University of Zurich[UZH]

Large-scale Active Learning for Concept Detection in Video

Thesis October 1, 2021

Simon Widmer
of Zürich ZH, Switzerland

Student-ID: 04-605-903
simon.widmer2@uzh.ch

Advisor: **Dr. Luca Rossetto**

Prof. Abraham Bernstein, PhD
Institut für Informatik
Universität Zürich
http://www.ifi.uzh.ch/ddis

Acknowledgements

My thanks go first and foremost to Luca Rossetto. He supported me at all times with his great technical knowledge and patiently answered all my questions. Whenever I got lost in unimportant details, he was able to get me back on track. In addition to his great support for this project, I would also like to thank him for the many interesting discussions we have had over the past six months. I also want to thank Prof. Abraham Bernstein, PhD, for giving me the opportunity to work on this challenging project. Sometimes, I almost despaired of the challenges. However, I was able to acquire a lot of new knowledge in topics that I had not been familiar with before. My thanks also go to Florian Spiess, who actively supported me whenever I faced a problem with the Cottontail DB Python client. I would also like to thank the entire team that develops and maintains the Cottontail DB, which I use extensively in this work. Last but not least, I would like to thank my friends and family who had to put up with my absence and sometimes my tense nature. I look forward to having more time for them in the future.

Zusammenfassung

Moderne Klassifizierungssysteme, die auf neuronalen Netzen basieren, benötigen oft grosse Trainings-Sets und haben mit einer abnehmenden Klassifizierungsleistung zu kämpfen, wenn sie mit ungesehenen Objektkategorien konfrontiert werden. Diese Arbeit untersucht praktische und effektive Wege, um eine aktive Lernpipeline für die Konzepterkennung in Videos zu implementieren, welche in der Lage ist, ständig neue Objektkategorien aus annotierten Bildern zu lernen, die von Menschen bereitgestellt werden. Um dieses Ziel zu erreichen, verwendet die vorgeschlagene Pipeline eine aktive Lernschleife mit einer einfachen, auf Unsicherheit basierenden Heuristik, um die informativsten Bilder für die Annotation auszuwählen. Die Bewertung von vier verschiedenen Convolutional Neural Networks für die Einbettung von Bildmerkmalen hat gezeigt, dass die InceptionResNetV2-Architektur in allen untersuchten Klassifizierungsszenarien die beste Leistung erbringt. Ausserdem gibt es keine spezifische Klassifizierungsmethode, die in allen Klassifizierungsszenarien am besten funktioniert. Es ist vorteilhaft, dem System die Wahl des 'besten' Klassifikators für jede Klassifizierungsaufgabe zu überlassen. Darüber hinaus kann die Klassifizierungsleistung bei sehr kleinen Trainings-Sets weiter verbessert werden, wenn extrahierte Box-Bilder als zusätzliche Trainingsinstanzen verwendet werden.

Abstract

Modern neural network based classifications system often require large training sets and struggle with degrading classification performance when confronted with unseen objects categories. This thesis investigates practical and effective ways to implement a large-scale active learning pipeline for concept detection in videos, which is capable to constantly learn new object categories from annotated images provided by human supervisors. The proposed pipeline uses an active learning loop with a simple uncertainty-based heuristic to select the most informative images for annotation to achieve this goal. The evaluation of four different convolutional neural networks for image feature embedding showed that the InceptionResNetV2 architecture delivers the best performance over all studied classification scenarios. Furthermore, there is no single classification methods which works best in all classification scenarios. It is advantageous to let the system chose the 'best' classifier for each classification task. Moreover, the classification performance can be further improved for very small training sets if extracted box images are added as training instances.

Contents

1

Introduction

Object classification in images and videos is one of the major problems in computer vision. Almost 20 years ago, Sivic and Zisserman [Sivic and Zisserman, 2003] presented a new approach to localize object in videos. They used scale-invariant feature transform (SIFT) descriptors and a visual vocabulary to detect and classify objects. Since then, significant advances have been made. Modern neural network based methods allow to classify objects with high precision. However, many approaches rely on an extensive collection of training instances and their classification performance starts to degrade when confronted with unseen objects categories. Another major drawback of neural network classifiers is that the learning process for new object categories requires the retraining of the entire network, or at least fine-tuning the classification layer. Yet, fine-tuning can lead to unsatisfactory classification results if the training sets for each object category are highly unbalanced. Fortunately, active learning loops provide a framework to address this challenge.

The aim of this thesis is to investigated practical and effective ways to implement a large-scale active learning pipeline for concept detection in videos. Using an active learning loop, the system should be able to continuously learn new object categories from annotated images provided by human supervisors. At the same time, the burden of image annotation should be reduced as much as possible. Hence, with each training cycle, the system is expected to extend its capabilities to detect objects in images and videos.

The project of developing an active learning pipeline for concept detection in videos can be divided into individual subtasks: First, object features from individual images must be extracted. Thereby, the feature vector space of the embedding model must be large enough to reliably distinguish between an unknown number of object types. Second, a suitable classification method must be found that has the ability to constantly learn new object categories. Third, the system must be an effective learner. This means that it only needs a limited number of annotated images to learn new objects. One such method is active learning, where the system only ask for image annotation which are most informative for classification. Last but not least, a working application must be developed, which allows for extensive testing of the proposed pipeline.

The structure of the thesis is based on the above-mentioned subtasks. Chapter 2 introduces convolution neural networks which demonstrated impressive classification performance results in large-scale image classification challenges like the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [Russakovsky et al., 2015]. The four different network architectures that are considered as feature extraction model for the active learning pipeline are discussed in detail. Chapter 3 examines the classification problem and presents seven classifier candidates for the pipeline. Chapter 4 presents preliminary classification results of different combinations of feature extraction models and classifier types. Then, chapter 5 takes a closer look at active learning methods and introduces the uncertainty based heuristic chosen for this thesis. Chapter 6 discusses the pipeline design and explains the mode of operation of the proposed pipeline. It also discusses implementation details and limitations. The evaluation results of the active learning pipelines are presented in Chapter 7. It also draws important conclusions for an active learning pipeline for concept detection and identifies open questions that should be explored in future research on this topic. Chapter 8 provides and overview over all aspects of the active learning pipeline for concept detection in video that could not be analyzed conclusively. The chapter formulates open questions guides future research on the topic. The thesis ends with the conclusions in Chapter 9.

2

Feature Extraction

Scientific interest for object detection in images and videos received a boost with the introduction of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2010 [Russakovsky et al., 2015]. A major breakthrough was achieved by Krizhevsky et al. [Krizhevsky et al., 2012] with the introduction of AlexNet. It was the first contender to use a convolutional neural network and won the competition by a big margin. Since AlexNet, ConvNets have continued to push the boundaries in the field of image classification and object detection.

This chapter introduces the four ConvNets used for feature extraction in this thesis. Those features are then fed to a bank of classifiers for classification (see Chapter 3). The chapter starts with an introduction of the general architecture of ConvNets. Sections 2.2, 2.3, 2.4 and 2.5 introduce the Inception, ResNet, InceptionResNet and MobileNet architectures, respectively. Every section briefly discusses the characteristics of each architecture and explores its properties.

2.1 Convolutional Neural Networks

The quality of image classification and object recognition tasks has made tremendous progress in recent years. This rapid progress is mainly due to the widespread use and of Convolutional Neural Networks (CNNs or ConvNets). This section provides a brief introduction to the general architecture of ConvNets and explores the reason for their success.

ConvNets consist of three main types of layers: (i) convolutional layers, (ii) pooling layers and (iii) fully-connected layers. In the following section, we take a brief look at each of the three layer types.

2.1.1 Convolutional layer

A convolutional layer consists of three main components: (i) an input, (ii) a filter or kernel, and (iii) an output array or feature map [IBM Cloud Education, 2020]. Figure 2.1 depicts the three components of a convolutional layer. For a color image, the input is a 3-dimensional matrix, where the third dimension corresponds to the RGB values and is called depth. The filter or kernel is a $n \times n$ matrix (or a $n \times n \times 3$ matrix in case

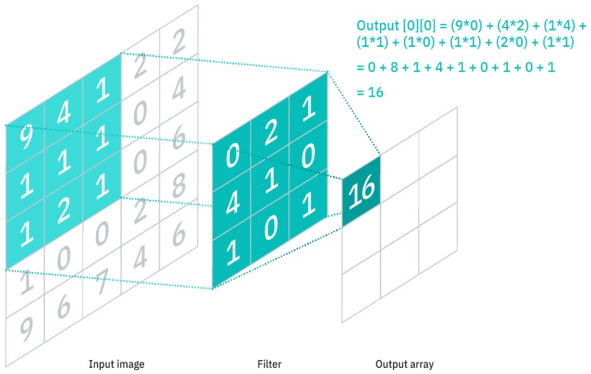
of an RGB image) of weights. These weights are learned by the neural network during training by stochastic gradient descend with back propagation of errors. The size of the kernel defines the receptive field and can vary in size, however, it is typically a 3×3 matrix. By taking the dot product between the input values and the kernel, we get a value for the feature map (in figure 2.1, the value is 16). The kernel is then moved over the entire image to produce a complete feature map. The pixel distance between each kernel movement is called stride, where larger strides leads to a smaller feature maps. However, most ConvNets apply strides of either 1 or 2 pixels.

After each convolutional operation, an activation function is applied to the feature map. Such activation functions introduce non-linearity and help the network to learn more complex mapping functions. **Sigmoid**, **hyperbolic tangent**, and **ReLU** are three common activation functions. The logistic sigmoid function maps the output values to the range $[0, 1]$, whereas the hyperbolic tangent maps the output values to the range $[-1, 1]$. Both activation functions feature symmetric saturation cut-offs, which means that values above the limits are truncated. Once saturation has been reached, it gets difficult to improve the model performance by adjusting the weights during the learning process. This ultimately leads to the *vanishing gradients problem*, where back propagated errors become too small to contain any useful gradient information for very deep layers of a neural network and, therefore, prevent effective learning. The rectified linear activation unit (ReLU) overcomes these issues. It maps the output values to the range $[0, \infty)$. [I. Goodfellow and Courville, 2016]

In order to see intuitively how convolutional layers are suitable for image classification tasks, consider a ConvNet consisting of a cascade of convolutional layers. Convolutional layers deep down in the network cover a spatially limited receptive field. Through training, these layers produce feature maps for locally prominent features like single edges, corners or radii. Progressing further up in the network, the convolutional layers take these feature maps as input and learn kernel weights which produce feature maps of more complex shapes, like circles, squares or star shapes. Continuing further up, more and more semantics are added to the shapes, e.g. circles might be learned to be a wheel. Finally, layers which are close to the top learn feature maps of semantic objects, like a car or a house.

2.1.2 Pooling layer

As discussed in Section 2.1.1, the receptive field of a standard 3×3 kernel matrix is small compared to the size of an input image (e.g. 224×224 pixels). In order to reach a receptive field which covers the entire image, a large number of convolutional layers would be necessary, resulting in a large number of model parameters. Furthermore, applying several different kernels in each layer would increase the number of parameters even further. Such a model would not only suffer from very poor training efficiency, it would also be prone to overfitting. Pooling layers help to reduce the dimensionality of the neural network by applying a kernel to the input layer similar to the convolutional layer. However, instead of a kernel consisting of learned weights, the pooling layer applies an aggregation function to the values in the receptive field. The two main types of pooling



Source: [IBM Cloud Education, 2020]
Figure 2.1: Basic architecture of a convolution layer

layers are **max pooling** and **average pooling** [IBM Cloud Education, 2020]. Where the former method adds the highest value of the receptive field, the latter adds the average of the values in the receptive field to the output array. Pooling layers also apply an activation function, similar to the convolutional layers.

2.1.3 Fully-connected layer

ConvNets used for image classification feature a fully-connected layer at the top of the network, where each node of the output layer connects directly to a node of the previous layer. This is different to convolutional and pooling layers, which only connect to the receptive field and are, therefore, only partially-connected. It performs classification by taking the feature vectors extracted from the previous layers as input and maps them to classification nodes in the output layer [IBM Cloud Education, 2020].

In this project, the ConvNets are used to extract feature vectors of the input images. The classification task itself is performed by trained classifiers (please refer to Chapter 3 to learn about the reasons). Consequently, the fully-connected layer will not be considered any further in this project.

2.2 Inception Architecture

There is an obvious way to increase the performance of ConvNets: increasing their size. The network size can grow in two dimensions, (i) the depth (the number in layers) and (ii) the width of the network (the number of kernels applied at each layer). However, simply increasing the size comes with the cost of an increasing number of parameters. As has been discussed in Section 2.1.2 this can lead to overfitting (especially if only a limited number of training samples are available) and might be computationally prohibitive. Szegedy et al. [Szegedy et al., 2015] introduced the Inception architecture which attempts to overcome the aforementioned problems.

2.2.1 Motivation

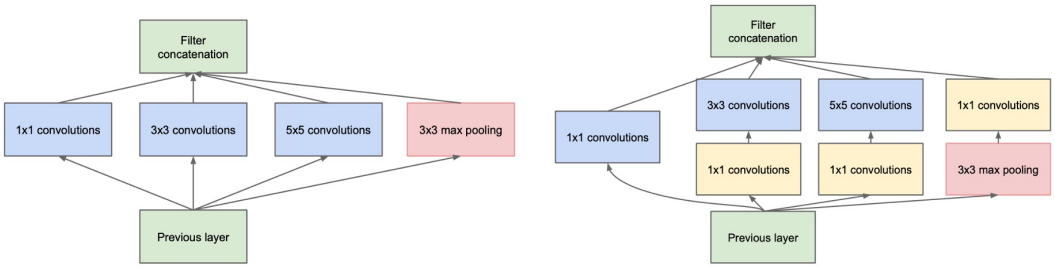
The architecture is inspired by the work of Arora et al. [Arora et al., 2014]. They present a mathematical proof for an optimal network topology, given that the probability distribution of a data set can be represented by a large, sparse deep neural network. It is constructed by analyzing the correlation statistics in the last layer. Then clusters of units with high correlation are built which form the units of the next layer. Finally, the units of the previous and the units of the current layer are connected. This layer-by-layer approach mimics the biological *Hebbian* principle: "Things that fire together wire together" [Hebb, 1949]. Arora et al. rephrase the principal as: "Nodes in the same layer that fire together a lot are likely to be connected (with positive weight) to the same node at the higher layer." [Arora et al., 2014] Szegedy et al. admit that the proof requires strict conditions which are not met with their approach. The proposed architecture tries to approximate an optimal local sparse structure with dense components for which modern computational hardware is optimized.

2.2.2 Layer Module Architecture

The suggested solution clusters groups of nodes with high correlation from the previous layers to form units of the next layer. These units are then connected to the previous layer. To do so, Szegedy et al. assume that "each unit from the earlier layer corresponds to some region of the input image and these units are grouped into filter banks. In the lower layers (the ones close to the input) correlated units would concentrate in local regions. This means, we would end up with a lot of clusters concentrated in a single region and they can be covered by a layer of 1×1 convolutions in the next layer. However, one can also expect that there will be a smaller number of more spatially spread out clusters that can be covered by convolutions over larger patches, and there will be a decreasing number of patches over larger and larger regions."[Szegedy et al., 2015] Following this reasoning, the authors propose to stack single Inception modules as shown in Figure 2.2. A naive inception module consists of three kernel sizes and a max pooling branch which get concatenated as input for the next layer. Intuitively speaking, the three different patch sizes process the visual information at different scales. Then, the next stage uses the aggregated input to abstract features from different scales simultaneously. Another important feature of the Inception architecture is the widespread use of dimension reduction by applying 1×1 convolutions. Figure 2.2 (b) shows a final Inception module. Although, dimension reduction leads to some information loss, it keeps the computational complexity under control and allows for deeper and wider inception networks. [Szegedy et al., 2015]

2.2.3 InceptionV3

For Version 3 of the Inception network (which is used in this thesis) Szegedy et al. introduced various optimizations over version 1. All new introduction were aimed at optimizing the computational efficiency of the Inception architecture. The most notable



(a) Inception module, naïve version (b) Inception module with dimension reductions

Source: [Szegedy et al., 2015]
Figure 2.2: Architecture of a inception module

introductions is the **spatial factorization into asymmetric convolutions**. The authors propose to replace the computationally expensive 5×5 convolution of the original Inception module by two consecutive 1×3 and 3×1 convolutions. Using the same number of filters, the asymmetric approach is more cost efficient and, thus, allows for a bigger network at the same computational cost. [Szegedy et al., 2016b]

2.2.4 Network Architecture

The InceptionV3 network used in this thesis has a depth of 159 layers and consists of 23'851'784 parameters. [Chollet et al., 2015] It achieved a top-1 accuracy of 0.779 and a top-5 accuracy of 0.937 on the ImageNet [Russakovsky et al., 2015] validation data set (see also table A.2 in the appendix, section A.1.5). A more detailed detailed layout of the InceptionV3 network architecture can also be found in the appendix, section A.1.1.

2.3 ResNet Architecture

The ResNet architecture pursues a similar goal as the Inception architecture: to reach even deeper convolutional neural networks. Where the Inception architecture exploits the theoretical findings from Arora et al. and makes widespread use of dimension reduction convolutions to reduce the computational complexity of the network, ResNet introduces a deep residual learning framework.

2.3.1 Motivation

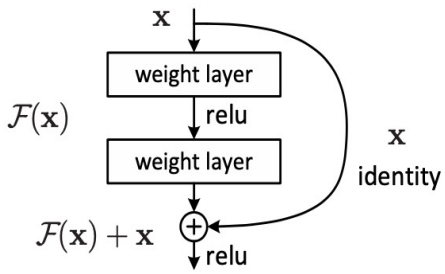
The ResNet architecture is motivated by a peculiar observation for deep neural networks. Many networks exhibit a degradation in accuracy when more layers are added to the network. [He and Sun, 2015] Interestingly, the degradation is not caused by overfitting, but higher training errors. He et al. investigated the problem further and found that when identity mapping layers are added to one of two otherwise identical networks, there is still a degradation in accuracy for the deeper network. Apparently, the solvers where

not able to find a solution to approximate an identity mapping by multiple non-linear layers. [He et al., 2016a]

2.3.2 Layer Module Architecture

He et al. hypothesize that it is easier for a solver to learn a residual mapping $\mathcal{F} := \mathcal{H} - x$ than a underlying mapping function $\mathcal{H}$. They argue that if "one hypothesizes that multiple nonlinear layers can asymptotically approximate complicated functions, then it is equivalent to hypothesize that they can asymptotically approximate the residual functions, i.e., $\mathcal{H} - x$ (assuming that the input and output are of the same dimensions)". [He et al., 2016a] In the aforementioned case of approximating an identity mapping, the solver would simply need to push the weights of the multiple non-linear layers towards zero to find a good solution.

Figure 2.3 shows a residual learning building block. Instead of the desired underlying mapping $\mathcal{H}$, the training algorithm fits the residual mapping function $\mathcal{F} := \mathcal{H} - x$. Figure 2.3 also shows the shortcut connection which allows for the restoration of the desired underlying mapping function by simply adding element-wise the identity x to the output feature maps, channel-by-channel. A residual learning block typically consists of 2 or 3 convolutional layers. He et al. emphasized that the shortcut connection still allows for stochastic gradient descend with back propagation of errors and neither adds extra parameters nor extra complexity to the network.



Source: [He et al., 2016a]
Figure 2.3: Residual learning building block

2.3.3 ResNetV2

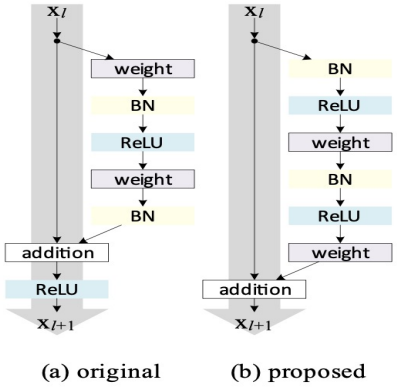
The main contribution of the second version of the ResNet architecture is the redesign of the shortcut path. He et al. propose an identity mapping not only within a residual learning building block but also between blocks. They found that a network which allows for *direct* propagation of information through the entire network is easier to train and generalizes better. This gives room for even deeper networks.

The new building block design is shown in Figure 2.4 b). Compared to the original design, no ReLU activation function is applied after restoring $\mathcal{F} = \mathcal{H} - x$. Instead, the

authors propose a *pre-activation* of the convolutional layers (please note, how the ReLU function is applied before the weight kernel). This is a departure from the conventional *post-activation* of feature maps. However, empirical evidence show that an identity mapping between blocks simplifies training and reduces overfitting. [He et al., 2016b]

2.3.4 Network Architecture

Figure A.4 in the appendix, Section A.1.3, provides a schematic overview of the network topology with 34 parameter layers. Please note, that the ResNetV2 network used in this thesis features 50 parameter layers. Keras [Chollet et al., 2015] actually provides even deeper pre-traind ResNetV2 networks. However, the network with 50 parameter layers features a similar parameter count as the InceptionV3 network (25'613'800 vs 23'851'784). Therefore, it allows for a direct comparison of the performance between the two layer modules architectures. ResNet50V2 achieved a top-1 accuracy of 0.749 and a top-5 accuracy of 0.921 on the ImageNet validation data set (see also table A.2 in the appendix, section A.1.5). [Chollet et al., 2015]



Source: [He et al., 2016b]
Figure 2.4: Residual learning building blocks in ResNet and ResNet V2

2.4 InceptionResNet Architecture

After the successful introduction of the ResNet architecture, Szegedy et al. investigated if their Inception architecture could gain any performance improvements if it was combined with the residual learning approach. In their study, the authors basically used residual learning blocks with *post-activation* architecture and replaced the conventional convolution layers with an Inception module. [Szegedy et al., 2016a]

2.4.1 Layer Module Architecture

The authors used slightly lighter Inception modules compared to original module, in order to roughly match the computational cost of the InceptionV3 network. Other technical adjustments to the Inception module were necessary. For example, a filter expansion layer (1×1 convolution without activation) was introduced after each block to match the dimensionality of the input. This is necessary, because the original Inception module uses dimension reduction to keep the computational complexity under control. However, the restoration of the underlying mapping function $\mathcal{H} = \mathcal{F} + x$ requires the input and output of a residual training block to be of the same dimensionality (see 2.3.2). Szegedy et al. also propose a technical adjustment to the residual learning block. For very deep variants, they found the network to become saturated during training. Also He et al. made this observation. [He et al., 2016a] Scaling down the linear activation by a factor between 0.1 and 0.3 stabilized the network.

