: Feature weigh factor min-, max-, mean- and standard deviation values for refrigerator images labeled with classes *Open* and *Not Open*.

4.5 Result Overview

An overview of the test results is given in Table 4.11.

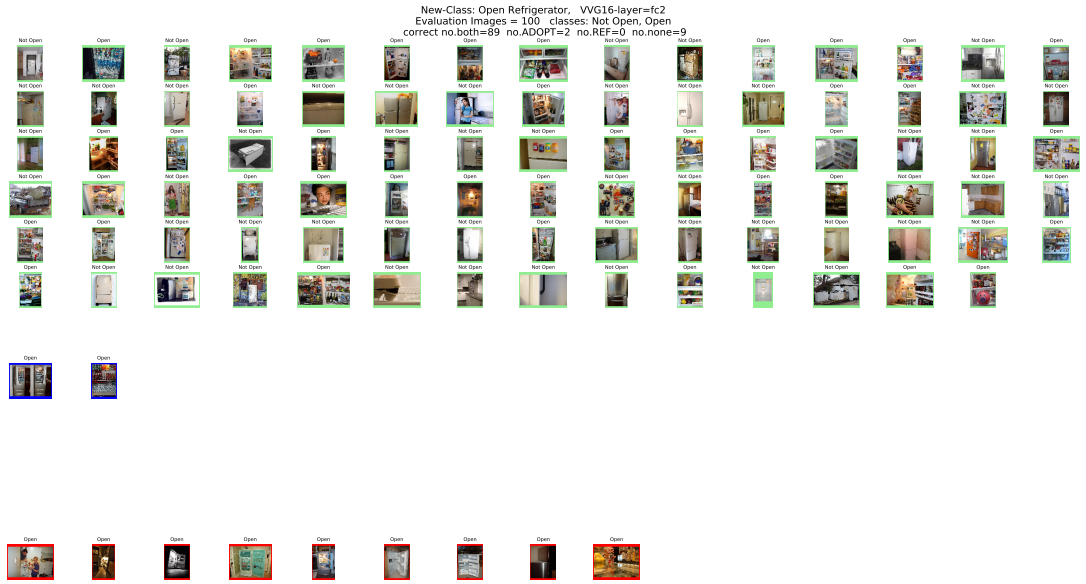


Figure 4.16: Image related performance of evaluation phase for refrigerator images labeled with classes *Open* and *Not Open*.

test category	test	VGG16 output layer	manually labeled [Y/N]	initialization image qty.	training image qty.	evaluation image qty.	classification rate [%]	classification rate ref. mod. [%]	w_f min	w_f max	w_f mean	w_f standard deviation
model verification	clearly sep.	fc2	N	4	10	0						
	neutral feat.	fc2	N	4	10	0						
	oppos. feat.	fc2	N	4	10	0						
VGG16 level selection	flatten layer	flatten	N	2	160	40	65	53	0.0	2.0	0.94	0.52
	fc1 layer	fc1	N	2	160	40	60	57	0.0	2.0	0.95	0.53
	fc2 layer	fc2	N	2	160	40	89	83	0.0	2.0	0.91	0.54
ImageNet ground-truth	subj. diverse	fc2	N	10	600	100	100	100	0.04	2.0	1.24	0.46
	subj. similar	fc2	N	10	600	100	99	95	0.00	2.0	1.24	0.42
Manually ground-truth	missile start	fc2	N	6	400	100	72	72	0.01	2.0	0.91	0.47
	refrig. open	fc2	N	60	400	100	91	89	0.00	2.0	1.09	0.51

Table 4.11: Overview of test results.

CHAPTER 5

Conclusion

5.1 Model Performance

We developed an image classification optimization model that is based on the use and optimization of feature weight factors. These feature weight factors are applied to the feature values, generated by the feature extraction process before they are processed by the clustering algorithm. This concept allows us the use of arbitrary feature extraction processes and clustering algorithms. It should also lead to a shorter process run-time in comparison to processes that optimize a large number parameters in different sections of the process because our optimization only applied on a relatively small number of feature weight factors in comparison.

We built our model architecture to enable step-wise image classification optimization. We calculate new weight factors in each optimization step, based on the clustering contribution of a newly added image to the clustering result of the previous step. This step-wise optimization approach of the current model allows a dynamic optimization during operation, where the performance optimization effect becomes immediately transparent after each step.