In addition to the adjustments on block level, Szegedy et al. [Szegedy et al., 2016a] also tuned the number of filters in the entire network. This optimization improved the training efficiency compared to Inception V3 without affecting the quality of the trained network.

2.4.2 Network Architecture

With 55'873'736 parameters and a depth of 572 layers, the InceptionResNetV2 network is by far the largest network considered in this thesis (see also Table A.2 in the appendix for a size comparison). It also exhibits the highest scores for top-1 and top-5 accuracy on the ImageNet validation data set. [Chollet et al., 2015] However, the marginal gains over the other ConvNets is relatively small compared to the increase in network size. Please refer to the appendix, Section A.1.3, to learn more about the network architecture.

2.5 MobileNet Architecture

The general trend in ConvNet architecture has been to build deeper and wider networks to improve accuracy in vision tasks such as image classification and object recognition. The MobileNet architecture pursues a different goal. It focuses mainly on network efficiency.

2.5.1 Motivation

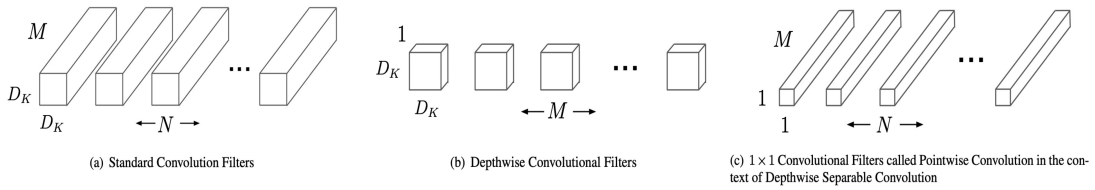
Computer vision is increasingly finding its way into real-world applications. Augmented reality applications on smartphones, situation aware self-driving cars and self navigating robots are just a few examples. Many of these applications run on computationally limited hardware and the recognition task must be executed in real time. The previously discussed architectures, Inception and ResNet, target very deep and wide networks with high detection accuracy. It comes at the cost of comparably high embedding latency

and high memory loads. MobileNet is an efficient network architecture which allows to build small and low latency networks [Howard et al., 2017].

2.5.2 Layer Module Architecture

Similar to the InceptionV3 architecture, MobileNet uses factorization of convolution layers to reduce the model size and, hence, computation. At the core of the MobileNet architecture are depthwise separable convolution layers, which split a standard convolution into a two-step process. The first step is to apply a single filter of size $D_K \times D_K \times 1$ on each input channel $m \in M$ (depthwise convolution), where D_K is the spatial dimension of the kernel. In the second step, a $1 \times 1 \times M$ kernel creates a linear combination over the M output feature maps from the first step. Figure 2.5 illustrates the two approaches. In order to see how depthwise separable convolutions reduce the network size, consider the following comparison: A standard convolution is parameterized by a kernel of size $D_K \times D_K \times M \times N$, where D_K is the spatial dimension of the kernel, M is number of input channels and N the number of output channels. The depthwise separable convolution requires kernels of size $D_K \times D_K \times M$ and $1 \times 1 \times M \times N$, respectively. The reduction in the number parameters is: [Howard et al., 2017]

$$\frac{D_K \cdot D_K \cdot M + 1 \cdot 1 \cdot M \cdot N}{D_K \cdot D_K \cdot M \cdot N} = \frac{1}{N} + \frac{1}{D_K^2} \tag{2.1}$$



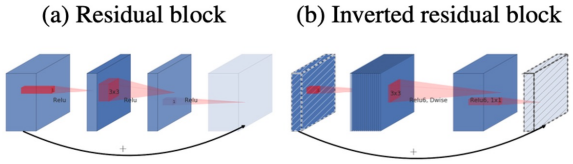
Source: [Howard et al., 2017]

Figure 2.5: A Standard convolution filter (a) is replaced by a depthwise convolution (b) and a pointwise convolution (c)

2.5.3 MobileNetV2

Sandler et al. [Sandler et al., 2018] improved the MobileNet layer module architecture with the introduction of a novel layer module: the inverted residuals with linear bottleneck. A *bottleneck* is a layer with fewer nodes then the layers above or below it. 1×1 convolutions are used to compress high dimensional representations into lower dimension space (see also [Szegedy et al., 2015]). The authors use linear activations after the bottleneck as it prevents non-linearities from destroying too much information. Figure 2.6 b) shows an inverted residual block with linear bottleneck. Please note that (i) the inverted residual block connects the bottlenecks with shortcuts, (ii) the input gets expanded into a higher dimensional space after the first bottleneck and (iii), the

convolutions applied between bottlenecks are conventional ReLU activations instead of the linear activations. Please refer to Figure A.12 in the Appendix A.1.4 to learn more about the specification of the module. According to the authors, this design "provides a natural separation between the input/output domains of the building blocks (bottleneck layers), and the layer transformation – that is a non-linear function that converts input to the output. The former can be seen as the capacity of the network at each layer, whereas the latter as the expressiveness". [Sandler et al., 2018]



Source: [Sandler et al., 2018]
Figure 2.6: A standard residual block in (a) and an inverted residual block in (b)

2.5.4 Network Architecture

The MobileNetV2 network consists of 20 blocks without the fully-connected layer. Out of the 20 blocks, 17 are inverted residual linear bottleneck blocks (please refer to figure A.13 in the appendix, Section A.1.4, to learn more about the details of the network architecture). The full model contains 3'538'984 parameters and features a depth of 88 layers. It achieved a top-1 accuracy of 0.713 and a top-5 accuracy of 0.901 on the ImageNet validation data set (see also Table A.2 in the appendix, Section A.1.5) [Chollet et al., 2015].

3

Classifiers

In the introduction of Chapter 2, the great successes of Convolutional Neutral Networks in image classification competitions, such as the ILSVRC was pointed out. Such contests are usually structured as follows: The teams are provided with a comprehensive training set of images with a fixed number of labeled objects. Furthermore, a validation set and a test set are provided, which contain different images compared to the training set, but identical objects. The winner is chosen based on the classification results on the test set. Since the individual objects are known and fixed in advance, it is advantageous to train the ConvNet using a huge number of training images containing these objects. In this regard, the fully-connected layer (as described in Section 2.1.3) functions as a classifier.

However, the challenge in this thesis is that the number and type of objects are not known in advance. The system is constantly confronted with new objects, which it must learn and identify in unseen images. One possibility would be to retrain the entire ConvNet including the fully-connected layer whenever a user adds newly labeled objects to the system. From a computational perspective, this would be rather challenging. A possible alternative would be to retrain the fully-connected layer only, which is known as transfer learning. However, the number of labeled images might vary significantly between different classes. Buda et al. [Buda et al., 2017] show that the effect of unbalanced data is detrimental on classification performance. A solution to this problem is to train a dedicated classifier for each object class. The ConvNet simply embeds the image into the feature vector space. The feature vectors are then used to train a classifier.

This chapter introduces the different classification methods considered in this thesis. Section 3.1 discusses the Gaussian Naive Bayes classifier and Section 3.2 introduces the k-Nearest Neighbor classifier. In Section 3.3, two related classifier are presented, first the Decision Tree classifier and, second, the Random Forest classifier. Section 3.4 presents the Support Vector Classifiers with a linear kernel and with a radial basis function kernel. The chapter ends with the introduction of the Multi-layer Perceptron classifier in Section 3.5. A detailed discussion about the classification performance of each method follows in Chapter 4.

3.1 Gaussian Naive Bayes

Naive Bayes (NB) is a relatively simple classification technique. The underlying assumption of a NB classifier is that features are conditionally independent from each other. Hence, the joint probability density function (pdf) can be written as the product of the individual conditional probabilities, or more formally:

$$P(\mathbf{X} \mid C) = \prod_{i=1}^{n} P(X_i \mid C),$$

where $\mathbf{X} = (X_1, \ldots, X_n)$ are the extracted features, e.g. the embedded feature of an image in a multidimensional vector space, and C is a a class. The Bayes classifier is then defined as [Rish, 2001]:

$$\begin{aligned} h^*(\mathbf{x}) &= \operatorname*{argmax}_x P(C = i \mid \mathbf{X} = \mathbf{x}) \\ &= \frac{P(\mathbf{X} = \mathbf{x} \mid C = i)P(C = i)}{P(\mathbf{X} = \mathbf{x})} \\ &= P(\mathbf{X} = \mathbf{x} \mid C = i)P(C = i), \end{aligned} \tag{3.1}$$

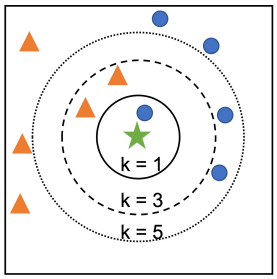
where $P(\mathbf{X} = \mathbf{x})$ can be ignored in Equation 3.1 because it is the same for all classes. A **Gaussian NB** classifier simply assumes that the values associated with each class follow a Gaussian (or normal) distribution. This assumption facilitates training substantially because the class-conditional distribution can be described using only the first two moments.

The assumption of independent features is highly unlikely in real-world applications, which is why it is called *naive*. However, the NB classifier has proven itself despite dependencies among features. [Kelly and Johnson, 2021], [Kamel et al., 2019] Domingos and Pazzani [Domingos and Pazzani, 1997] argue that for an optimal classifier (in terms of classification error) it is more important that the estimated and the actual distribution agree on the same class. The quality of the fit to an assumed probability distribution is of secondary importance. Rish [Rish, 2001] shows that NB classifiers perform best when (i) features are completely independent or for (ii) almost-deterministic (low-entropy) feature dependencies. In between the two extremes, the NB classifiers performs poorly. This supports the empirical finding that NB classifiers perform very well in certain tasks. Combined with their simplicity, fast training time and low memory requirements it makes them the classifier of choice in many applications.

3.2 K-Nearest Neighbor

Another very simple but surprisingly effective classification method is the k-Nearest Neighbor classifier. Fix and Hodges [Fix and Hodges, 1951] first introduced the method. The algorithm classifies samples based on the majority vote of its nearest neighbors and is specified by a single parameter k. Figure 3.1 illustrates the mode of operation of

the classifier. For $k = 1$ the query sample (green star) is assigned to class *Blue*. In case $k = 3$, the majority vote of the three nearest neighbors is *Orange*. For $k = 5$ the assigned class is *Blue* again. Besides the simple specification, the classifier has other desirable properties: (i) the method allows for non-linear decision boundaries, (ii) the prediction quality improves with the number of training samples and (iii) the trivial learning procedure rarely causes overfitting. [Goldberger et al., 2004] The aforementioned benefits are a direct consequence of the classifier being an *instance-based learning* algorithm: it simply stores all training instances and compares the query sample against them. *Instance-based-learners* do not learn an internal model for classification. This has one major drawback: because the algorithm relies on a distances metric between the (multi-dimensional) feature vectors of the query sample and the training samples, it has to go through all training instances to find the right class. For large training sets this can become very expensive, both from a computational and a memory load perspective. Another problem is how to choose the best distance metric for the classification problem at hand. Figure 3.1 uses the *Euclidean* distance with a uniform distance weight to determine the nearest neighbors. For specific classification tasks other distance metrics with non-uniform distance weights might be more useful. However, the best specification cannot be known in advance. [Goldberger et al., 2004]



Source: own depiction

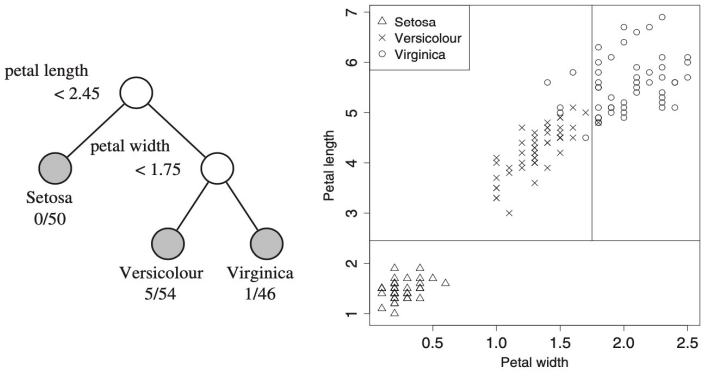
Figure 3.1: k-nearest neighbor classification with $k = 1$, $k = 3$, $k = 5$, distance metric *Euclidean* and uniform distance weight

3.3 Decision Trees

Decision Trees belong to the class of *model-based learning* algorithms. Based on the feature vectors and the class labels of the training set, these algorithms attempt to find a tree representation (model) of the classification problem at hand. A Decision Tree consists of the following components: (i) the terminal nodes (or leaves) of a Decision Tree represent the discrete class labels, (ii) branches, which lead to the leaves, represent conjunctions of input features which ultimately determine the class of a single sample. Breiman et al. [Breiman et al., 1984] popularized the method among statisticians and data scientists with the introduction of the CART algorithm. Figure 3.2 shows a Decision Tree for the Iris data set on the left. The gray terminal nodes are the class labels. The

white intermediate nodes represent individual features of the input data. The condition at which an instance either follows the right or the left branch is also shown for each feature (e.g. if the *petal length* is less than 2.45cm the flower is classified as a *Setosa*).

There exists a variety of different algorithms to construct and train Decision Trees: *Chi-square Automatic Interaction Detectors (CHAID)* by Sonquist and Morgan was the first algorithm for Decision Trees [Sonquist and Morgan, 1964]. Quinlan introduced three incrementally improved algorithms: *ID3* [Quinlan, 1986], *C4.5* [Quinlan, 1993], and *C5.0* which is the most recent variant. Other algorithms are *Multivariant Regression Splines (MARS)* by Friedman [Friedman, 1991] and *CART* by Breiman et al. [Breiman et al., 1984]. These algorithms differ mainly with respect to the applied splitting criteria, pruning method and noise handling [Kohavi and Quinlan, 1999]. In the following sections, we will first take a closer look at the *CART* algorithm. A modified version of it is used in this thesis. [Pedregosa et al., 2011] Then we will discuss the Random Forest classifier which fixes a drawback of simple Decision Trees.



Source: [Loh, 2014]

Figure 3.2: Classification tree model for the iris data set. The numbers beneath each terminal node shows the number of misclassified samples and the node sample size.

3.3.1 CART Decision Tree

The basic strategy of the *CART* algorithm, as most other Decision Tree algorithms, is the recursive partitioning of the feature vector set into subsets. The procedure is induced top-down and continued until the subsets all have the same target value, or splitting does not add any information to the prediction. The CART algorithm uses the *Gini impurity* as splitting criteria and only allows for binary splits in a tree. The splitting criteria is defined as:

$$I_G(p) = \sum_{i=1}^{J} \left(p_i \sum_{k \neq i} p_k \right) = \sum_{i=1}^{J} p_i(1 - p_i) = 1 - \sum_{i=1}^{J} p_i^2 \tag{3.2}$$

where J is the set of classes in the set of training instances and p_i is the fraction of

instances labeled with class i. The *Gini impurity* can be understood as the probability of mislabeling a randomly chosen instance from the training set if the labels where randomly selected according to the distribution of labels in the subset. An optimal split is reached if $I_G(p) = 0$, which means that all instances at a node belong to the same target class.

After growing the tree, *CART* prunes it based on a *minimal cost complexity*. The cost is assigned to each subtree. It assumes that the resubstitution error (which measures the error rate of the modeled tree evaluated on the training set) increases linearly with the number of terminal nodes. Breiman et al. [Breiman et al., 1984] showed that there exists a unique smallest tree which minimizes that cost.

Important advantages of the algorithm are that it can handle both, numerical and categorical features (the former is of importance for this thesis), the cost of predicting data is logarithmic in the number of instances used for training and that it can easily handle outliers. On the downside, learning an optimal Decision Tree is a NP-hard problem and the *CART* algorithm may produce overly complex trees and is prone to overfitting. This issue can be mitigated by defining a maximal tree depth and a minimal number of samples required at the terminal nodes. The algorithm may also produce unstable Decision Trees. Already small changes in the training set might result in a different tree. Using a Decision Tree within an ensemble can reduce this problem. The Random Forest classifier does exactly that. [Pedregosa et al., 2011]

3.3.2 Random Forest

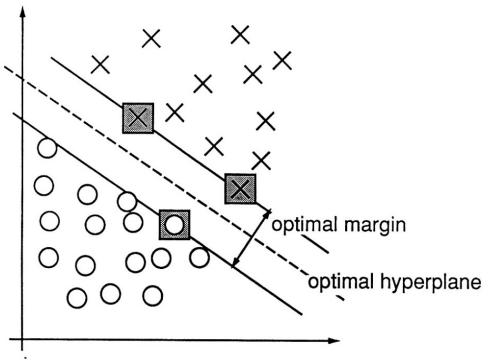
The Random Forest classifier was popularized by Breiman in 2001 [Breiman, 2001]. It is a method to construct a classification ensemble consisting of a collection of tree-structured classifiers. Its main idea is to grow a set of Decision Trees in randomly selected subspaces of the training features. To classify sample x, each Decision Tree casts one vote, and the class that receives the majority of votes is assigned to sample x. Breiman explains the creation of a random forest as follows: "The simplest random forest with random features is formed by selecting at random, at each node, a small group of input variables to split on. Grow the tree using CART methodology to maximum size and do not prune." [Breiman, 2001]

Two advantages of Random Forest classifiers are based on the fact that they consist of a set of individual Decision Trees. First, due to majority voting, Random Forest classifiers are much more stable than simple Decision Trees. Second, each Decision Tree can be grown independently from each other. This allows for easy parallelization of the compute process and makes it a highly efficient classifier even for large data sets. However, Xu et al. [Xu et al., 2012] mention a potential drawback, especially when used with high dimensional input data, like images. They argue that through randomly selecting a small subspace from the high dimensional feature input space it is likely that *uninformative* features (with regard to the class label) get selected. If the resulting Random Forest consist of a large portion of such *uninformed* trees, it is likely that the classifier suffers from poor classification power.

3.4 Support Vector Machine

Vladimir Vapnik, in collaboration with various colleagues, is considered the inventor of the Support Vector Machine (SVM), which has had a major impact on machine learning ever since. In 1993, Boser et al. [Boser et al., 1992] proposed a training algorithm which extended the original idea and solved a general problem in machine learning. The authors describe the problem as follows: "Good generalization performance is achieved when the capacity of the classifier function is matched to the size of the training set. Classifiers with a large number of adjustable parameters and therefore large capacity likely learn the training set without errors, but exhibit poor generalization. Conversely, a classifier with insufficient capacity might not be able to learn the task at all. In between, there is an optimal capacity of the classifier which minimizes the expected generalization error for a given amount of training data."[Boser et al., 1992] The idea behind their algorithm is to maximizes the margin between training examples and class boundary. To do so, only a small subset of the training data is usually required, the so-called *support vectors*. Figure 3.3 illustrates the idea. More formally, the algorithm constructs a *hyperplane* in a high-dimensional space with optimal margin to the training samples. For class prediction, it is sufficient to check on which side of the hyperplane the test sample falls.

SVM offers both, a linear and a non-linear variant for classification. The linear variant of the SVM is briefly discussed in Section 3.4.2 and a non-linear variant in Section 3.4.3. We start with the problem definition in Section 3.4.1.



Source: [Cortes and Vapnik, 1995]

Figure 3.3: An example of a separable problem in a 2-dimensional space. The support vectors, marked with grey squares, define the margin of largest separation between the two classes.

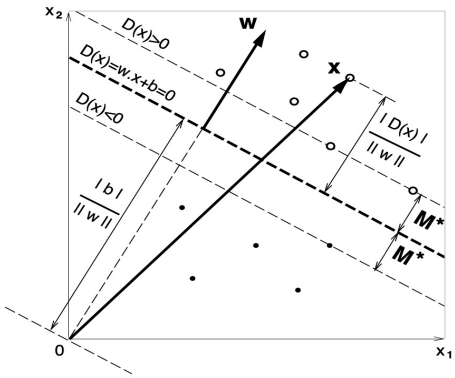
3.4.1 Problem Definition

The goal of the algorithm from Boser et al. [Boser et al., 1992] is to find a decision function $D(\mathbf{x})$ for the n-dimensional feature vectors $\mathbf{x}$ in the training set. The training set consists of p samples with labels $\mathbf{y}$, where y_i is either 1 or -1. A sample x_i is of class

1 if $D(x) > 0$ and of class -1 otherwise. Formally, the decision function, which defines a separating hyperplane in $\boldsymbol{\varphi}$-space, is given by:

$$D(\mathbf{x}) = \mathbf{w} \cdot \boldsymbol{\varphi}(\mathbf{x}) + b, \tag{3.3}$$

where $\mathbf{w}$ is an n-dimensional normal vector to the hyperplane and b is a bias. Both, $\mathbf{w}$ and b are adjustable parameters of the decision function. $\boldsymbol{\varphi}$ is a predefined function of $\mathbf{x}$. Figure 3.4 illustrates the decision function in a $(\boldsymbol{\varphi} = \mathbf{x})$-dimensional space. If the two classes can be separated by a hyperplane with margin M then the following inequality applies to all training samples:



Source: [Boser et al., 1992]
Figure 3.4: Decision function $D(\mathbf{x}) = \mathbf{w} \cdot \boldsymbol{\varphi}(\mathbf{x}) + b$ with $\boldsymbol{\varphi} = \mathbf{x}$

$$\frac{y_k D(\mathbf{x}_k)}{\|\mathbf{w}\|} \geq M, \tag{3.4}$$

where $D(\mathbf{x}_k)/\|\mathbf{w}\|$ is the distance between the hyperplane and the sample $\mathbf{x}_k$ (see Figure 3.4). The objective function of the algorithm is therefore:

$$\begin{aligned} M^* &= \max_{\mathbf{w}, \|\mathbf{w}\|=1} M \\ \text{subject to} & \\ & y_k D(\mathbf{x}_k) \geq M, \text{ for } k = 1, \ldots, p \end{aligned} \tag{3.5}$$

The samples with the smallest margin are called support vectors. The bound M^* in Figure 3.4 is defined by these support vectors:

$$M^* = \min_k y_k D(\mathbf{x}_k) \tag{3.6}$$

The optimization problem can be rewritten as a minimax-problem by plugging Equation 3.6 into in Equation 3.5. Furthermore, instead of fixing the norm $\|\mathbf{w}\| = \mathbf{1}$ in Equation 3.5 the authors fix $M \cdot \|\mathbf{w}\| = \mathbf{1}$. Consequently, maximizing M is equivalent to minimizing the norm $\|\mathbf{w}\|$ and the optimization problem simplifies to:

$$\begin{aligned} &\min_{\mathbf{w}} \|\mathbf{w}\|^2 \\ \text{subject to} \quad & \\ &y_k D(\mathbf{x}_k) \geq 1, \text{ for } k = 1, \ldots, p \end{aligned} \tag{3.7}$$

and the maximum margin is $M^* = 1/\|\mathbf{w}^*\|$.
In principal, Equation 3.7 can be solved numerically. However, this can become impractical for high-dimensional $\boldsymbol{\varphi}$-spaces. To solve this issue, the authors transform the maximization problem into the *dual-space*. For the purpose of this section, this is considered out-of-scope. Please refer to the paper [Boser et al., 1992] to learn more about it.