We developed the optimization algorithms and tested our model classification performance in comparison to a reference model. We performed our tests on a model configuration that uses the pre-trained VGG16 CNN for image feature extraction and the Keras in-built k-means algorithm for clustering. We carried out three tests on three hidden VGG16 layers, to determine the *fc2* hidden VGG16 layer that we use in our model. The classification performance tests we carried out with different ground-truth, each containing two classes that were defined in ImageNet and which the VGG16 was trained on (see Section 4.3). Furthermore, we carried out two classification performance tests with different manually labeled ground-truth, each with two classes that were not defined in ImageNet and which the VGG16 was not trained on.

Our tests showed that our model provides a better or at least equivalent classification performance in all tested scenarios compared to the reference model. The absolute performance evaluation results (with exception to the test case with manually labeled missile images) showed high classification rates in the range from about 91% to 100%. Especially in the test cases with a ground-truth that the VGG16 CNN was not trained on, we evaluated promising performance for practical use.

In the course of our work, we developed a Python application that implements our model and provides a user-friendly interface. It allows the easy set-up of test cases and provides graphics and tables for a comprehensive evaluation on process steps level. We consider this application as a starting point for future work.

We see as the advantage of our model approach that the optimization requires no internal changes of the applied feature extraction and clustering algorithms, hence pre-trained models or closed-source models can be used. As a further advantage we see the step-wise transparency of the performance development during the training phase for each newly added image as opposed to batch-based training for CNNs. This enables a dynamical control of the training phase by the user. We further see as an advantage the small number of parameters to be optimized, which results in reduced processing time. A last advantage is the classification performance of our model that outperforms the reference model without feature weight optimization.

5.2 Future Work

During our work, we discovered several interesting potential tasks for future work, as well as some more detailed analysis that could be performed on our results.

An interesting task would be the comparison of the classification model parameters in our model to the corresponding classification parameters in the VGG16 CNN. We are using in our model the output values of the VGG16 *fc2* hidden layer, which is the last fully connected layer in the VGG16 CNN before the *predictions* layer that delivers the classification results of the VGG16 CNN. Comparing the feature weight factors we calculate in our model to the input weights of the *predictions* layer of the VGG16 CNN and analyzing the result could be a task for future work. This can become challenging, because we use only one weight factor vector with feature dimension and the VGG16 uses for each class a different weight factor vector with feature dimension. When considering the comparison of our calculated weight factors for classification based on the VGG16 *flatten* layer to the weight factors used by the VGG16 CNN to do the classification, an additional problem occurs, because there are three fully connected layers in the VGG16 CNN in between. Therefore, this could be a hard challenge to set up a comparison model that evaluates the results and parameters on feature level.

A less work-intensive task of future work could be a plain but detailed comparison of classification performance of the VGG16 CNN to our model, using different image set

configurations. This would show the performance differences of our model that uses k-means clustering in comparison to a classification model that is a pure CNN solution.

A very important task for future work would be the evaluation of our model, applied on more than two classes. Our evaluations focused on two class classification cases only, also we designed our training and evaluation tool to process multiple class classification. Especially the classification performance evaluation with multiple class sets of our model, in comparison to the reference model, would be interesting, and would finally show the robustness of our model.

A further task for future work should be the performance analysis of our model, using different image set sizes on different hardware platforms. Our tests were made with a maximum image set size of 710 images, from which 600 images were used for training and 100 images for evaluation. We used the hidden pre-trained VGG16 *fc2* output layer and the application was running on an intel i7 processor without using AVX/AVX2 and no NVIDIA GPU support. The duration for the training and evaluation phase with 710 images was about one and a half hour. We expect much faster processing when using a NVIDIA GPU or AVX/AVX2.

Finally, we see an important task for future work in analyzing the rare performance reductions in the test case described in section 4.4.1.

List of Figures

List of Tables

Bibliography

[AV07] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[CWM+17] Jianlong Chang, Lingfeng Wang, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. Deep adaptive image clustering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5879–5887, 2017.

[FTG18] Maziar Moradi Fard, Thibaut Thonet, and Eric Gaussier. Deep k-means: Jointly clustering with k-means and learning representations. *arXiv preprint arXiv:1806.10069*, 2018.

[GGTN17] Joris Guérin, Olivier Gibaru, Stéphane Thiery, and Eric Nyiri. Cnn features are also great at unsupervised classification. *arXiv preprint arXiv:1707.01700*, 2017.

[HL17] Chih-Chung Hsu and Chia-Wen Lin. Cnn-based joint clustering and representation learning with feature drift compensation for large-scale image data. *IEEE Transactions on Multimedia*, 20(2):421–429, 2017.

[Llo82] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[RGB16] Andras Rozsa, Manuel Günther, and Terrance E Boult. Are accuracy and robustness correlated. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 227–232. IEEE, 2016.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[XGF16] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487, 2016.

[YFSH17] Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3861–3870. JMLR. org, 2017.