3.4.2 Linear SVC

The simplest variant is the linear Support Vector Classifier (SVC) with $\boldsymbol{\varphi} = \mathbf{x}$. It is depicted in Figure 3.4. The decision function is defined by:

$$D(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} - b, \tag{3.8}$$

In order to get the maximum margin hyperplane, the algorithm solves Equation 3.7 subject to the decision function defined in 3.8.

3.4.3 Radial Basis Function SVC

The beauty of the algorithm by Boser et al. [Boser et al., 1992] is that it also allows for classification of non-linearly separable data sets. To do so, the authors take advantage of the *kernel trick*. The idea behind this method is to transform the data into a higher dimensional space, in which a linear discrimination between classes is possible. The *kernel trick* is a well established method and kernels for different non-linear functions exists. Among others, kernels are known for polynomials, hyperbolic tangents and the Gaussian radial basis function (RBF). The exact workings of the *kernel trick* is beyond the scope of this section. However, a SVC with a RBF kernel is used as a classifier in this thesis. The RBF kernel is defined as:

$$K(\mathbf{x}, \mathbf{x}') = \exp\Big(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\Big),$$

and corresponds to a radial basis expansion of $\boldsymbol{\varphi}(\mathbf{x})$.

3.4.4 Properties of SVCs

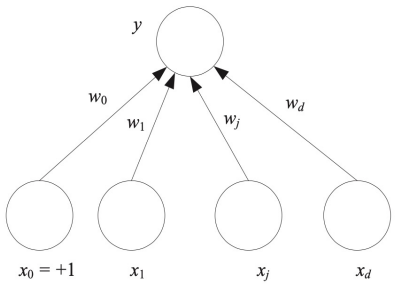
Due to definition of the algorithm, as described above, SVCs are highly effective in high dimensional space and are still effective if the number of feature dimensions outnumbers the training sample size. These two properties makes it a very suitable choice for the image classification tasks at hand. Furthermore, it is very memory efficient as only a subset of training sample is required to form the decision function.

3.5 Multi-Layer Perceptron Classifier

A Multi-Layer Perceptron (MLP) can be understood as a basic neural network. The research foundations were laid by Rosenblatt [Rosenblatt, 1958]. Inspired by the neurons in a human brain, he proposed a a single neuron for classification, the *perceptron*.

3.5.1 Perceptron

It is the basic processing element of every neural network. In Figure 3.5 x_j, $j = 1, \ldots, d$ are the input units, e.g. an input feature vector or the outputs of other perceptrons. Each input is associated with a connection weight w_j. The output y of a simple perceptron is given by the weighted sum of all inputs plus an intercept value x_0 (also called *bias*): $y = \mathbf{w}^T \cdot \mathbf{x}$. The perceptron with several inputs (as shown in Figure 3.5) defines a hyperplane and can be used as a linear discriminant function to separate to two classes (similar to the Linear SVC in Section 3.4.2). In other words, perceptrons are just a way to implement a hyperplane. [Alpaydin, 2014]



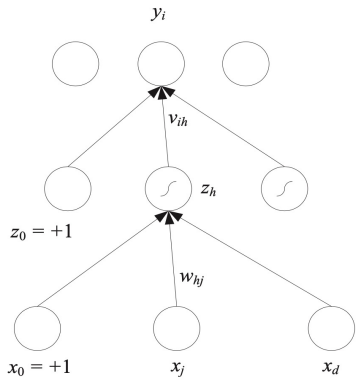
Source: [Alpaydin, 2014]
Figure 3.5: Depiction of a simple perceptron

3.5.2 Multi-Layer Perceptron

As discussed above, simple perceptrons can only approximate linear discrimination functions. In order to classify input with non-linear separable features, additional *hidden* layers can be added. Figure 3.6 shows the architecture of an MLP, where the nodes z_h, $h = 1, \ldots H$, are the hidden units. MLP are fully connected networks, so every hidden node receives an input from each input node and is connected to each node in the next level. The value of z_h is defined by:

$$z_h = \phi(\mathbf{x_h^T} \cdot \mathbf{x}),$$

where ϕ is an activation function (see also 2.1.1 to learn more about activation functions). The output y_i are then given by: $y_i = \mathbf{v_i^T} \cdot \mathbf{z})$, where v_i are the weights.



Source: [Alpaydin, 2014]
Figure 3.6: Depiction of a multi-layer perceptron

3.5.3 Training of MLPs

MLP are trained in a supervised fashion using back propagation of errors. In the example of a three layer MLP with one input layer, a hidden layer and an output layer and for classification with two classes, the error in an output node j for a training example with index t is defined by,

$$E^t(\mathbf{v} \mid \mathbf{z}^t, \mathbf{r}^t) = -r^t \log y^t - (1 - r^t) \log(1 - y^t)$$

where r^t is the target value at the output node and y^t is the value produced by the perceptron. [Alpaydin, 2014] Using *stochastic gradient descent* the weights of each hidden node are changed according to the update rule:

$$\Delta v_h^t = \eta(r^t - y^t) z_h^t, \text{ for } h = 0, \ldots, H \tag{3.9}$$

In Equation 3.9 η is a learning factor and gradually decreases in time for convergence. In order to calculate the input-layer weights, $w_{h,j}$, we can simply propagate the errors from the hidden layer backward using the chain rule:

$$\frac{\partial E}{\partial w_{h,j}} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial z_h} \frac{\partial z_h}{\partial w_{h,j}}.$$

3.5.4 Properties of MLPs

As has been discussed in Section 3.5.2, MLPs can learn to discriminate non-linear separable data sets. They are capable to learn *on-line*, meaning the network updates is parameters with each training instance, one by one. MLPs *can* delivery good prediction accuracy already with only a small training sample set. However, MLP requires tuning of several hyperparameters, such as the network size (number of hidden nodes and layers). The optimal specification depends on the classification task and the most common approach is to test many variations of network sizes. [Alpaydin, 2014] Furthermore, the

hidden layers have non-convex loss functions. Depending on the initial weight initialization, stochastic gradient descent may result in different local minimum which in return can result in different validation accuracy [Pedregosa et al., 2011].

4

Preliminary Evaluation

This chapter reports the classification performance results for different combinations of feature extraction models and classifiers. Section 4.1 formulates 5 research question to guide the evaluation. Section 4.2 introduces the data sets used in the evaluation and Section 4.3 explains the evaluation setup. The chapter finishes with the evaluation results in Section 4.4.

4.1 Research Questions

This chapter aims to answer the following research questions:

- **RQ1:** Which combination of feature vector embedding model and classifier performs best for the task of image classification?
- **RQ2:** How does the classification performance depend on the number of training instances?
- **RQ3:** How good is the classification performance for a data set containing non-iconic images?
- **RQ4:** How good is the classification system in distinguishing subtypes of an object category?
- **RQ5:** Does the classification performance improve if (in addition to the labeled images) the system is also trained on extracted box images of the objects?

4.2 Data Sets

In order to answer the research questions stated above, 2 different data sets were used. Section 4.2.1 introduces the COCO data set and Section 4.2.2 explains the tf_flowers data set.

4.2.1 COCO Data Sets

Microsoft common object in context (COCO) [Lin et al., 2014] is a large-scale data set introduced in 2014 as part of a new object detection and image captioning challenge. Since its introduction, other tasks have been added to the annual competition, like keypoint detection or dense pose tasks. The data set used in this thesis was released in 2014. It was specifically designed to address three research problems: (i) detecting objects from non-iconic views, (ii) precise localization and segmentation of objects in 2D and (iii) contextual reasoning between objects. For the task at hand, the detection of objects from non-iconic views is of particular relevance. It means that an object is shown in its natural context and might be in the background or partially obscured. Figure 4.1 c) shows some examples of non-iconic images.

The data set consists of, among others, a training split and a validation split, containing 82'783 and 40'504 images, respectively. The images are labeled with 80 different objects categories, like person, motorcycle, giraffe, knife, toilet or hair drier, just to name a few. Furthermore, the data set also contains separate bounding boxes for each object instance. To answer research question 4, the bounding boxes were used to extract the individual objects from the training images. The extracted box images where then fed to the classifiers as additional training instances (please refer to Section 4.4.2 to learn more about the results). The data set was downloaded from Tensorflow Hub [Abadi et al., 2015] in a *TFRecord* format.

Source: [Lin et al., 2014]
Figure 4.1: Examples of different images types in the COCO data set

4.2.2 TF_Flowers Data Sets

The tf_flowers data set is also provided by Tensorflow [The TensorFlow Team, 2019]. It contains 5 different types of flowers: dandelion, daisy, tulips, sunflowers, roses. The data set consist of 3670 images with roughly 700 to 750 images of each flower type. In most images, the flowers are depicted in an iconic view, centered and unobscured. The data set does not provide any bounding boxes. Therefore, the data set is primarily used to answer research question 4. As the objects are mainly depicted in an iconic view, the classification results reported in Section 4.4.3 can also be understood as a base against

which the results of the the COCO data set can be compared. Hence, it also helps to answer research question 3.

Figure 4.2: Examples of the tf_flowers data set

4.3 Evaluation Setup

This sections outlines the test setup. Section 4.3.1 gives additional information about the feature extraction models used for this evaluation. Section 4.3.2 briefly discusses the parameterization of the classifiers and Section 4.3.3 explains how the training and validation sets were created.

The entire evaluation was run on a MacBook Pro with macOS 10.15.7, an Intel Core i7 processor and 8 GB of RAM. No dedicated GPU resources were assigned to the task. The code is written in Python, version 3.8, with extensive use of the Tensorflow package, version 2.4.1, and classifiers from Scikit-learn, version 0.24.1 [Pedregosa et al., 2011].

4.3.1 Feature Extraction Models

For all architectures discussed in Chapter 3, Tensorflow Hub provides pre-trained feature extraction models from Keras [Chollet et al., 2015]. They were trained on the *ILSVRC-2012-CLS* data set, which contains 1,281,167 training images and spans 1000 object classes [Russakovsky et al., 2015]. The embedded feature vectors of the images were extracted right below the full-connected classification layer. The architectures differ significantly in layer depth and width (please refer to Table A.2 in the Appendix). The same is true for the output feature vector sizes. Table 4.1 provides an overview. However, the architecture of the second-to-last layer determines the output layer size. Therefore, a higher parameter count does not necessarily translate to a large feature output vector.

Model	**Image input size**	**Output feature vector size**
InceptionV3	224 x 224 x 3	51200
ResNet50V2	224 x 224 x 3	100352
InceptionResNetV2	224 x 224 x 3	38400
MobileNetV2	224 x 224 x 3	62720

Table 4.1: Input and output sizes of the pre-trained models from Keras

4.3.2 Classifiers

The classifiers were obtained from Scikit-learn [Pedregosa et al., 2011]. The parameterization of the classifiers corresponded mainly to the default settings. In order to control the classification results over several runs, the initialization of the random state was set to 0 where necessary. Please be aware that this might cause the stochastic gradient descent to result in a non-optimal local minimum for the MLP classifier. Table 4.2 provides an overview of important model parameters

Classifier	**Important model parameterization**
GaussianNB	-
K-Nearest Neighbor	k = 5; uniform weights; Euclidean distance
Decision Tree	Gini impurity criterion; unlimited max depth
Random Forest	nr. trees = 100; Gini impurity criterion; unlimited max depth
Linear SVC	kernel = linear; optimization problem in dual space
SVC	kernel = rbf; optimization problem in dual space
MLP	hidden layer size = 100, activation = ReLU

Table 4.2: Classifier parameterization

4.3.3 Training and Validation Sets

The aim of this thesis is to study the performance of an active learning image classification system. As will be discussed in more details in Chapter 5, active learning requires a user to feed new object categories to the system. Because annotating large numbers of images is extremely time consuming, it is of high importance that the classification system performs reasonably well even for small numbers of training instances. In order to answer research question 2, I incrementally increased the number of positive training instances in a pseudo-logarithmic way, starting from 1: $n_{i,j} \in \{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000\}$, where n_i is the number of training instances and j is an object category. All images contained in n_i are also part of n_{i+1}. Validation was always performed on the total number of positive validation instances (capped at 1000 if more were available).

COCO

In order to keep the computing time within limits, 10 object categories were sub-sampled from the 80 categories available in the COCO data set. The selection of sub-sampled categories is shown in Table 4.3. Please note that for certain categories insufficient instances were available to run the entire test schedule. Training of the classifiers was carried out with strictly balanced training set: For every randomly sampled positive training instance, a randomly sampled negative training instance was added to the training set. The same applies to the validation set.

Category	Positive training	Positive validation
Person	1000	1000
Elephant	1000	714
Zebra	1000	677
Knife	1000	1000
Carrot	1000	578
Remote	1000	1000
Toaster	151	74
Vase	1000	1000
Scissors	673	302
Toothbrush	700	431

Table 4.3: Available positive training and validation instances per category in the COCO data set (capped at 1000 if the data set contains more)

COCOBox

For the COCOBox data set, the individual annotated objects are extracted from each COCO training image using the object bounding boxes provided by the COCO data set. Each extracted box object is treated as a separate image and added to the training set as an additional training instance. Let us consider an example, were the system is trained on 1 training image of object category *elephants*: Let say we have three elephants (and three bounding boxes) in the training image. The three elephants are extracted from the image and added to the training set as three separate images. The resulting training set consists of 4 positive training instances of category elephant. To keep the training set balanced, three extracted box images that do not contain elephants are also added as negative training instances. In total, the training set consist of 8 images: 2 original images and 6 extracted box images.

The aim of this training set is to understand whether the classification performance can be improved, if the system is additionally trained on iconic-views of the object.

TF_Flowers

The images in the tf_flowers data set were randomly assigned to the training set (2600 images) and the validation set (1000 images). Table 4.4 reports the available numbers for each flower category. For evaluation all training and validation instances were used. Since only a limited number of images were available, the test run with 1000 training instances was waived.

4.3.4 Evaluation Procedure

For the evaluation of the classification system a separate classifier for each combination of feature extraction model, classifier, object category and training sample count was

Category	Positive training	Positive validation
Dandelions	631	267
Daisy	475	158
Sunflower	591	208
Tulips	513	186
Roses	460	181

Table 4.4: Available positive training and validation instances per category in the COCO data set (capped at 1000 if the data set contains more)

trained. The test data set was then classified in a binary fashion: The test image either contains the trained object or it does not.

4.3.5 Performance Metrics

For the evaluation of the classification performance, I mainly focus on two metrics: **Accuracy** and **F1**. Accuracy is the number of true predictions over all predictions. More formally, it is defined as:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}, \tag{4.1}$$

where TP are the number of true positives, TN the true negatives, FP the false positive and FN the false negative predictions of the classification system. Accuracy can be a problematic metric for highly imbalanced data sets, where the distribution of condition positive and condition negative examples in the validation set is not equal. However, for this evaluation a balanced validation set was used (see Section 4.3.3). The second performance metric used is F1. It is the harmonic mean of Precision and Recall:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}, \tag{4.2}$$

where $Precision = TP/(TP+FP)$ and $Recall = TP/(TP+FN)$. Precision measures the proportion of true positives out of all positive classifications and Recall quantifies the number of correctly positively classified examples out of all positive classifications. Or in other words: For an object classification system, if an image is classified to contain an object, Precision measures if the image really contains the object and Recall indicates whether the system is capable to capture as many images containing the object as possible. The F1 score balances the two targets.

Accuracy and F1 scores where calculated for each combination of feature extraction model, classifier, object category and training sample count. However, to get a better understanding of the overall performance of a feature extraction model and classifier combination, the performance metrics had be aggregated over the different object categories. Instead of calculating Accuracy and F1 based on Equations 4.1 and 4.2 over all object categories, the weighted average over all object categories is considered. The

weight is determined by the number of positive validation samples for each category (see also Table 4.3).

$$Accuracy\ \ Weighted = \frac{\sum_j (Accuracy_j \cdot v_j)}{\sum_j v_j}, \tag{4.3}$$

where j is the object category and v_j is the number of validation samples per object category. Correspondingly, the weighted F1 scores is defined as:

$$F1\ \ Weighted = \frac{\sum_j (F1_j \cdot v_j)}{\sum_j v_j}. \tag{4.4}$$

4.4 Classification Results

The following section briefly discusses the classification results. Section 4.4.1 and 4.4.3 discusses the results on the COCO data set and the tf_flowers data set, respectively. Section 4.4.2 reports the classification results on the COCOBox data set. Section 4.4.4 concludes this chapter with a discussion of the research questions introduced in Section 4.1.

4.4.1 Results COCO Data Set

Figure 4.3 shows weighted F1 results over all object categories for each vector extraction model. The weighted Accuracy results are reported in the Appendix, Section A.2.1, Figure A.14. The support vector machines with linear kernel (LinearSVC) performs consistently well on all feature extraction models. It performs best on lower numbers of training samples. SVC has a slight advantage when training is done with larger training samples (> 100). Another classifier which perform reasonably well for larger training sets is the RandomForest classifier. GaussianNB and the MLP classifier show very mixed results and seem to be sensitive to the composition of the training sets. Please note that the KNeighbors classifier requires at least 5 training samples. Therefore, no results are reported if the training set contains fewer instances than 5. Another interesting finding is that the relatively small MobileNetV2 architecture performs much better with the LinearSVC than the InceptionV3 architecture for small training sets (< 10).

Figure 4.4 plots the F1 classification scores for selected classifiers. Again for small training sets, the LinearSVC seems to work well in combination with the ResNet50V2 architecture. For larger training samples (> 20) the different architectures seem to have only a limited influence on the classification performance, except for the MLP classifier. The Accuracy plots can be found in the Appendix, Section A.2.1, Figure A.15.

However, for the classification task at hand, not only the overall performance of the classification system is of importance. The classification system should specifically show good results for detecting objects of a particular category. Table 4.5 reports selected Accuracy scores for the feature extraction model InceptionV3 and a training set size of 5 images. For most object categories, the LinearSVC performs best: *Zebra*, *Knife*, *Carrot*,

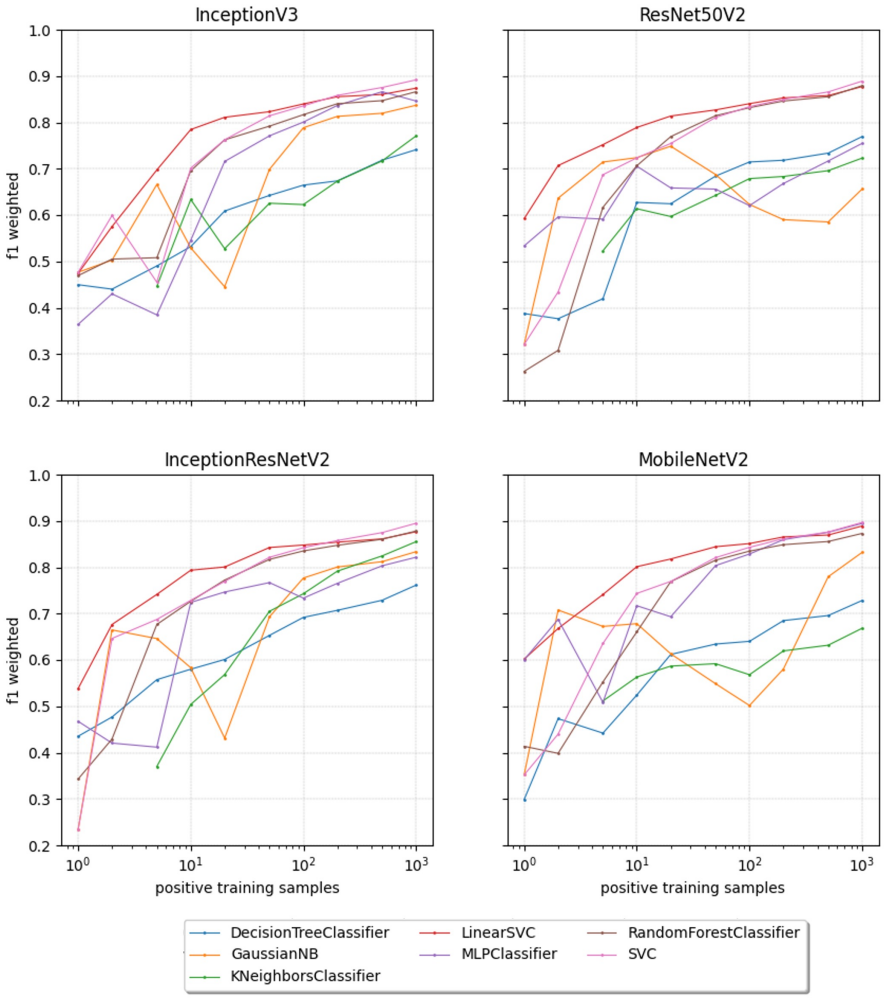


Figure 4.3: Weighted F1 scores for each feature extraction model for the COCO data set

Remote, *Vase* and *Toothbrush*. The GaussianNB classifier performed best for *Person* and *Scissors* and MLP performed best for *Elephant* and *Toaster*. Even though the overall performance of the RandomForest classifier is much better than the GaussianNB and the MLP classifiers, it could not reach best classification performance for any of the categories. Similar results can be found for other combination of feature extraction models, classifiers and training set sizes. Consequently, research question 1 (see **RQ1** in Section 4.1) cannot be answered conclusively. Table 4.5 also impressively shows the variation of classification performance for different object categories. For many object types, the Accuracy score is between 0.8 and 0.95, even for a small training set of only 5 images. For other categories such as *Person* or *Remote*, Accuracy can be reported as low as 0.58 and 0.70, respectively.

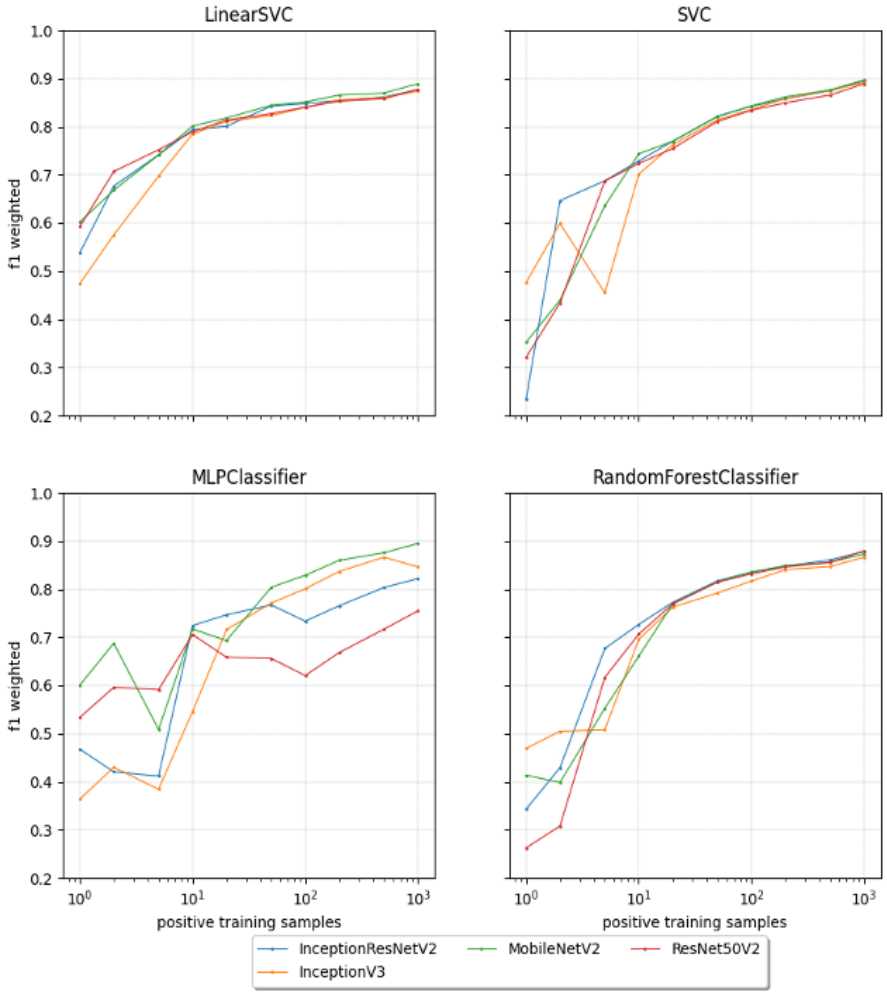


Figure 4.4: Weighted F1 scores for selected classifiers for the COCO data set

4.4.2 Results COCOBox Data Set

Adding extracted object box images to the training set yields interesting results. Figure 4.5 plots the weighted F1 scores for selected classifiers. The plots of the F1 scores for each feature extraction model and the plots with the weighted Accuracy scores are depicted in the Appendix, Section A.2.2. Please note, that the reported number of positive training samples only includes the original images. The box objects extracted form these images and added to the training set are not included. First, the classification performance peaks at around 100 training instances and declines afterwards. This pattern is particularly pronounced for the F1 score (see Figures A.18 in the appendix for comparison). It is a strong indication that classifiers tend to overfit if they are trained on too many box images. Second, adding box images does not seem to improve the classification accuracy at all. The weighted Accuracy scores are lower for all training set sizes compared to the

Category	GaussianNB	LinearSVC	RandomForest	MLP
Person	0.58	0.56	0.53	0.50
Elephant	0.82	0.93	0.79	0.94
Zebra	0.85	0.95	0.55	0.51
Knife	0.57	0.80	0.65	0.52
Carrot	0.75	0.83	0.70	0.65
Remote	0.57	0.70	0.58	0.50
Toaster	0.21	0.82	0.88	0.93
Vase	0.73	0.79	0.68	0.50
Scissors	0.74	0.67	0.53	0.32
Toothbrush	0.38	0.85	0.71	0.77

Table 4.5: Accuracy for feature extraction model *InceptionV3* and 5 positive training instances and selected classifiers

scores on the COCO data set (see Figures A.16 and A.18). Third, adding box images to the training set seems to improve the F1 scores at least for very small training set sizes of 1 or 2 images. This can be observed for all combination of feature extraction models and classifiers (see Figures 4.5 and A.17 in the appendix). Hence, it might be a good strategy to extract box images if only very few images of an object are available.

4.4.3 Results TF_Flowers Data Set

In general, the classification performance scores for the tf_flowers data set show similar patterns as for the COCO data set. Figure 4.6 shows the weighted F1 scores for each embedding model. The plots for the weighted Accuracy scores as well as the plots for the F1 scores for selected classifiers are available in the appendix, Section A.2.3. Weighted Accuracy is even better for the tf_flowers data set than for the COCO data set. The best performing classifier reaches Accuracy scores of 0.9 or above for all feature extraction models if trained with a large training set ($>$ 100 images) (see Figure A.19). Overall the LinearSVC, SVC and the RandomForest classifier perform comparably well again. The MLP and GaussianNB classifier deliver very interesting results: For very small training sets (2 images), the two classifiers seem two work very well in combination with the MobileNetV2 architecture. However, for larger sets, MLP shows mixed results and GaussianNB has a tendency for overfitting. Even more interesting is the fact that the weighted F1 scores are considerably lower for the tf_flowers data set than for the COCO data set. The partially nuanced distinction between the different types of flowers seems to be problematic especially for smaller training sets. The F1 scores increase steadily with an increasing number of training instances, whereas for the COCO data set, the curve runs flatter for more than 10 training instances. There is strong indication, that in order to distinguish subtypes of an object category, a much higher number of training instances is required even if the objects are mainly depicted in an iconic view. This

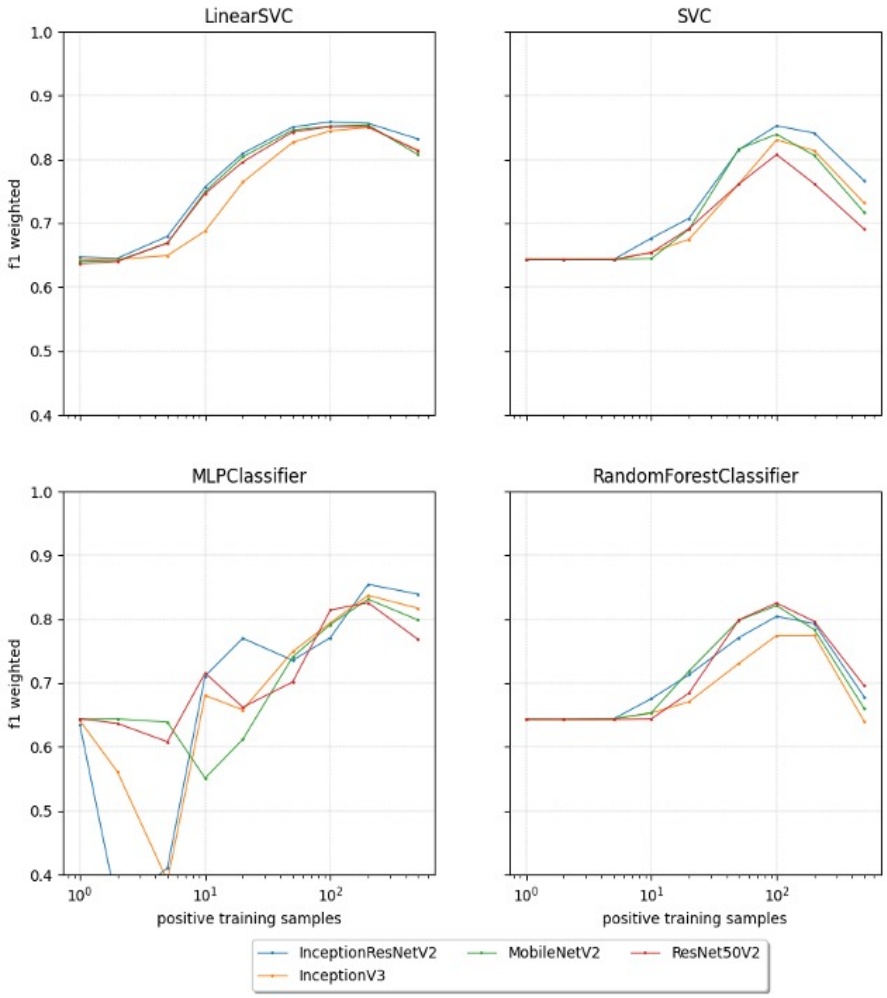


Figure 4.5: Weighted F1 scores for selected classifiers for the COCOBox data set

answers research question 4.

4.4.4 Discussion

The evaluation results discussed earlier allow some conclusions to be drawn about the research questions listed in Section 4.1. First, there is not a single combination of a feature extraction model and a classifier, which performs best for all categories tested in this evaluation (**RQ1**). Second, the classification performance increases steadily with an increasing number of training instances for most combinations of feature extraction model and classifier. Only adding too many extracted box images leads to overfitting. However, the biggest marginal gains in classification performance can be made between 1 and 50 training images. After 50 images the curves flatten out quite a bit (**RQ2**). Third, **RQ3** cannot be answered based on the results gained from these evaluation runs.

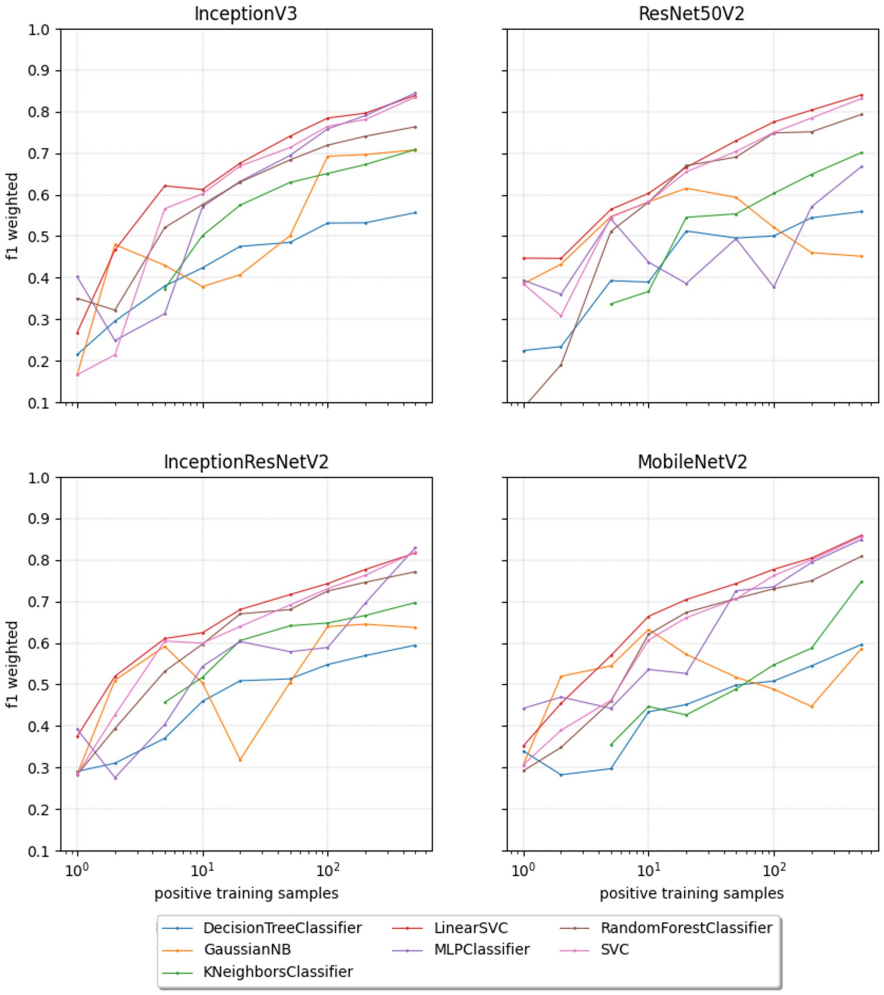


Figure 4.6: Weighted F1 scores for each feature extraction model for the tf_flowers data set

The weighted Accuracy scores are much larger for the tf_flowers data set (which consist mainly of iconic-view images) compared to the COCO data set. However, the weighted F1 scores are lower. Forth, distinguishing subtle subtypes of object categories requires larger numbers of training samples (**RQ4**). This conclusion can be drawn from the evaluation on the tf_flowers data set. Last, adding extracted box images to the training set might improve the classification performance if only very few training images of an object are available *RQ5*.

5

Active Learning

The following Chapter discusses active learning in the context of the classification problem at hand. The Chapter starts with a problem definition in Section 5.1. Section 5.2 introduces different active learning methods: It briefly discusses active learning methods for ConvNets in Section 5.2.1 and more importantly, active learning methods for classifiers considered in this thesis in Section 5.2.2. Section 5.2.3 draws a brief conclusion about human machine interaction and Section 5.2.4 provides some preliminary design requirements for an active learning loop. The final pipeline design is introduces in the next chapter.

5.1 Problem Definition

Traditionally, supervised learning methods requires large amounts of annotated data. Important image classification challenges like the Large Scale Visual Recognition Challenge (ILSVRC) [Russakovsky et al., 2015] or the COCO challenge [Lin et al., 2014] provide large annotated data sets on which new network architectures can be trained and tested, resulting in systems with very impressive classification and object detection results. However, annotating large image sets is not only a tedious work, it is also very expensive. This is especially true for annotation tasks which require experts, such as annotating radiology images of cancer patients or Covid-19 infected lungs. Extensive research has been conducted to develop methods which help to reduces the amount labeled data to achieve good classification results. Active learning is a branch which attracted much attention in the last years. Succinctly put, active learning is about minimizing human effort by selecting those instances for annotation which are most informative to the classifier.

5.2 Active Learning Methods

Three main selection strategies can be distinguished: (i) membership-query synthesis, (ii) stream-based selection and (iii) pool-based selection.
The membership-query synthesis was introduced by D. Angulin [Angluin, 1988]. In this approach the system itself generates the instances that require a label. Zhang et al.

[Zhang et al., 2020] indicate that the strategy achieves good results in some domains. However, they also refer to a scientific study by Baum and Lang [Baum and Lang, 1992] which shows that query instances generated by a model are not suitable when people are asked for annotations. Often, the model generates feature patterns which are unrecognizable to humans. In the stream-based approach, the system decides sequentially for each instance whether a label is required by a (human) oracle. In a pool-based approach, the entire unannotated data set is evaluated and ranked according to its informativeness [Zhang et al., 2020]. It is the most widely used method in the field of active learning research. Different methods to determine the informativeness of an instance have been proposed.

5.2.1 Active Learning Methods for ConvNets

Hemmer et al. [Hemmer et al., 2020] indicate that many traditional active learning methods struggle to deal with high-dimensional data, such as images. Therefore, researchers developed methods specifically for image classification with ConvNets. Among the pool-based approaches, two main categories can be distinguished: *diversity-based* approaches and *uncertainty-based* approaches. The former aims to select a set of images for annotation that best represents the pool of unannotated images. The latter follows the intuition that the more *uncertain* the model is about a prediction, the more informative it is for training. Sener and Savarese [Sener and Savarese, 2018] propose a diversity-based, core-selection approach which minimizes the Euclidean distance in the feature vector space between the sets of selected and non-selected images. Hemmer et al. point out an important disadvantage of such a distance-based approach: distance metrics can concentrate in high-dimensional spaces and make the distances between different elements appear to be similar. Recent uncertainty-based approaches where proposed by Wang et al. [Wang et al., 2016], Gal et al. [Gal et al., 2017], Beluch et al. [Beluch et al., 2018], Yoo and Kweon [Yoo and Kweon, 2019] and Hemmer et al. [Hemmer et al., 2020]. Wang et al. apply a minimal margin criterion to the class probabilities of the softmax output layer to determine the most uncertain and, therefore, the most informative samples. Gal et al. apply a technique called Monte Carlo dropout. It approaches uncertainty from a Bayesian perspective. The framework conducts multiple forward passes of each instance through the model. Because dropout is enabled, each forward pass leads to different prediction results. Hemmer et al. write that this leads to more accurate uncertainty approximates compared to the approach by Wang et al. The main advantage is that it approximates a uncertainty distribution over the model parameters compared to a single softmax point estimate. Beluch et al. try to approximate such a uncertainty distribution not by multiple forward passes but by an ensemble of fully parameterized ConvNets. Their evaluation results indicate that such an ensemble of ConvNets can better infer prediction uncertainty. Yoo and Kweon follow a different idea. They train a separate loss prediction model using the output of the ConvNet to predict the losses of unlabeled samples. The image selection process for annotation then simply consist of querying those samples with high expected loss. Finally, Hemmer et al. propose to apply a Dirichlet distribution on the class probabilities generated by the

softmax layer. According to the authors, the main advantage of this approach is that it only requires a few forward passes of each instance through the network in order to achieve good uncertainty approximations.

5.2.2 Active Learning Methods for other Classifiers

In this thesis, classification is not directly performed by the softmax classification layer of the ConvNet but dedicated classifiers, which take the extracted feature vectors of the images as input (see Chapter 3). However, the goal of active learning in such a setting stays the same: Selecting those instances for annotation which are most informative for the model. Among others, Schohn and Cohn [Schohn and Cohn, 2000] have studied active learning methods for the support vector machine (SVM) classifier. They consider the *probabilistic correct approach* as one possible active learning criteria for sample selection. The probabilistic approach relies on the algorithm of Platt [Platt, 1999] which assigns probabilities to instances in the space classified by the SVM. This method is required because the SVM is a discriminant classifier and does not provided any probabilistic estimates of a classification confidence. Schohn and Cohn describe the algorithm of Pratt in an intuitive way: "project all examples onto an axis perpendicular to the dividing hyperplane, and perform logistic regression on them to extract class probabilities. By integrating the probability of error over the volume of the space, weighted by some assumed distributions of test examples, we can estimate the expected error of the classifier" [Schohn and Cohn, 2000]. Once, classification probabilities are available for each instance, the authors compute the expected effect of adding an arbitrary unlabeled example x. The greedy algorithm to do so works as follows:

1. Compute the class probability $P(y = 1 \mid x)$ and $P(y = 0 \mid x)$ (where y is the class under consideration)
2. Add $(x, 1)$ to the training set, retrain and compute the expected error $E_{(x,1)}$
3. Remove $(x, 1)$ and add $(x, 0)$ to the training set, retrain and compute the expected error $E_{(x,0)}$
4. Estimate the expected error after annotating and adding x, by:
   $E_x = P(y = 1 \mid x) \cdot E_{(x,1)} + P(y = 0 \mid x) \cdot E_{(x,0)}$
5. From all candidate instances the model selects those which minimize E_x.

The proposed greedy procedure is the best one can do but it is computationally very intensive and therefore, impractical for large classification tasks. This is also recognised by the authors. However, the method gives rise to a simplified active learning heuristic which uses the classification probability as an indicator of *uncertainty*. This uncertainty criteria is then exploited in a pool-based approach: Consider a classification system which is trained on a small training set, consisting of images containing J categories. Classifying the pool of unannotated images M results in J sets of images with $(x_i \mid y_j = 1) \in M$,

where j is a single object category and x_i is an image. To construct the selection of images for annotation, the system simply queries the positively classified images for each category. In order to get the most informative images, the system sorts the images based on the classification probability $P(y_j = 1 \mid x_i)$ in a ascending order. The classification probability $P(y_j = 1 \mid x_i)$ can be calculated for all classifiers considered in this thesis. It is available in the Scikit-learn [Pedregosa et al., 2011] implementations.

5.2.3 Human Machine Interaction

For many visual tasks, the human visual perception is superior to the visual perception of trained machines. This superiority is particularly pronounced when only a limited number of image are available for 'training' and the objects are shown in their natural context where they might be in the background or partially obscured. The human brain is structured in such a way that in many cases a single picture of an object is sufficient to distinguish it from other objects. Humans are also very fast in detecting specific objects in images. With an experiment where people where asked to detect a specific letter in a simple line image, Spence [Spence, 2002] showed that humans are capable to correctly process up to 10 images per second. This finding is a strong indication that people can rapidly scan large amount of images for a specific object. The active learning loop should take this finding into account and utilize human annotation capacities accordingly.

5.2.4 Active Learning Loop

Based on the findings in Sections 5.2.2 and 5.2.3, the active learning loop should feature the following characteristics: (i) Annotation should mainly be done on images which are most informative to the classification system and (ii) it is most likely more efficient to utilize the human annotation capacity in a way where people annotate image of a single category at the time.
In order to select the most informative images based on the *uncertainty* criterion described in Section 5.2.2, the classification system has to calculate the classification probability of each object category for each image in the pool of unannotated images. This requires some annotated images for initial classifier training. Furthermore, the system should be capable to learn a variety of different object categories. The more categories the system knows, the better. This leads to a third requirement: (iii) The system must provide functionality which allows the user to annotated images in the most effective way possible.
Once the classification probabilities have been calculated for each object category, the positively classified images are presented to the user for annotation. In this context, one could also speak of confirming, since the user either confirms or rejects the presence of an specific object category in the image. The images which are confirmed to contain a specific object can then be used in the pool of annotated images in the next training cycle. With an increased number of training instances, the classification performance is expected to increase (see evaluation results in Section 4.4.4). Again, the classification results are presented to the user who confirms or rejects the presence of a specific object

category and the confirmed images are fed back to the pool of annotated images.
The next chapter introduces the proposed pipeline design in more detail and explains the implementation details.

6

Pipeline Design

Based on the findings from Chapters 4 and 5, a large scale active learning pipeline for concept detection in video is designed, implemented and tested. This chapter focuses on the design and implementation details of the pipeline. The evaluation results are reported in Chapter 7. The Chapter starts with made assumptions and requirements for the proposed pipeline in Section 6.1. A design overview is shown in Section 6.2 and Section 6.3 discusses implementation details. The chapter ends with a brief discussion of existing opportunities for improvements.

6.1 Assumptions and Requirements

The following section introduces the scope and the major challenges for a large scale active learning pipeline for concept detection in video. Section 6.1.1 briefly explains why the pipeline uses single images as input and Section 6.1.2 list the functional requirements for the classification pipeline.

6.1.1 Object Classification in Video

Concept detection in videos requires some sort of segmentation. It is computationally inefficient to process every single frame of a video. *vitrivr* is an award winning, fully working system for retrieving multimedia data based on its content. *Cineast* by Rossetto et al. [Rossetto et al., 2014], which is the retrieval engine of vitrivr, performs shot segmentation of videos using a fuzzy color diagram. Based on the Euclidean distance of two consecutive normalized histograms, Cineast is capable to distinguish scene changes in a video. Furthermore, it identifies an average and median image for each shot. The proposed pipeline is based on the idea of analyzing a single representative frame. It assumes that such a frame is available for each scene. Consequently, the proposed pipeline design performs image classification on single images instead of entire videos.

6.1.2 Requirements

Chapter 4 and 5 provided important insights into the functional requirements for the pipeline. The following list gives an overview of some essential functional requirements:

- The system must be capable to handle large amounts of data. Extracting feature vectors, training classifiers and the classification of unclassified images is computationally very demanding. It would required large amount of memory if everything is done in memory. Therefore, an efficient storage for annotated ground truth data, embedded feature vectors and classification results is required.
- The user must be able to upload new image data to the system so that they can search these images for specific categories.
- The user must be able to train the system with new object categories so that the system is capable to identify these new object categories in the pool of uploaded images.
- In Section 4.4.1, we saw that there is not a specific classifier which works best for every feature extraction model and object category. The pipeline must be capable to select the best classifier for the classification task at hand so that the user gets the best possible search results.
- Sections 4.4.3 and 4.4.4 provided clear evidence that classification performance is highly positively correlated with the number of training instances. The system must provide functionality which allows the user to rapidly grow the number of training instances so that the classification performance can be rapidly improved.
- The classification performance of object categories for which only a very limited number of training instance is available can be improved by adding extracted box object images to the training set (see 4.4.2). The pipeline must provide functionality to manually annotate box objects in images and use these box annotations for training so that the user gets the best possible classification performance even for very small training sets.

6.2 Overview

The mode of operation for the proposed active learning pipeline for concept detection is as follows: 1) The user uploads unannotated images to the system. 2) In order to teach the system new object categories the user can either upload images of a specific object category or box annotate randomly selected images from the pool of unannotated images. 3) The system is then trained on the available training instances. 4) Using the best performing classifier for each object category, the system classifies the pool of unannotated images. 5) The positively classified images are then presented to the user who can confirm the classification given the object is present in the image. 6) The confirmed images are fed back to the pool of annotated images and used in the next training and validation cycle.

Figures 6.1 provides a schematic overview of the image import and annotation process of the proposed pipeline. The import process of images, which form the pool of unannotated images, is straight forward: Based on the import path provided by the user, the

images are uploaded and feature vectors are extracted. The user can choose between the different feature extraction models introduced in Chapter 4. Both, the information of the uploaded images as well as the feature vectors are stored in a database (see Section 6.3.2 to learn more about the database). For object annotation, the system provides two options: First, the user can randomly select images from the pool of unannotated images and box annotate all objects present in the image using the *VGG Image Annotator (VIA)* application by Dutta et al. [Dutta et al., 2016] and [Dutta and Zisserman, 2019]. Second, the user can upload images containing a specific object directly to the system. The second option allows to rapidly grow the pool of available annotated images of a specific object category, while the first option is more exploratory. Once the images are annotated, the images are split into a training data set and validation data set, based on a splitting ratio defined by the user.

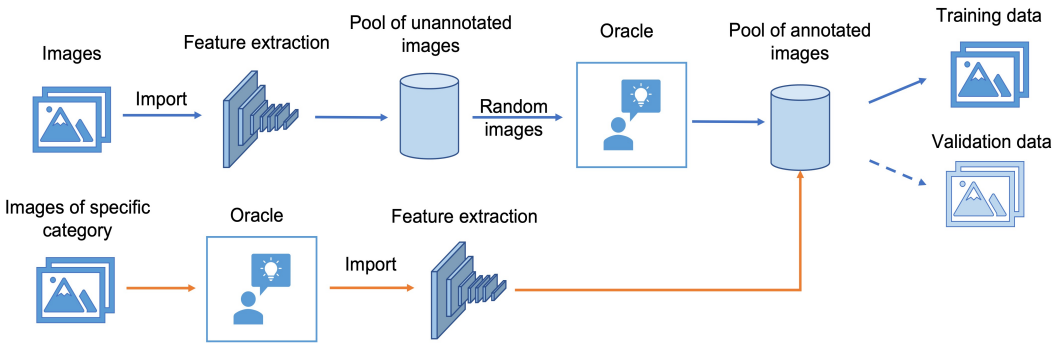


Figure 6.1: Schema of the import and annotation process. For image annotation, there are two different ways to grow the training and validation data pools. First, the user can randomly select images from the pool of unannotated images (indicated by blue arrows). Second, the user can import images which contain a specific object category (indicated by orange arrows).

Figure 6.2 schematically shows the training, classification, object search and object confirmation process. In the training process, for each object category all classifiers introduced in Chapter 3 are trained with the extracted feature vectors. The trained classifier are stored on a local drive. The validation set is then used to validate each classifier and store the classification performance results on the database. Based on these result, the best classifier is chosen by the system to classify the pool of unannotated images in the classification process. If the user decided to allocate none of the annotated images to the validation pool, the LinearSVC is selected as a default classifier. After classification, the user can search the pool of classified images for specific object categories. The search results are presented in the web frontend. The user has then the possibility to confirm images which contain the object of interest. Confirmed images are then back assigned to the pool of annotated images and can be used in the next training and validation cycle. This functionality helps to rapidly grow the pool of annotated images.

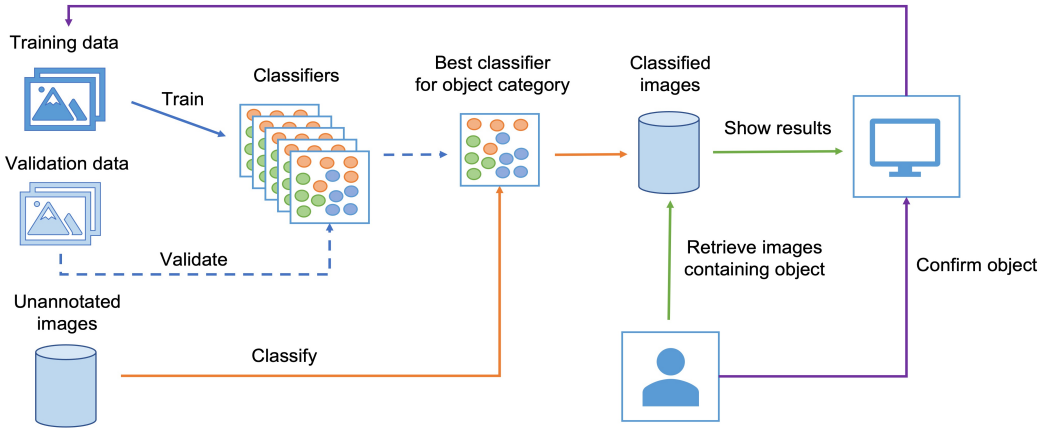


Figure 6.2: Schema of the training (blue arrows), classification (orange arrows), object search (green arrows) and object confirmation process (purple arrows)

6.3 Implementation Details

This section provides further implementation details of the classification pipeline. Section 6.3.1 starts with a overview of the technologies used. Section 6.3.2 explains the database schema

6.3.1 Technologies

The backend of the classification pipeline is entire build in Python, version 3.8, with extensive use of the Tensorflow package [Abadi et al., 2015], version 2.4.1, and classifiers from Scikit-learn [Pedregosa et al., 2011], version 0.24.1. For storage, the system relies on the Cottontail DB by Gasser et al. [Gasser et al., 2020], version 0.12.12 and the Cottontail-Python-Client, version 0.0.4. The Web frontend uses HTML, CSS, Javascript and jQuery and the web server is powered by Flask, version 2.01. A REST API with Swagger documentation ensures communication between frontend and backend.

6.3.2 Cottontail DB

Cottontail DB is a column store developed by Gasser et at. [Gasser et al., 2020] specifically for multimedia retrieval. The proposed pipeline uses Cottontail DB to store all relevant data for the active learning task. Figure 6.3 depicts the database schema. Please note that certain connections between units have been omitted for the sake of clarity. Specifically, all connection between the entity **categories** and the entities depicted on the right have been omitted. Furthermore, there exist a dedicated entity **feature_vectors_** for each feature extraction model.
The central entity is the **multimedia_objects** entity. For each imported image an entry is written in the entity. The column *object_id* contains a unique identifier for each image. It is generated by hashing the image with the SHA256 algorithm. Please note

that Cottontail DB currently does not support primary keys. However, *object_id* would be the primary key of the entity. Also during import, the feature vectors get extracted. The feature vectors are stored in separate entities for each feature extraction model.

The entity **annotated_objects** contains the necessary information of annotated images. Each entry represents an annotated image. Images annotated using the VIA box annotation tool may contain several box object of the same category. Again, annotation information on image level are stored in the **annotated_objects** entity. All information regarding annotated box objects within an image are stored in the entities **annotated_boxes** and **extracted_box_objects**.
As has been discussed Section 6.2, the system is capable to select the *best* classifier for each feature embedding model and object category. In order to determine the *best* classifier, the system requires images assigned to the validation pool. If such images are available, four different performance metrics are calculated for each classifier: *Accuracy, Precision, Recall* and *F1* and the results are stored in the **classifier_performance** entity.
The entity **classifications** contains all classification results. Only positively classified images are stored in the the entity. An object_id may occur multiple times in the entity due to the fact that en entry is written for each combination of feature extraction model and classifier.
The remaining entities are **categories**, **classifiers** and **embedders**. The entity **categories** contains all object categories known by the system. If an new category is added, an additional entry is written to the entity. The latter two entities contain the names of the classifiers and feature extraction models known by the system.

6.3.3 Backend and API Endpoints

This section provides more details about the backend processes of the system. The section is structures based on the most important API endpoints and shows pseudo code for the most important routines.

Import

As shown in Figure 6.1 there are two methods to import images to the system: (i) importing unannotated images and (ii) importing annotated images. The following pseudo codes shows the routine for each method.

Annotation

The endpoint *annotation* is used for annotation with the VIA application as shown in 6.1 (see process with blue arrows). It randomly selects unannotated images from the **multimedia_objects** entity and hands them over to the VIA application. A representation in pseudo code is omitted.

Algorithm 1 import.import_unannotated_media(*media_path*, *embedder*)

for each image **do**
    get sha256 image hash
    rename image with image hash
    collect image information required for **multimedia_objects**
end for
batch insert all image information into entity **multimedia_objects**
for each *embedder* **do**
    extract feature vectors of all unannotated images
    insert feature vectors into entity **feature_vectors_*modelname***
end for
update entity **multimedia_objects**, set $embedded = TRUE$
return *object_ids*

Algorithm 2 import.import_annotated_media(*images*, *embedder*, *category_label*)

Do Algorithm 1
if *category_label* not in entity **categories then**
    get new *category_id*
    insert *category_label, category_label* into entity **categories**
end if
insert *object_ids, category_id* into entity **annotated_objects**
entity **multimedia_objects**, set $annotated = TRUE$

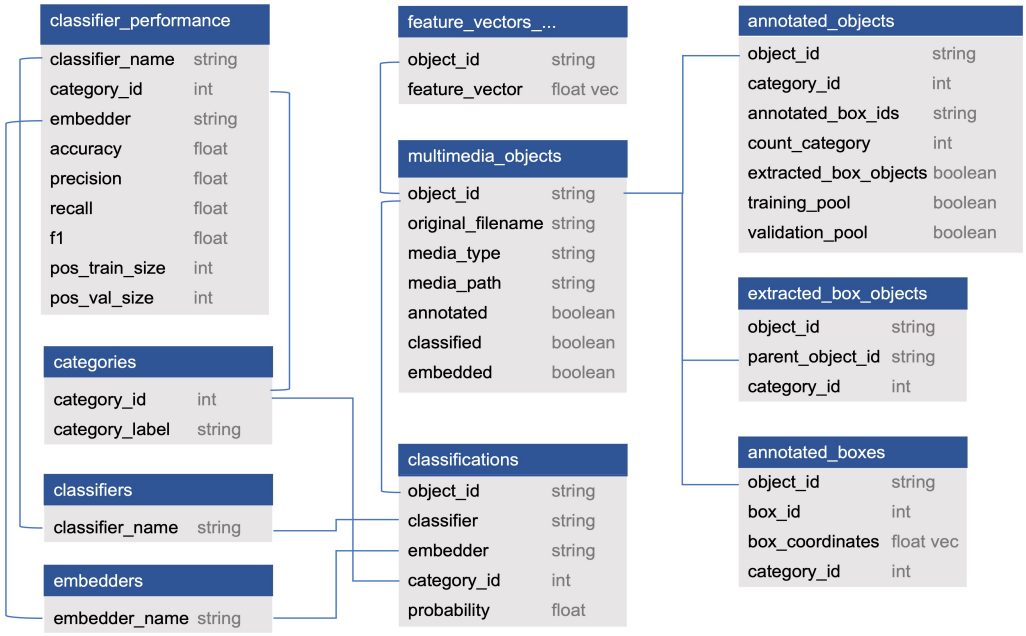


Figure 6.3: Cottontail DB schema

Process

Process provides three functionalities: (i) *process.annotations_to_db* processes the annotation file, which is provided by the VIA application. (ii) *process.split_training_validation* splits the pool of annotated images into a training set and a validation set based on the splitting ratio defined by the user and (iii) *process.read_annotated_images* reads the pool of annotated images and returns a statistic about the number of available training and validation instances for each object category back to the frontend. In the following, pseudo code for *process.annotations_to_db* is shown.

Train

The training process can be started with the endpoint *train*. The following pseudo code shows a simplified version of the procedure.

In line 7 and line 8 of Algorithm 4 the training and validation sets are created. Their composition is always perfectly balanced, meaning they consist of an equal number of positive and negative training instances: First, all positive instances are selected. Then, the system randomly chooses an equal number of negative instances from the pool of annotated training/validation images. If the user wants to include extracted box images to the training set, the system adds all extracted box images of the positive training instances to the training set, which contain the object of interest. Again, it then randomly selects an equal number of negative box images. Please note, that no extracted box images are added to the validation set.

Algorithm 3 process.annotations_to_db(*annotation_file*, *embedder*)

extract *object_ids, annotations, category_ids, category_labels,* from *annotation_file*
insert *category_ids, category_labels* into entity **categories**
for each *object_id* **do**
    for each *annotation* **do**
        collect *box_coordinates, category_id*
    end for
    insert box objects info into entities **annotated_objects, annotated_boxes**
end for
for each *object_id* **do**
    get all *box_coordinates*
    for each *box_coordinates* **do**
        crop image to *box_coordinates*
        resize image
        get sha256 image hash
        insert extracted box image info into entity **extracted_box_objects** table
    end for
end for
for each *embedder* **do**
    extract feature vectors of all box images
    insert feature vectors into entity **feature_vectors_*modelname***
end for
update entity **multimedia_objects**, set $embedded = TRUE$
update entity **multimedia_objects**, set $annotated = TRUE$
update entity **annotated_objects**, set $extracted = TRUE$

Algorithm 4 train.train_classifiers(*embedder*, *category_ids*, *category_id*, *all_categories*, *include_box_images*)

```
1: if all_categories then
2:     get all category_ids from entity categories
3: else
4:     get category_id
5: end if
6: for each category_id do
7:     get training set (including box images if include_box_images = TRUE)
8:     get validation set
9:     get feature vectors of training set and validation set
10: end for
11: get all classifier
12: for each embedder do
13:     get feature vectors of training set and validation set for embedder
14:     for each classifier do
15:         train classifier
16:         store classifier to local drive
17:         if validation set available then
18:             predict validation set with classifier
19:             insert performance metrics into entity classifier_performance
20:         end if
21:     end for
22: end for
```

Classification

The endpoint *classification* provides two functionalities:
(i) with *classification.classify_multimedia_objects* the classification process is started. (ii) *classification.confirm_classification* allows the user to confirm classification results which are then added to the pool of annotated images (see also Figure 6.2. The following pseudo code shows the procedure for *classification.classify_multimedia_objects*.

Results

The endpoint *results.get_results* retrieves the classification results for specified *embedder*, *category_id*, *classifier* and *performance_measure* and make the images accessible to the frontend. If *classifier* is defined as 'best', the best *classifier* is selected based on the *performance_measure* and the results are selected accordingly. A representation in pseudo code is omitted.

6.3.4 Frontend

The frontend provides the user with access to the system. The entry page is divided into a control area and a display area. The control area consists of the following sections:

- **General Settings**: Under general settings, the user can define the feature extraction model used by the system. Besides the four models introduced in Chapter 2, the user can also choose to run the system with all feature extraction models. If 'all Models' is selected, the import, training and classification is done for each feature extraction model separately. Also under general settings, the user can define the classifier with which the training and classification is performed. The recommended setting is 'best': The system will train all classifiers and select the best performing classifier (based on the performance metric defined) for classification. Alternative settings are 'All Classifiers' in which case the classification is done for each classifier separately or a specific classifier. Please be aware that the general settings apply for all control areas like **Import, Annotation, Train** and **Classification** but also for the view area **Classification Results**

- **Import**: This control area allows the user to import unannotated images from a local folder.

- **Annotation**: There are two ways to provide image annotations to the system (see also Figure 6.1). Both processes can be started via this control area. First, the user can provide box annotation for randomly selected images using the VIA application by Dutta et al. [Dutta et al., 2016] and [Dutta and Zisserman, 2019]. To do so, the user can define a number of images and press the *Start Annotation* button. The VIA application then automatically starts. Please refer to the Appendix, Section A.4, to get a short tutorial on how VIA should be used. It is very important to note that the integration of the VIA application was done in a very shallow way.

Algorithm 5 classification.classify_multimedia_objects (*embedder*, *category_id*, *all_categories*, *classifier*, *performance_measure*, *classify_all_media*)

```
1: if classify_all_media then
2:     get object_ids of all unannotated images form entity multimedia_objects
3: else
4:     get object_ids of all unclassified images form entity multimedia_objects
5: end if
6: if all_categories then
7:     get all category_ids from entity categories
8: else
9:     get category_id
10: end if
11: for each embedder do
12:     get feature vectors of all object_ids
13:     for each category_id do
14:         if classifier = 'best' then
15:             get name of best classifier from entity classifier_performance
16:         else if classifier = 'all' then
17:             get all classifier names from entity classifiers
18:         else
19:             get classifier name
20:         end if
21:         for each classifier do
22:             load classifier from local drive
23:             predict class of feature vectors
24:             if classify_all_media then
25:                 delete classification results in entity classifications
26:                 update entity classifications with classification results
27:             else
28:                 update entity classifications with classification results
29:             end if
30:         end for
31:     end for
32:     update entity multimedia_objects, set classified = TRUE
33: end for
```

Please refer to Section 6.4 to learn more about it. The second option to import annotations is the direct import of images of a specific category. After image selection, a preview is shown and the user is ask to provide a category label.

- **Train**: In this control area, the user can start the training process. Training can either be done for a specific category, in which case the user has to provide the category id and set 'All Categories' to false, or for all categories ('All Categories' to true). The user can also determine if training should include extracted box images.
- **Classification**: Here, the user can start the classification process. Again, classification can either be done for a specific category or for all categories. Furthermore, the user can define, whether classification should be done for all unannotated media objects in the database or only for those which are not classified yet.

The view area only consists of two sections: First, there is a section **Available annotated images** which provides a table with the number count of available training and validation instances for each category. Second, the classification results for a specified category can be accessed in the Section **Classification Results**. Please be aware that classification results can only be shown for a specific feature extraction model. No results are shown if 'all Models' is selected in the *General Settings*.

6.4 Known Issue And Future Work

The current implementation still has some issues. These known issues are listed below and may guide future efforts to improve the pipeline:

- **Database**
  - Currently, primary keys are not supported in Cottontail DB. Therefore, it is possible to import several copies of the same image. This may cause problems during the creation of the training and validation sets. The problem should be solved as soon as Cottontail DB supports primary keys. The initialization script of the database entities has already been prepared accordingly.
  - The two entities **extracted_box_objects** and **annotated_boxes** can be merged without any loss of information. The current schema design resulted from a different approach which was later abandoned.
- **Backend**
  - The primary goal of this thesis was to get a working prototype of the pipeline which seamlessly inter-operates with vitrivr's query engine Cineast. Unfortunately, due to time constraints, I could not establish inter-operability with Cineast. Furthermore, the processes were not specifically optimized for efficiency regarding process time and memory consumption.

  - Currently only '.jpg' and '.JEPG' images are supported
  - When training and classification is done for all feature extraction models, the backend loads the feature vectors for all models into memory. There might be more efficient ways to do it.
  - Currently, the random split of annotated images into training and validation sets is done for the entire pool of annotated images. It would be more efficient if the split would only be performed for categories, for which new images were added.
  - For the evaluation of the pipeline, unannotated images were not uploaded to the web server during import, but kept in the local directory. This way, the classification performance results could be calculated in a more controlled way. When displaying the classification results to the user, the positively classified images are copied to a folder managed by the web server and provided from there to the classification results page. In doing so, the sorting based on the classification probability gets lost and the images are no longer presented in a way that the most uncertain classified images come first. This issue can be easily solved by uploading the images to the web server during import and serve them in the correct order to the results page.

- **VIA integration**
  - Currently, the VIA application has difficulties with category ids which are not continuous. Therefore, it is important to assign continuously increasing integer ids when adding new categories in VIA (see also Section A.4 in the Appendix).
  - The integration of the VIA application in the pipeline is very shallow. For example, randomly selected images for annotation are not directly provided by the web server. The web server provides the images to the entry page and from there the images are stored to the local storage. The VIA application reads the image from the local storage. Due to the limited size of the storage (around 5MB) only a limited number of images can be transferred.

7

Pipeline Evaluation

This chapter presents the evaluation results of the proposes classification pipeline. The chapter starts with three research questions in Section 7.1. The analysis of the results were guided by these questions. Section 7.2 introduces the custom data set used for the evaluation. Section 7.3 briefly discusses the Setup and Section 7.4 discusses the results. The chapter ends with a brief summary and answers the research questions.

7.1 Research Questions

This chapter aims to answer the following research questions:

- **RQ1:** Which combination of feature vector embedding model and classifier performs best in the proposed pipeline?
- **RQ2:** Can we achieve a reasonable classification performance for iconic and non-iconic view objects with only a limited number of annotated training instances?
- **RQ3:** Can we reduce the annotation effort by adding extracted box images to the training set?

7.2 Data Set

For the evaluation of the proposed pipeline, a custom data set was created. To understand the classification capabilities of the system, the data set has to cover a variety of different classification scenarios. The following scenarios were considered: (i) The object is depicted in an iconic view, (ii) the object is depicted in its natural context. This includes animals in their natural environment or objects which are in the background of an image or partially obscured. (iii) Images which represent different concepts. This may include real-world images, x-ray images or comics/cartoons.

In addition to the classification scenarios stated above, the test data set should consist of images from sources other than those used for the evaluation in Chapter 4.

7.2.1 Training and Test Sets

As a primary source of annotated images I used the *Open Images V5* validation data set [Kuznetsova et al., 2020]. It consists of 41,620 images containing 600 object categories. However, for many object categories there are not enough images available to run the intended evaluation. Therefore, a subsample of 8 object categories was selected. Table 7.1 provides an overview of all object categories including a subjective assessment of the classification scenario and classification difficulty.

Object category	**Data source**	**Scenario**	**Difficulty**
Ball	Open Images	Natural context	Hard
Bird	Open Images	Natural context	Medium
Bread	Open Images	Iconic view	Easy
Camera	Open Images	Mainly iconic view	Medium
Fish	Open Images	Natural context	Hard
Goggles	Open Images	Natural context	Hard
Guitar	Open Images	Mainly iconic view	Medium
Chainsaw	Imagenette	Mainly iconic view	Medium
French Horn	Imagenette	Mainly iconic view	Easy
Motorcycle	Open Images	Natural context	Medium
Radiology	Kaggle	Concept	Easy
Comic	Kaggle	Concept	Easy

Table 7.1: Object categories in the evaluation data set

In addition to the 8 categories from Open Images, 4 more categories where added to the data set. *Chainsaw* and *French horn* are both from the *Imagenette* data set [Hammel et al., 2019]. Imagenette is a subsample of relatively easy classifiable object categories of the *ImageNet* data set [Russakovsky et al., 2015]. Please note, that the feature extraction models were trained on the *ImageNet*. The object category *Radiology* is one of two categories representing a different image concept. It consists of chest x-ray images collected from the Kaggle [Mooney, 2017]. Last but not least, a collection of screen shot images from the animated series 'The Simpsons' were added. The data set is also available on Kaggle [Malov, 2021]. It represents the concept *Comics*. An example image of each object category is depcited in Figure 7.1.

Test Set

From each object category described in Section 7.2.1, 30 images were added to the test set. 640 more images were randomly selected from the *Open Images* data set taking care to use only images that did not contain any of the above objects. Consequently, for each object category, the data set contains 30 condition positive and 970 condition negative images.

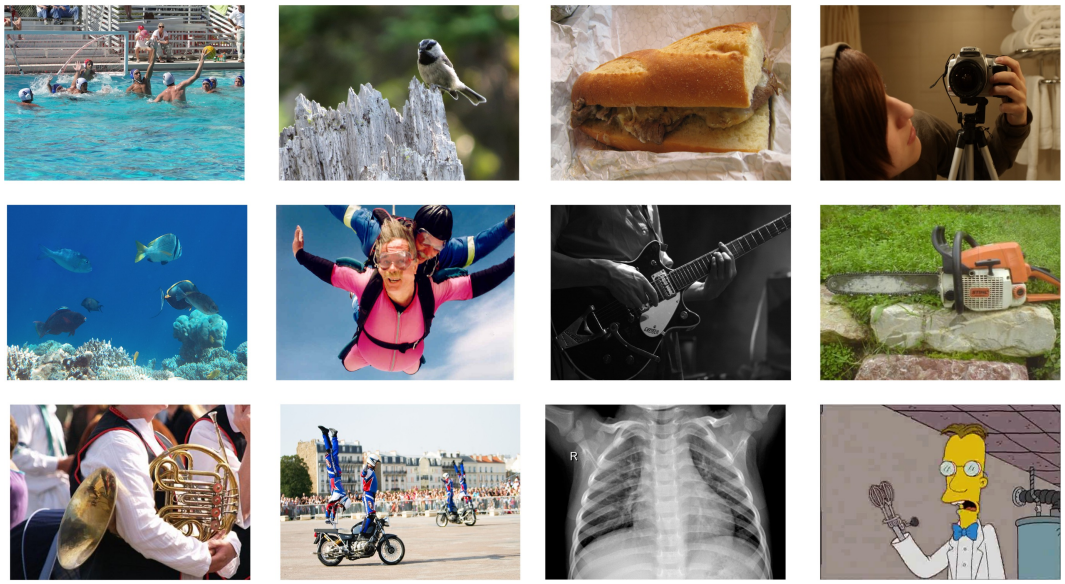


Figure 7.1: Classification test examples for each category. Starting from the top left: *Ball*, *Bird*, *Bread*, *Camera*, *Fish*, *Goggles*, *Guitar*, *Chainsaw*, *French Horn*, *Motorcycle*, *Radiology*, *Comic*

Training Set

The training set consist of 70 images of each object category. Similar to the procedure described in Section 4.3.3 the number of positive training instances is incrementally increased in a pseudo logarithmic way, starting from 1. However, the maximum number of positive training instances is limited to 50. This is due to the finding in Section 4.4.4 which showed that after 50 positive training instances the marginal gains in classification performance reduced substantially. For the test run with 50 positive training samples 20 instance were assigned to the validation pool. For all other training steps, a split rate of 50% was applied, resulting in a equal training pool and validation pool size.

7.3 Evaluation Setup

The entire evaluation was run on a MacBook Pro with macOS 10.15.7, an Intel Core i7 processor and 8 GB of RAM. No dedicated GPU resources were assigned to the task.

7.3.1 Pipeline Configuration

The pipeline was configured to select the 'best' classifier based on the F1 performance metric. As described in Section 6.2, the performance of each classifier is determined by the classification of the validation data set. The evaluation was performed for all feature-extraction models simultaneously, using the provided functionality of the pipeline.

7.3.2 Performance Metric and Result Analysis

For the evaluation of the proposed pipeline I focused on the same performance metrics as described in Section 4.3.5. However, for the intended purpose of the pipeline, the F1 metric captures more relevant information than the Accuracy metric. The evaluation results were then analyzed and visualized using Microsoft Excel.

7.4 Evaluation Results

To evaluate the classification performance, three separate evaluation runs were conducted. It was necessary to account for the randomness in the system: 1) Annotated images are randomly assigned to the training and validation pools based on the split ration defined by the user. 2) The selection of negative training samples (also of negative training box images) is conducted in a random fashion. Hence, the following results are shown as an average over all evaluation runs if not stated differently.

The Section is structured as follows: Section 7.4.1 presents the classification performance over all object categories. Section 7.4.2 discusses the performance on individual object categories. In Section 7.4.3 I answer the question whether it is beneficial to included extracted box images for the initial training cycle and Section 7.4.4 analyses the chosen classifiers for the classification scenarios.

7.4.1 Classification Performance Over All Object Categories

Figure 7.2 plots the F1 classification performance metric for each feature extraction model over all object categories. Overall, the InceptionResNetV2 architecture outperforms the other models by some distance. The performance difference is especially apparent for small training sets. Interestingly, InceptionResNetV2 reaches an F1 value of almost 0.6 with only 10 training instances but cannot improve much with larger training sets. On the other hand, the InceptionV3 and ResNet50V2 models gain steadily from more training instances and almost ketch up to the InceptionResNetV2 with 50 positive training instances. InceptionV3 also performs well with small training sets (≤ 5). The MobileNetV2 architecture is the second best performer for training set between 5 and 20 training instances. For a lightweight architecture, this might come as a surprise. However, already the preliminary results presented in Chapter 4 indicated good classification performance of the MobileNetV2 architecture. However, the architecture might contribute to some overfitting of the classifiers, because the F1 decreased between 20 and 50 positive training instances. In general, there seems to be some saturation tendencies in all architectures, which occur earlier than in the previous tests (see for instance Figure 4.3 for comparison). The Accuracy scores can be found in the Appendix, Section A.3.1.

As has been mentioned in the introduction to this Section, the pipeline features some randomness. It is therefore important to know how this randomness effects the classification performance of the system. Figure 7.3 shows the average weighted F1 score together with the minimum and maximum score over three separate evaluation runs (please note

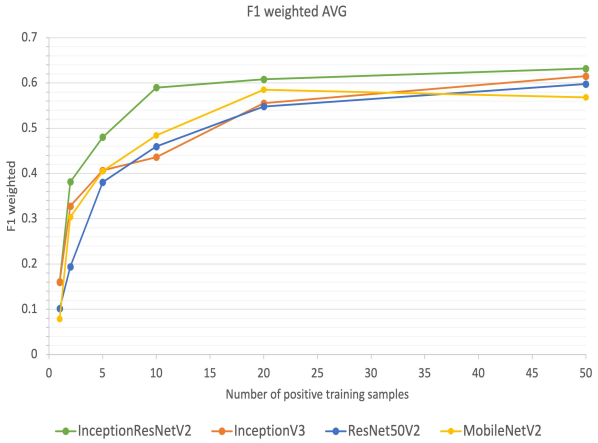


Figure 7.2: F1 classification performance for each feature extraction model over all object object categories

that the graphs feature a different x-axis scale). The InceptionV3 model has the highest variance across runs. The ResNet50V2 model seems to deliver the most consistent results. Interestingly, the MobileNetV2 model had almost the exact same weighted F1 scores in each run for training sets with 2 and 50 instances. Even when accounting for the minimum score of the InceptionResNetV2 architecture, it still outperforms the other feature extraction models in most cases.

A first preliminary conclusion can be drawn from the analysis of the results over all object categories: The InceptionResNetV2 architecture achieved the highest F1 scores and features an acceptable variance over different runs. The average and minimum and maximum Accuracy scores can be found in the Appendix, Section A.23.

7.4.2 Classification Performance for Single Categories

Looking at the F1 scores for each category individually underlines the finding from Section 7.4.1. Table 7.2 reports the average weighted F1 scores for each feature extraction model over all training set sizes, from 1 positive training instance up to 50 positive training instances. For 6 out of 12 categories, the InceptionResNetV2 model achieved the highest score. Especially, for object classification problems which are considered to be hard, like *Ball* and *Fish*, the model performed well compared to the other models. Only in the third hard problem, *Goggles*, the MobileNetV2 architecture performed better. Interestingly, it struggled in the category *Bird*, with an average weighted F1 score of only 0.192.

Table 7.2 also shows how different the classification performance is for different object types: *Chainsaw*, *French Horn*, *Radiology* and *Comic* could be classified with high scores. On the other end are *Goggles*, *Ball*, *Fish* and *Bird*. This was expected due to the different classification difficulty of the object types. However, *Chainsaw* and *French Horn* performed much better than *Bread* which is also considered to be an easy classifi-

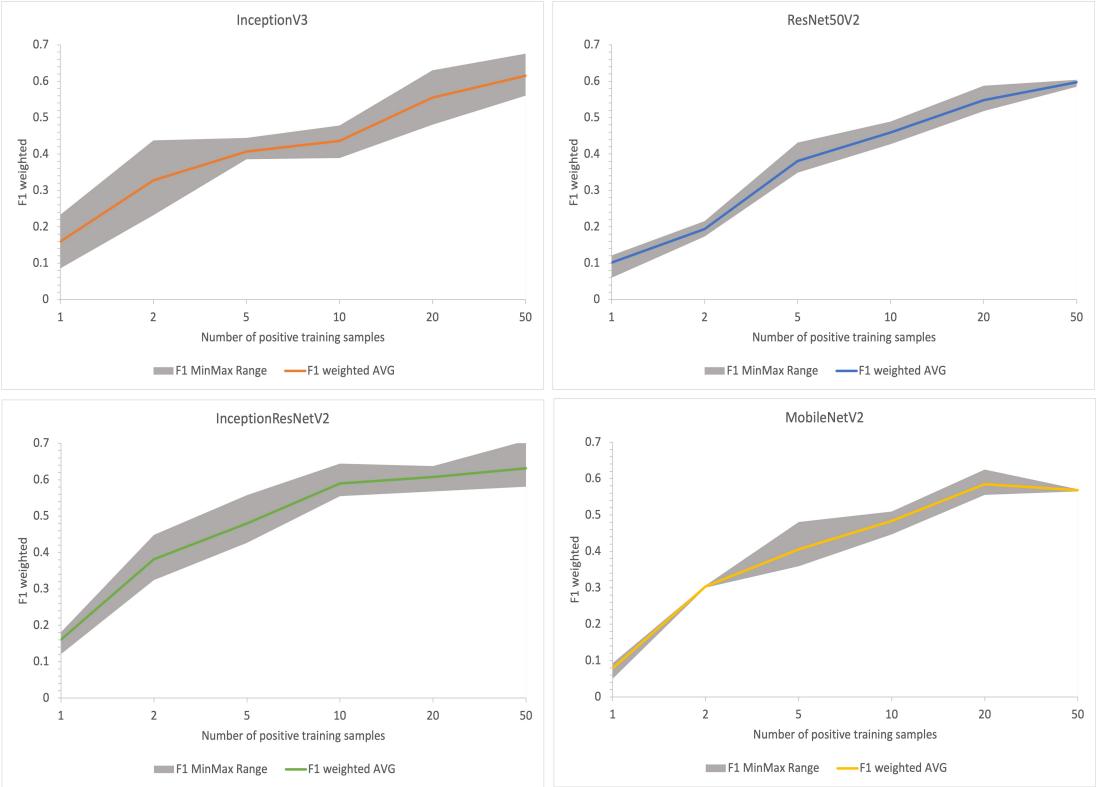


Figure 7.3: Average and MinMax range of the F1 classification performance over three separate evaluation runs

Category	InceptionV3	InceptionResNetV2	ResNet50V2	MobileNetV2
Ball	0.275	0.322	0.177	0.257
Bird	0.188	0.192	0.382	0.317
Bread	0.387	0.436	0.317	0.390
Camera	0.420	0.392	0.410	0.374
Fish	0.316	0.321	0.193	0.254
Goggles	0.238	0.268	0.244	0.288
Guitar	0.541	0.555	0.397	0.345
Chainsaw	0.473	0.735	0.324	0.341
French Horn	0.852	0.781	0.457	0.474
Motorcycle	0.379	0.371	0.405	0.354
Radiology	0.608	0.763	0.693	0.815
Comic	0.621	0.610	0.563	0.749

Table 7.2: Highest average weighted F1 scores over all training sets

cation task. One might conclude that the former classes performed way better because the feature extraction model were trained on these categories. However, this conclusion

is not fully supported by Figure 7.4 which illustrates the differences graphically. Substantially better results were only achieved for the InceptionResNetV2 (*Chainsaw* and *French Horn*) and InceptionV3 architectures (*French Horn*). All other models resulted in similar classification performance as for the category *Bread*. The figure also shows that InceptionResNetV2 performs comparably well in all but one category.

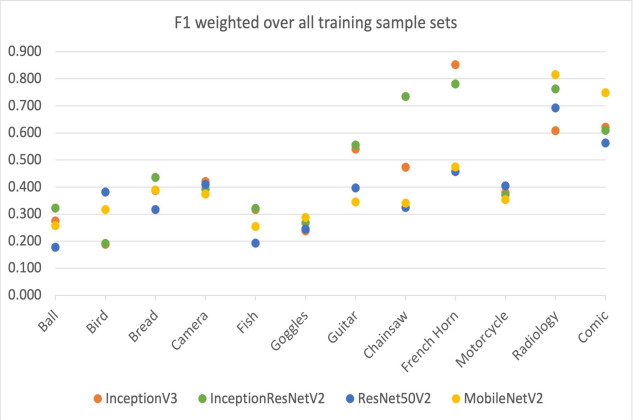


Figure 7.4: Highest average weighted F1 scores over all training sets

For the proposed pipeline, object classes must not only be classified efficiently over all training set sizes. Since the pipeline's sequence of operations assumes that the system is initially trained with a limited number of annotated images and the positively classified images are then presented to the user for confirmation, the best possible classification performance is required even for small training sets. Figure 7.5 plots the average F1 scores for all object categories. Analysing the results provides important insights: First, for classification tasks which are considered to be easy – like *French Horn*, *Radiology* and *Radiology* – provide good results even for training sets with only 5 instances. Also *Guitar* and *Chainsaw* were classified well with only 5 instances. This is especially true for the InceptionResNetV2 architecture. Other categories like *Bread*, *Camera*, *Motorcycles* and *Bird* require ideally 10 instances to achieve acceptable results (the InceptionResNetV2 model requires 50 instances to achieve a F1 score of 0.4 for the the category *Bird*). The category *Fish* also achieves good results with 10 instances but only with the InceptionResNetV2 model. The most difficult categories are *Ball* and *Goggles*. Although for both categories an F1 requires of 0.5 is achieved with 20 and 10 positive training instances, respectively, with the InceptionResNetV2 model, there seems to be a substantial saturation tendency. Also, other categories like *Bread*, *Guitar*, and *Motorcycle* exhibit a substantial saturation tendency, if not overfitting. This is problematic because we can no longer follow the simple heuristic that the more training instances, the better. Future work on this topic should be directed toward investigating the reasons for this trend.

Analysing the classification performance for each object category supports the finding from Section 7.4.1 that the InceptionResNetV2 network is over all the most capable feature extraction model even for small training sets. The saturation tendency for many

object categories is problematic and should be investigated in more detail in a future study.

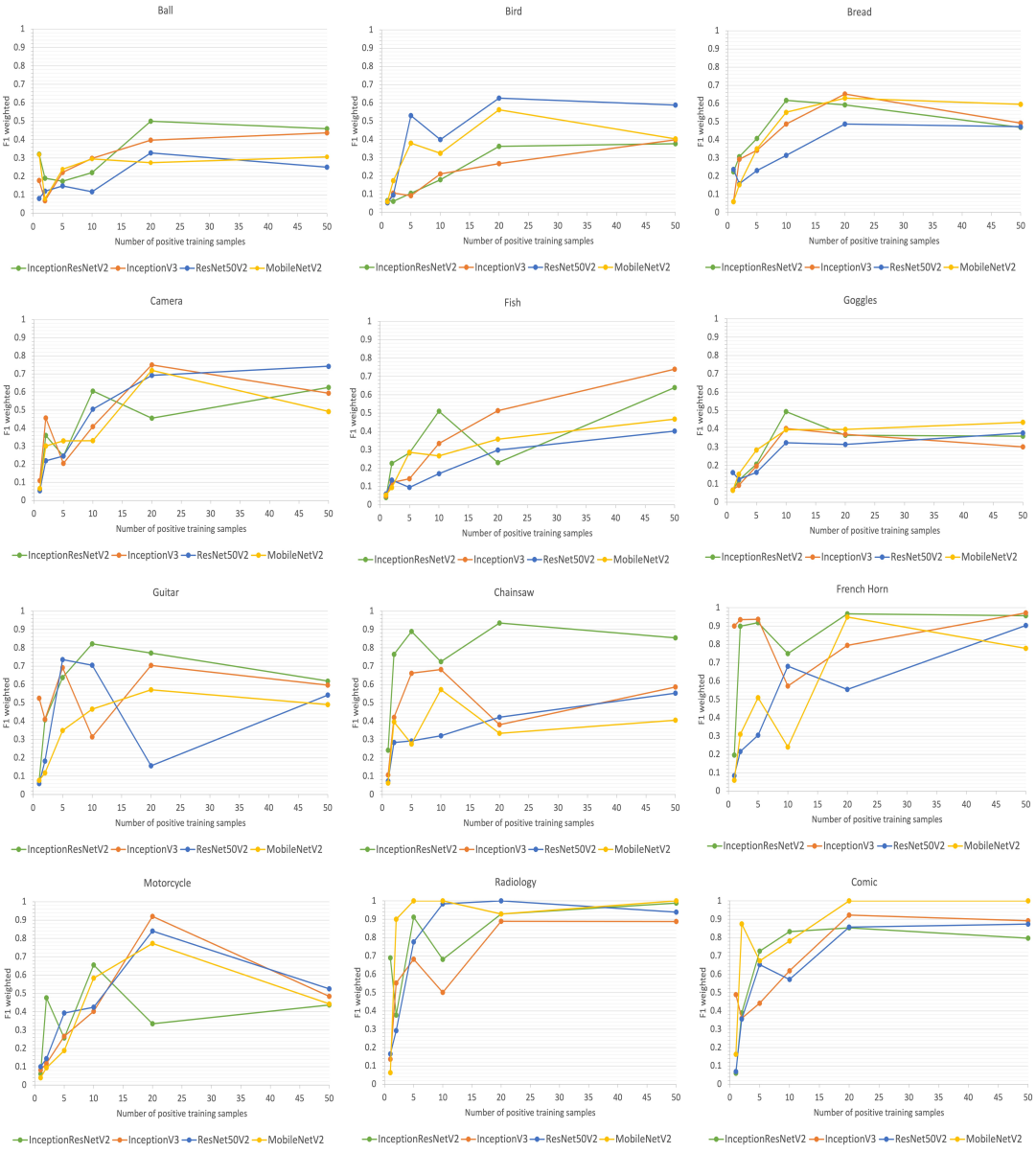


Figure 7.5: F1 scores for each object category

7.4.3 Boosting Performance by adding Box Images?

Research question 3 asks if the burden of annotation can be reduced by adding extracted box images to the training sets. The preliminary evaluation results from Section 4.4.2

indicated that it is advantageous to add extracted box images if only very few training instances are available (≤ 5). This Section reports the results from the pipeline evaluation.

To evaluate the effect of adding extracted box images, separate evaluations with and without extracted box images were conducted, each evaluation consisting of 3 runs. The evaluation focused on training sets with 1, 2 and 5 training instances. Only a single object was box annotated in each image. Care was taken to ensure that the training in the box vs. no box cases was done on the same positive training samples. Figure 7.6 plots the average weighted F1 scores over all object categories. The InceptionV3 and InceptionResNetV2 architectures register marginal gains when the classifiers are trained with additional extracted box objects. ResNet50V2 and especially MobileNetV2 performed worse.

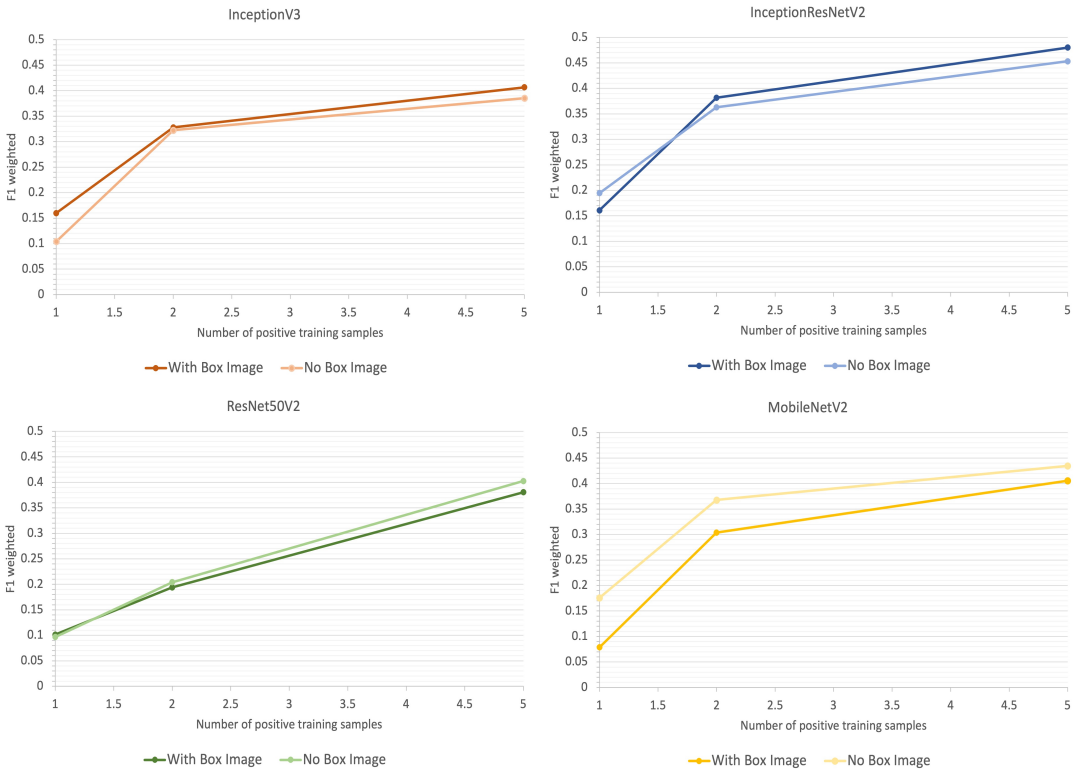


Figure 7.6: Average weighted F1 scores over all object categories with and without extracted box objects

Table 7.3 provides more detailed results for the object categories *Ball*, *Fish*, *Goggles* and *Motorcycle*. The Figure shows F1 scores averaged over 3 runs for each training set size. Interestingly, extracting a box image was especially beneficial in the scenario where only one training instance of category *Ball* was used. Other then that, there is no obvious pattern which clearly indicates that one strategy works better than the other. However, if we consider the sum of the individual net gains $\sum_{i\in[1,2,5],j}(b_{i,j} - n_{i,j})$ of

each strategy, where b is the F1 score with box and n the score without box, j is the category and i the size of positive training instances, adding box images is most effective for the InceptionResNetV2 model. Over the four categories depicted in Figure 7.6 a net gain of 0.318 can be reported. For the MobileNetV2 architecture adding box images is counterproductive.

The presented results provide some indication that adding extracted box images to the training is beneficial for the InceptionResNetV2 model. However, the marginal gains depend on object type to be classified.

	Category	**Ball**			**Fish**			**Goggles**			**Motorcycle**		
	Nr. pos training samples	**1**	**2**	**5**	**1**	**2**	**5**	**1**	**2**	**5**	**1**	**2**	**5**
InceptionV3	**F1 AVG Box**	0.228	0.068	0.222	0.048	0.121	0.142	0.066	0.092	0.196	0.083	0.118	0.266
	F1 AVG No Box	0.074	0.066	0.235	0.089	0.087	0.266	0.063	0.186	0.141	0.068	0.196	0.127
InceptionResNetV2	**F1 AVG Box**	0.390	0.190	0.174	0.039	0.225	0.283	0.065	0.120	0.207	0.062	0.476	0.257
	F1 AVG No Box	0.055	0.076	0.380	0.064	0.090	0.188	0.311	0.100	0.307	0.123	0.182	0.297
ResNet50V2	**F1 AVG Box**	0.100	0.120	0.149	0.058	0.135	0.094	0.162	0.125	0.163	0.100	0.145	0.393
	F1 AVG No Box	0.055	0.099	0.119	0.060	0.061	0.159	0.054	0.258	0.362	0.074	0.110	0.258
MobileNetV2	**F1 AVG Box**	0.349	0.078	0.239	0.053	0.094	0.286	0.064	0.153	0.284	0.041	0.094	0.189
	F1 AVG No Box	0.056	0.132	0.327	0.061	0.141	0.299	0.057	0.313	0.128	0.070	0.176	0.286

Table 7.3: Average F1 scores for each object category and training set size with and without extracted box objects

7.4.4 Classifier Selection

As has been discussed in Chapter 6, the classification system selects the 'best' classifier for each object category. This section takes a closer look at the selected classifier. Figure 7.7 shows the ratio at which a specific classifier was selected for each category. Please note, that the figure does not present average ratios but the results from run 1. Two main insights can be gained from this figure: First, there is no such thing as the 'best' classifier for a specific object category. Second, the 'best' classifier task depends strongly on the specific embedding model. Please keep in mind that the exact same training and validation sets where used for each feature extraction model.

Furthermore, even if the object categories and the feature extraction model is kept constant, the selection of the 'best' classifier varies considerable if different images of the same object category are used for training. Figure 7.8 shows the ratio of selected classifiers for the InceptionResNetV2 model for 3 different runs. Please note that for each run a the pool of annotated images was randomly split to the training and validation set. Also the set of negative training and validation images differ between runs. However, the within-model variance is not as big as the between-model variance.

Whereas the object category is not a very good indicator to determine the 'best' classifier, a stronger correlation between the number of training samples and classifier type is observable. Figure 7.9 presents the ratio of selected classifiers for each training set size. InceptionV3 and InceptionResNetV2 show very similar patterns: With an increase in the training set size, the share of the GaussianNB classifier decreases. For training set sizes ≥ 5 the LinearSVC is the most popular choice. The LinearSVC is also the

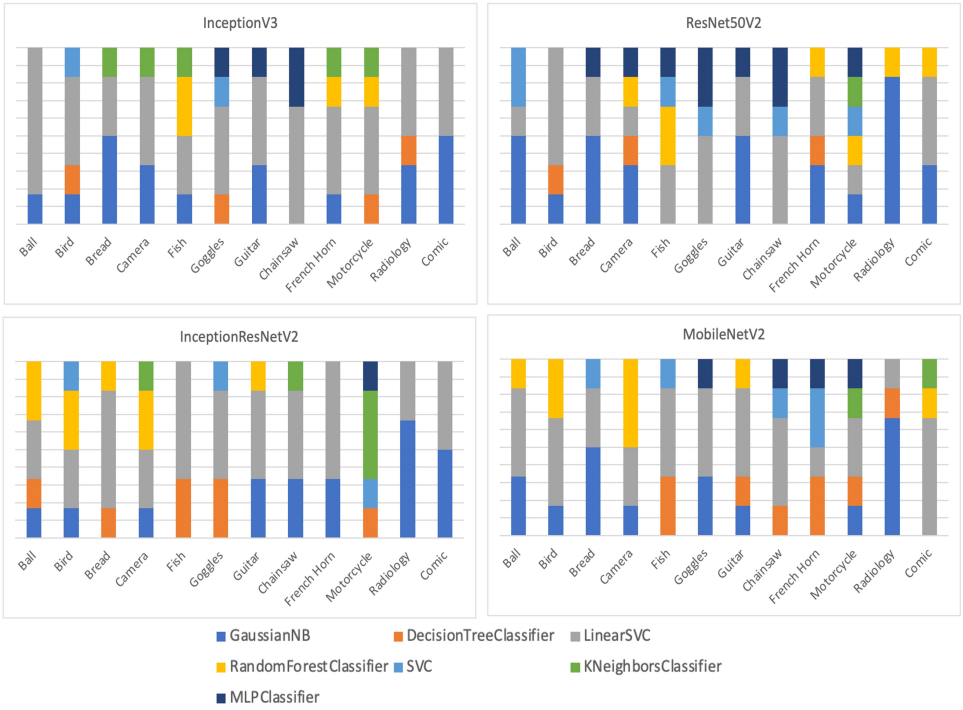


Figure 7.7: Ratio of selected classifiers for each object category for evaluation run 1

dominant classifier type for bigger sample sets for the other two networks. The Figure showing the ratio of selected classifiers for the InceptionResNetV2 model for 3 different runs can be found in the Appendix, Section A.24.

One important take away from this section is that the 'best' classifier depends on various factors, like the number of training instance, the feature extraction model, the object type and the individual images that make up the training and validation sets. It is therefore highly recommended to assign a certain share of annotated images to the validation pool. Without such validation images, a default classifier is selected (LinearSVC), which might not deliver the best results. The optimal size of the validation set is still an open question. The composition and size of the validation set has an influence on the 'best' classifier. On the other hand, annotating images just for validation without gaining any positive training effect is costly. There might be an optimal size which balances the two opposing factors. I have to leave the answer to this question to future research on the subject.

7.5 Discussion

Section 7.1 lists three research questions, the answers to which may contribute to a productive large-scale active learning application for concept detection in video. Sections 7.4.1 and 7.4.2 provided strong evidence that the InceptionResNetV2 network achieves

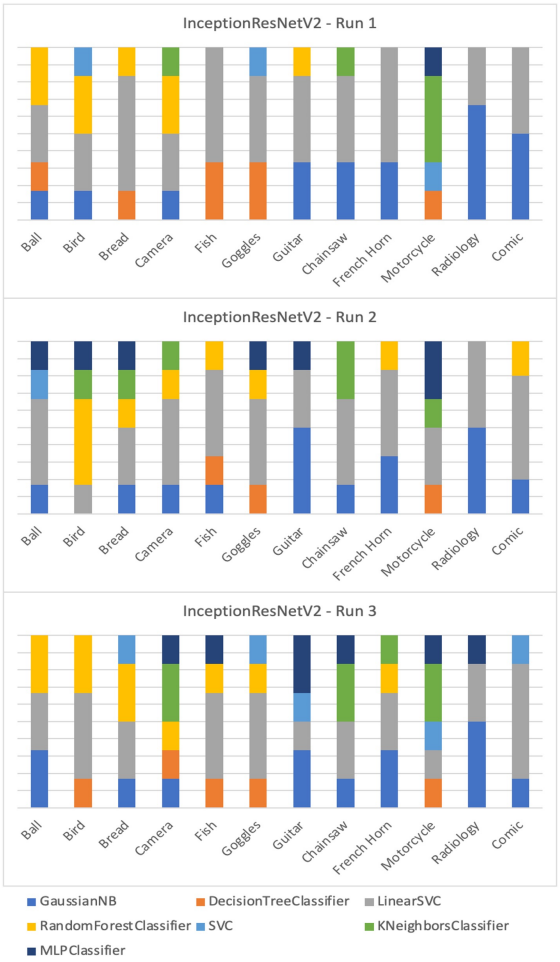


Figure 7.8: Ratio of selected classifiers for the InceptionResNetV2 model over 3 runs

the best classification performance for the proposed sequence of operations. It achieved the highest F1 scores over all object categories. Also for single object categories it is the most capable model. This is true even for small training sets. Regarding the best combination of feature extraction model and classifier, Section 7.4.4 shows that there is no single classifier which works 'best' in all classification scenarios. Each tasks requires a specific classifier. Therefore, it is recommended to let the system choose the 'best' classifier for each task. Consequently, a certain share of annotated images should be assigned to the validation pool. The optimal split ratio between training and validation pool could not be determined in this thesis. These findings answer **RQ1**. Section 7.4.2 discusses in much detail the F1 scores for single categories. Iconic view objects like *Bread*, *Guitar*, *Chainsaw*, *French Horn* and *Camera* were classified with a high F1 score even with only 5 or 10 positive training instances. For the two categories representing different concepts (*Radiology* and *Comic*) the classification performance was even better. Hence, **RQ2** can be answered with a yes for iconic view objects. Non iconic view objects like *Ball*

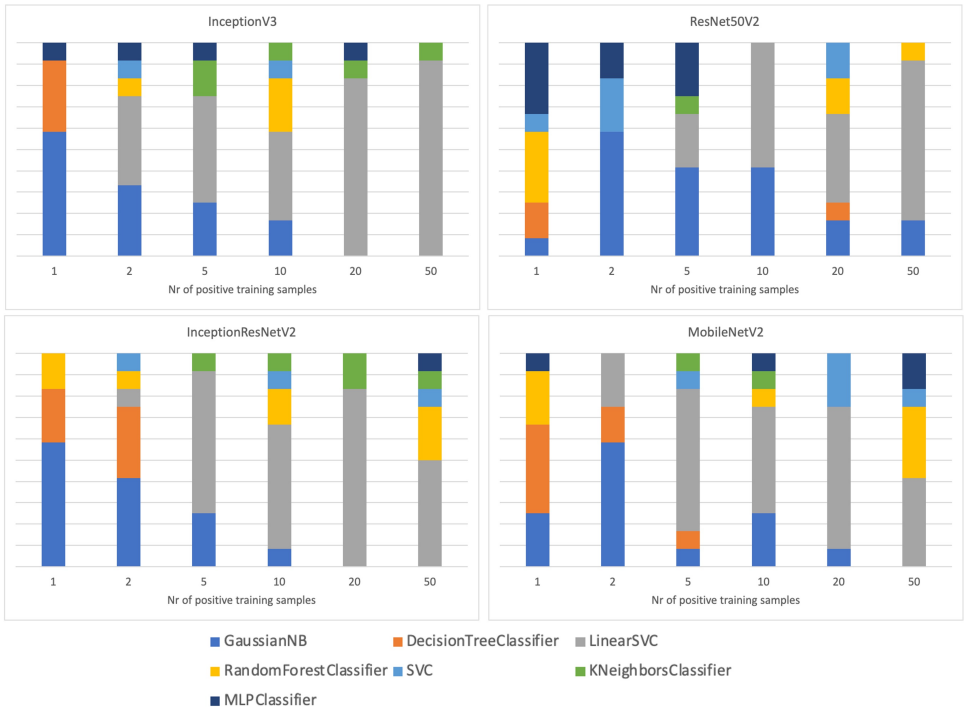


Figure 7.9: Ratio of selected classifiers for each training set size for evaluation run 1

and *Fish* required at least 20 or 50 training instances, respectively. For small training sets (≤ 5), the F1 scores for *Goggles* is 0.2 or below. This means a higher annotation effort for the user when scrolling through the predicted positive samples. The saturation tendency for many object categories is problematic. This problem should be addressed in a future study. **RQ3** asks whether it is beneficial to add extracted box images to the training set. In Section 7.4.3 I show that marginal gains can be achieved for the InceptionResNetV2 network. Therefore, it is recommended to box annotate new object categories using the VIA application in the proposed pipeline.

8

Future Work

During the processing of the thesis, many aspects of a large-scale active learning pipeline for concept detection in video could not be analyzed conclusively due to time constraints. Furthermore, the evaluation of the proposed pipeline revealed new questions, which could not be answered within the given time frame. This chapter identifies the most important open questions and discusses the limitation of the proposed pipeline. Hence, it should help to guide future research on the topic. The chapter is structured as follows: Section 8.1 discusses potential future work related to the feature extraction models. Section 8.2 presents possible research topics in connection with the classifiers. Section 8.3 explains how an adjusted active learning method could potentially improve the classification performance and proposes to investigate more on meta-learning approaches. The chapter ends with a few suggestions for further evaluation test scenarios in Section 8.4. A list of known issues and possible improvements for the pipeline implementation is omitted here. It was already discussed in Section 6.4.

8.1 Future Work on Feature Extraction

Chapter 2 discusses in great detail the different feature extraction architectures considered in this thesis. However, research progresses rapidly in this field. It would be interesting to see if future architectures can provide even better classification results. Another interesting approach is to extract features from different hidden layers of the ConvNet and combine them to a single feature representation. One could also study the effect of a condensed feature vector on the classification performance. In the current setup, all elements of the output vector are used for training and classification. It was the intention to use as much information as possible from each image. However, this comes at the cost of high computational loads and higher storage requirements. Future research could focus on the limits to which the output vector can be condensed without compromising classification performance in a few-shot classification system.

8.2 Future Work on Classifiers

All classifiers described in Chapter 3 can be parameterized for the specific classification task at hand. For instance, for the K-Nearest Neighbor classifier, the number of

neighbors, the distance metric, and the weighting function can be defined. The MLP classifier provides several hyperparameters, such as the number of hidden nodes and layers or the initial weight initialization. An optimal parameterization can further improve classification performance. It would be interesting to see if future research could derive near optimal parameter settings for each classifier. Another potential way is to introduce new classifier types which are not covered in this thesis. Also, the question whether combining different classifiers and/or combining the output feature vectors of the feature extraction models improves classification performance is a potential field of future research.

8.3 Future Work on Active Learning

Another field which holds much potential for further improvements is the field of active learning and few shot image classification techniques. In the proposed active learning method, positive classified images are presented to the users. They are then asked to confirm the most uncertain classified images which contain the object category. This provides the system image information which is highly informative. However, one downside of this method is that no informative negative training samples can be gathered. An additional functionality which allows the user to reject falsely-positive classified images would be necessary. Furthermore, such a method would require dedicated negative training set pools for each object category. Future work could be dedicated to investigate the effectiveness of such a method. Another topic of interest are few-shot techniques with meta-learners. For instance Sung et al. [Sung et al., 2018] use a meta-learning method which is based on a distance metric between images. The intuition behind this 'learning to compare' approach is that if a model can determine the similarity between two images, it can classify an unseen input image with the labelled instances. Chen et al. [Chen et al., 2019] also mention approaches which focus on transfer learning and fine-tuning methods. In transfer learning, the classification layer of a ConvNet is trained on new image objects. 'Learning to-fine tune' tries to learn an optimal model initialization and 'Learning an optimizer' replaces the stochastic gradient decent optimizer and the weight-update mechanism with an external memory. Future research could implement such meta-learning techniques and validate their effectiveness in comparison to the more traditional approach chosen for this thesis.

8.4 Ideas for further Evaluations of the Proposed Pipeline

The proposed pipeline should be evaluated with large-scale user experiments. Unfortunately, the time restrictions did not allow me to do these tests. Such user experiments could provide important insights about the optimal specification of the proposed mode of operations. For instance, it should be tested how frequently a user should be asked to annotate new images in the active learning cycles. Another open question is about the frequency at which a user should confirm positive classified images. The answers to

these question heavily depends on the annotation budget of a user. In order to motivate users to spend more time annotating images, one could also think about potential user incentives. Other future research could be dedicated to the question about the optimal validation set size. As has been discussed in Section 7.4.4, assigning more annotated images to the validation pool may result in a better classifier selection. However, it comes with the cost that these images are not available for training anymore. Further test runs might provide insights about an optimal validation set size. Another unanswered question is about the saturation tendency in certain object categories (see Section 7.4.2). Further tests should focus on this issue.

9

Conclusions

The presented Master's thesis investigated practical and effective ways to implement a large-scale active learning pipeline for concept detection in videos. The journey started with the search for suitable methods to extract the most informative features from key frame images. The feature vector space of the models should not only allow for effective object classification. It should also be large enough to distinguish a variety of different object types. Because of the outstanding success of convolutional neural networks in large-scale image classification challenges, it was natural to consider these models for feature embedding. However, the ConvNets could not be used for classification because the system is constantly confronted with new object categories. Retraining the entire network is computationally prohibitive. Also, retraining only the classification layer is not a valid alternative because the training sets for each object category are expected to be highly unbalanced. Therefore, a different approach was taken: For each object category a dedicated classifier would be trained. A preliminary evaluation of the most effective combination of feature extraction model and classifier type yielded ambiguous results. There is no unique combination of feature extraction model and classifier type which performs best in all classification scenario. Hence, the classification system should select the best classifier for each classification task at hand.

Another challenge was to find a practical method that would reduce the burden of manual image annotation as much as possible. Traditionally, supervised learning methods require huge amounts of annotated images. For a classification system which is expected to constantly learn new object categories from its users, requiring large numbers of training instances is not practical. Fortunately, active learning methods provide a framework to address this challenge. The aim of active learning is to select only the most informative images for the classification system. The proposed pipeline uses a simple uncertainty based heuristic to query the most informative images. The user is then asked to confirm those images which contain the object of interest. However, this approach requires an initial training cycle and thus a few initial training instances. In order to further improve the classification performance on very small training set, I experimented with box annotation of objects. It was found that the classification performance can be improved for certain feature extraction models when extracted box objects are added to the training set.

A final evaluation of the proposed active learning pipeline found that the Inception-ResNetV2 network is best suited for the task. It not only delivered the best performance over all classification scenarios, it also showed the best results for individual object categories and for small training sets. Adding extracted box images as training instances further improved the classification performance with the InceptionResNetV2 model. The evaluation also confirmed that there is no classifier type which works best in all classification scenarios. However, for the system to select the 'best' classifier, it requires a set of validation images to evaluate the classification performance of each classifier. The optimal trade-off between gaining information about the 'best' classifier and loosing potential training instances could not be answered conclusively. This question is left to future research on the subject.

References

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Alpaydin, 2014] Alpaydin, E. (2014). *Multilayer Perceptrons*, pages 267–316. The MIT Press, 3rd edition.

[Angluin, 1988] Angluin, D. (1988). Queries and concept learning. *Machine Learning*, 2:319–342.

[Arora et al., 2014] Arora, S., Bhaskara, A., Ge, R., and Ma, T. (2014). Provable bounds for learning some deep representations. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 584–592, Bejing, China. PMLR.

[Baum and Lang, 1992] Baum, E. B. and Lang, K. (1992). Query learning can work poorly when a human oracle is used. *Proc. Int. Joint Conf. Neural Netw*, pages 335–340.

[Beluch et al., 2018] Beluch, W. H., Genewein, T., Nurnberger, A., and Kohler, J. M. (2018). The power of ensembles for active learning in image classification. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9368–9377.

[Boser et al., 1992] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA. Association for Computing Machinery.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

[Breiman et al., 1984] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA.

[Buda et al., 2017] Buda, M., Maki, A., and Mazurowski, M. (2017). A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106.

[Chen et al., 2019] Chen, W., Liu, Y., Kira, Z., Wang, Y. F., and Huang, J. (2019). A closer look at few-shot classification. *CoRR*, abs/1904.04232.

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. *https://keras.io*.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support vector networks. *Machine Learning*, 20:273–297.

[Domingos and Pazzani, 1997] Domingos, P. and Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2):103–130.

[Dutta et al., 2016] Dutta, A., Gupta, A., and Zissermann, A. (2016). (vgg) image annotator (via). http://www.robots.ox.ac.uk/ vgg/software/via/. Version: 2.0.11, Accessed: 14.05.2021.

[Dutta and Zisserman, 2019] Dutta, A. and Zisserman, A. (2019). The VIA annotation software for images, audio and video. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, New York, NY, USA. ACM.

[Fix and Hodges, 1951] Fix, E. and Hodges, J. L. (1951). Discriminatory analysis. nonparametric discrimination: Consistency properties. Technical report, USAF School of Aviation Medicine, Randolph Field, Texas.

[Friedman, 1991] Friedman, J. H. (1991). Multivariate Adaptive Regression Splines. *The Annals of Statistics*, 19(1):1–67.

[Gal et al., 2017] Gal, Y., Islam, R., and Ghahramani, Z. (2017). Deep bayesian active learning with image data. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1183–1192. JMLR.org.

[Gasser et al., 2020] Gasser, R., Rossetto, L., Heller, S., and Schuldt, H. (2020). Cottontail db: An open source database system for multimedia retrieval and analysis. In *Proceedings of the 28th ACM International Conference on Multimedia*, MM '20, pages 4465–4468, New York, NY, USA. Association for Computing Machinery.

[Goldberger et al., 2004] Goldberger, J., Roweis, S., Hinton, G., and Salakhutdinov, R. (2004). Neighbourhood components analysis. In *Proceedings of the 17th International Conference on Neural Information Processing Systems*, NIPS 04, pages 513–520, Cambridge, MA, USA. MIT Press.

[Hammel et al., 2019] Hammel, H., Howard, J., Turgutlu, K., Wright, L., and Varty, J. (2019). Imagenette. https://github.com/fastai/imagenette. Accessed: 15.08.2021.

[He and Sun, 2015] He, K. and Sun, J. (2015). Convolutional neural networks at constrained time cost. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5353–5360.

[He et al., 2016a] He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

[He et al., 2016b] He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Identity mappings in deep residual networks. In Leibe, B., Matas, J., Sebe, N., and Welling, M., editors, *Computer Vision – ECCV 2016*, pages 630–645, Cham. Springer International Publishing.

[Hebb, 1949] Hebb, D. (1949). *The Organization of Behavior: A Neuropsychological Theory*. John Wiley And Sons, Inc, New York.

[Hemmer et al., 2020] Hemmer, P., Kühl, N., and Schöffer, J. (2020). DEAL: deep evidential active learning for image classification. *CoRR*, abs/2007.11344.

[Howard et al., 2017] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications.

[I. Goodfellow and Courville, 2016] I. Goodfellow, Y. B. and Courville, A. (2016). *Deep Learning*. The MIT Press, Cambridge.

[IBM Cloud Education, 2020] IBM Cloud Education (2020). Convolutional neural networks. *https:// www.ibm.com/ cloud/ learn/ convolutional-neural-networks*. Accessed: 03.05.2021.

[Kamel et al., 2019] Kamel, H., Abdulah, D., and Al-Tuwaijari, J. M. (2019). Cancer classification using gaussian naive bayes algorithm. In *2019 International Engineering Conference (IEC)*, pages 165–170.

[Kelly and Johnson, 2021] Kelly, A. and Johnson, M. A. (2021). Investigating the statistical assumptions of naÃ¯ve bayes classifiers. In *2021 55th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–6.

[Kohavi and Quinlan, 1999] Kohavi, R. and Quinlan, R. (1999). Decision tree discovery. In *Handbook of data mining and knowledge discovery*, pages 267–276. University Press.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[Kuznetsova et al., 2020] Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Kolesnikov, A., and et al. (2020).

The open images dataset v4. *International Journal of Computer Vision*, 128(7):1956–1981.

[Lin et al., 2014] Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Doll'a r, P., and Zitnick, C. L. (2014). Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312.

[Loh, 2014] Loh, W.-Y. (2014). Fifty years of classification and regression trees. *International Statistical Review*, 82.

[Malov, 2021] Malov, Y. (2021). Simpsons. https://www.kaggle.com/ymalov/simpsons. Accessed: 15.08.2021.

[Mooney, 2017] Mooney, P. T. (2017). Chest x-ray images (pneumonia). https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia. Accessed: 16.08.2021.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Platt, 1999] Platt, J. C. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *ADVANCES IN LARGE MARGIN CLASSIFIERS*, pages 61–74. MIT Press.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Rish, 2001] Rish, I. (2001). An empirical study of the naive bayes classifier. *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, 3.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408.

[Rossetto et al., 2014] Rossetto, L., Giangreco, I., and Schuldt, H. (2014). Cineast: A multi-feature sketch-based video retrieval engine. In *2014 IEEE International Symposium on Multimedia*, pages 18–23.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

[Sandler et al., 2018] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520.

[Schohn and Cohn, 2000] Schohn, G. and Cohn, D. (2000). Less is more: Active learning with support vector machines. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 839–846, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Sener and Savarese, 2018] Sener, O. and Savarese, S. (2018). Active learning for convolutional neural networks: A core-set approach.

[Sivic and Zisserman, 2003] Sivic and Zisserman (2003). Video google: a text retrieval approach to object matching in videos. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 1470–1477 vol.2.

[Sonquist and Morgan, 1964] Sonquist, J. and Morgan, J. (1964). The detection of interaction effects : a report on a computer program for the selection of optimal combinations of explanatory variables. University of Michigan.

[Spence, 2002] Spence, R. (2002). Rapid, serial and visual: A presentation technique with potential. *Information Visualization*, 1(1):13–19.

[Sung et al., 2018] Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H., and Hospedales, T. M. (2018). Learning to compare: Relation network for few-shot learning. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1199–1208.

[Szegedy et al., 2016a] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2016a). Inception-v4, inception-resnet and the impact of residual connections on learning. *AAAI Conference on Artificial Intelligence*.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9.

[Szegedy et al., 2016b] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016b). Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826.

[The TensorFlow Team, 2019] The TensorFlow Team (2019). Flowers. http://download.tensorflow.org/example_images/flower_photos.tgz. Accessed: 14.04.2021.

[Wang et al., 2016] Wang, K., Zhang, D., Li, Y., Zhang, R., and Lin, L. (2016). Cost-effective active learning for deep image classification. *IEEE Transactions on Circuits and Systems for Video Technology*, 27:1–1.

[Xu et al., 2012] Xu, B., Ye, Y., and Nie, L. (2012). An improved random forest classifier for image classification. In *2012 IEEE International Conference on Information and Automation*, pages 795–800.

[Yoo and Kweon, 2019] Yoo, D. and Kweon, I. S. (2019). Learning loss for active learning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 93–102.

[Zhang et al., 2020] Zhang, L., Sun, J., Wang, T., Min, Y., and Lu, H. (2020). Visual saliency detection via kernelized subspace ranking with active learning. *IEEE Transactions on Image Processing*, 29:2258–2270.

A

Appendix

A.1 Network Architectures

This section contains complementary information for the network architectures presented in Chapter 2.

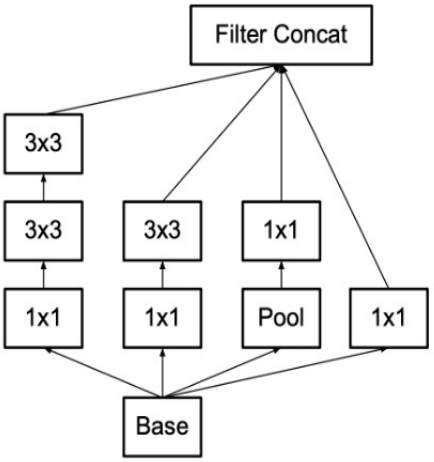
A.1.1 InceptionV3

The following table provides an overview of the InceptionV3 network architecture from section 2.2.3.

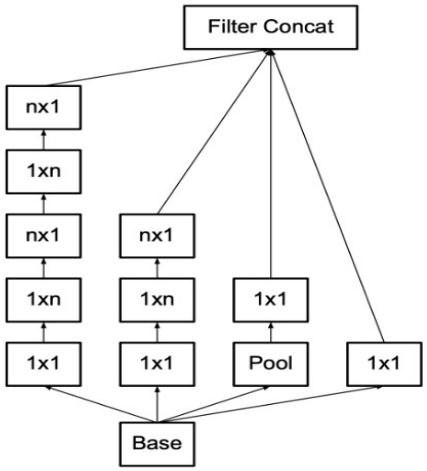
Type	**Patch Size / Stride**	**Input Size**
Conv	3×3 / 2	$299 \times 299 \times 3$
Conv	3×3 / 1	$149 \times 149 \times 32$
Conv padded	3×3 / 1	$147 \times 147 \times 32$
Pool	3×3 / 2	$147 \times 147 \times 64$
Conv	3×3 / 1	$73 \times 73 \times 64$
Conv	3×3 / 2	$71 \times 71 \times 80$
Conv	3×3 / 1	$35 \times 35 \times 192$
3× Inception	as in A.1	$35 \times 35 \times 288$
5× Inception	as in A.2	$17 \times 17 \times 768$
2× Inception	as in A.3	$8 \times 8 \times 1280$
Pool	8×8	$8 \times 8 \times 2048$
Linear	Logits	$1 \times 1 \times 2048$
Softmax	Classifier	$1 \times 1 \times 1000$

Table A.1: Outline of the InceptionV3 network architecture

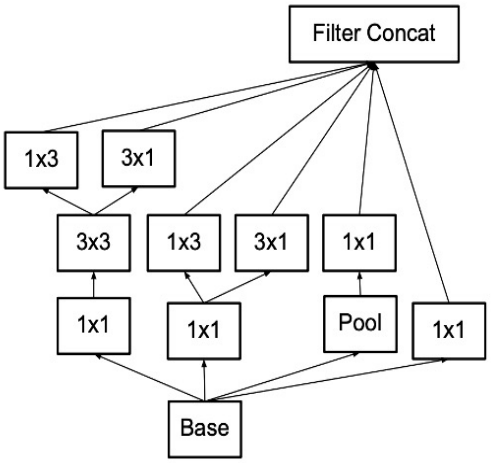
Source: [Szegedy et al., 2016b]



Source: [Szegedy et al., 2016b]
Figure A.1: Inception modules where each 5×5 convolution is replaced by two 3×3 convolution.



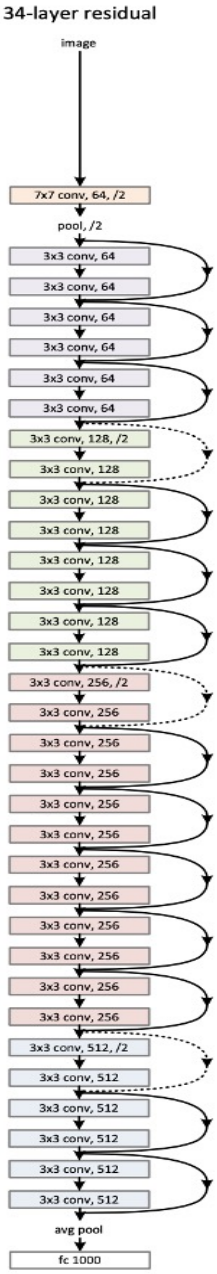
Source: [Szegedy et al., 2016b]
Figure A.2: Inception modules after the factorization of the $n \times n$ convolutions. In the proposed architecture, the authors chose $n = 7$ for the 17×17 grid.



Source: [Szegedy et al., 2016b]
Figure A.3: Inception modules with expanded filter bank outputs. This architecture is used on the coarsest (8×8) grids to promote high dimensional representations.

A.1.2 ResNetV2

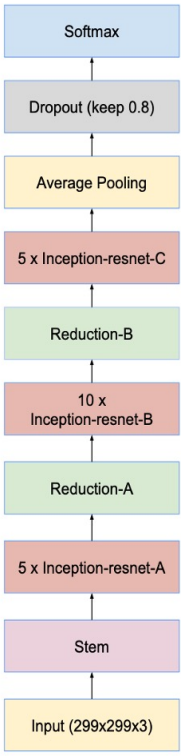
Figure A.4 shows a schematic ResNet network with 34 parameter layers. Please note that the ResNetV2 network used in the thesis consists of 50 parameter layers.



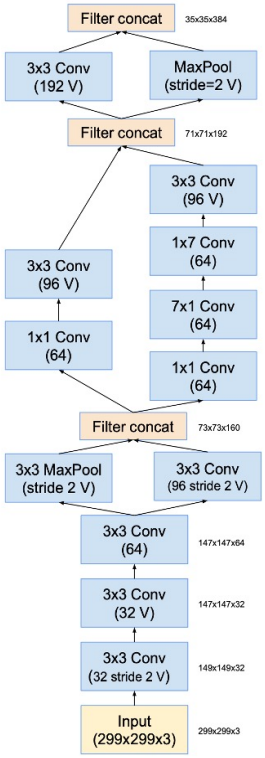
Source: [He et al., 2016a]
Figure A.4: ResNet with 34 parameter layers. The dotted shortcuts increase dimensions.

A.1.3 InceptionResNetV2

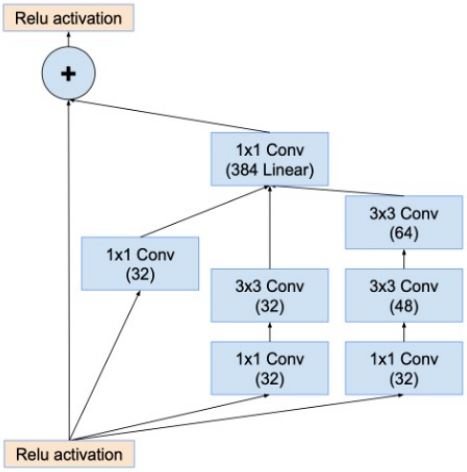
The following figures provide supplement details for the InceptionResNetV2 architecture from section 2.4. Figure A.5 shows the general schema and figures A.6, A.7, A.8, A.9, A.10 and A.11 its components.



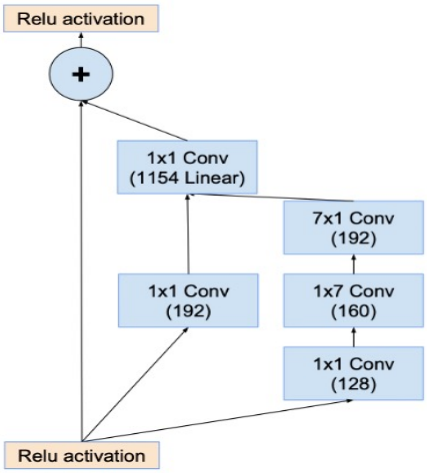
Source: [Szegedy et al., 2016a]
Figure A.5: Schema for the InceptionResNetV2 network



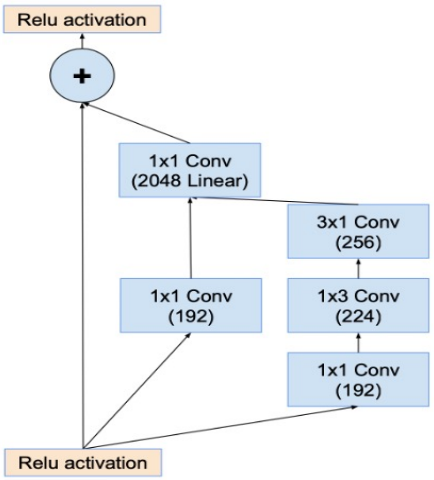
Source: [Szegedy et al., 2016a]
Figure A.6: Stem of the InceptionResNetV2 network



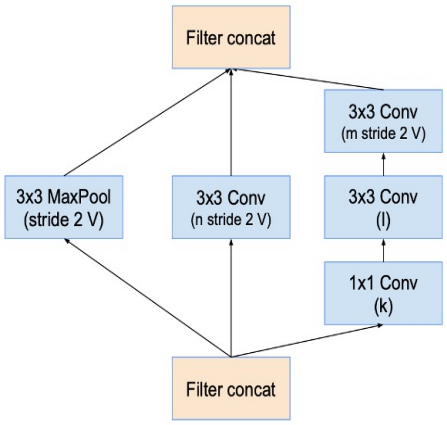
Source: [Szegedy et al., 2016a]
Figure A.7: Inception-resnet-A Block



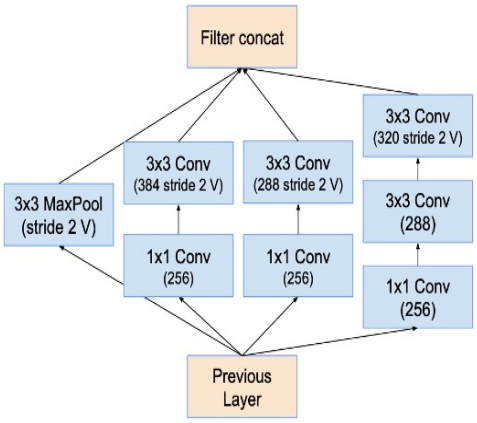
Source: [Szegedy et al., 2016a]
Figure A.8: Inception-resnet-B Block



Source: [Szegedy et al., 2016a]
Figure A.9: Inception-resnet-C Block



Source: [Szegedy et al., 2016a]
Figure A.10: Reduction-A Block



Source: [Szegedy et al., 2016a]
Figure A.11: Reduction-B Block

A.1.4 MobileNetV2

The following figures provide supplement details for the MobileNetV2 architecture from section 2.5.3. Figure A.12 shows the architecture of a linear bottleneck layer and figure A.13 describes the general layout of the MobileNetV2 network.

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Source: [Sandler et al., 2018]
Figure A.12: Bottleneck residual block transforming from k to k' channels, with stride s, and expansion factor t. h is the height and w the width of the input

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Source: [Sandler et al., 2018]

Figure A.13: Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated n times. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor t is always applied to the input size

A.1.5 Keras Pre-trained Models

The following table provides an overview of the pre-trained models used in this thesis:

Table A.2: Pre-trained models from Keras

Model	**Size**	**Top-1**	**Top-5**	**Parameters**	**Depth**
InceptionV3	92 MB	0.779	0.937	23'851'784	159
ResNet50V2	98 MB	0.749	0.921	25'636'712	-
InceptionResNetV2	215 MB	0.803	0.953	55'873'736	572
MobileNetV2	14 MB	0.713	0.901	3,538,984	88

Source: [Chollet et al., 2015]

A.2 Feature Extraction Models and Classifier Evaluation Results

The following section reports additional evaluation results from Section 4.4

A.2.1 Results Coco Data Set

This Section reports additional result plots for the Coco data set.

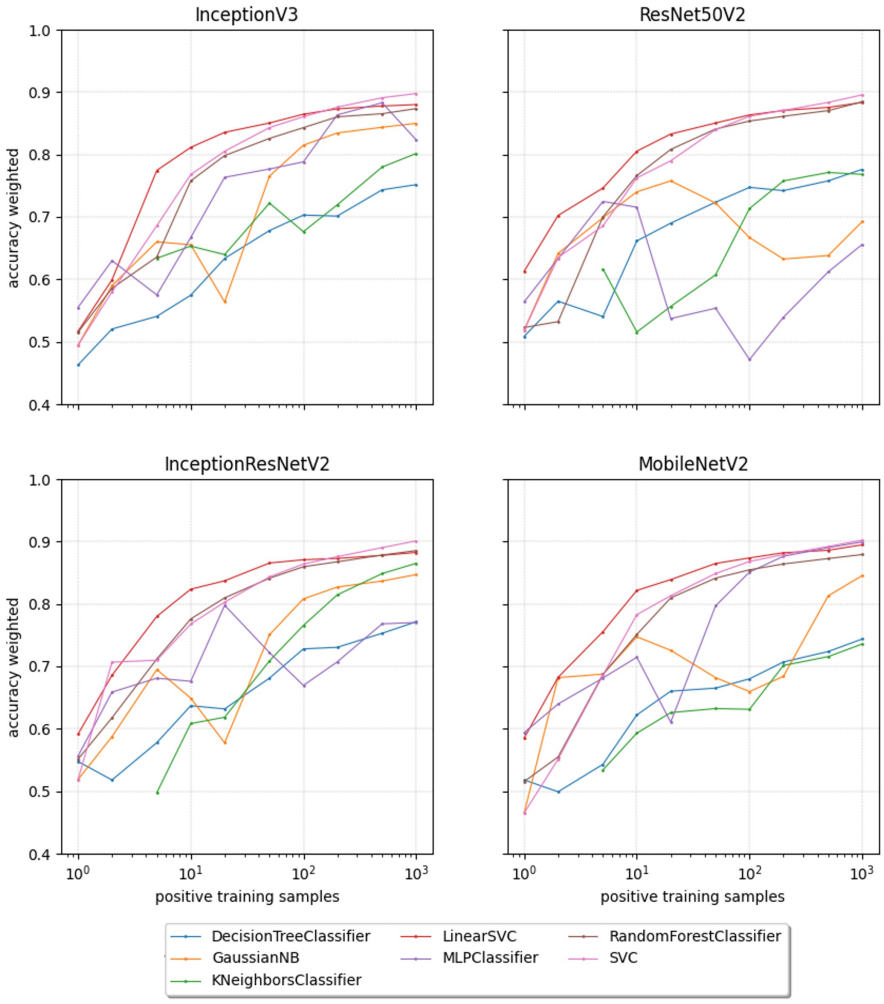


Figure A.14: Weighted Accuracy scores for each feature extraction model of the Coco data set

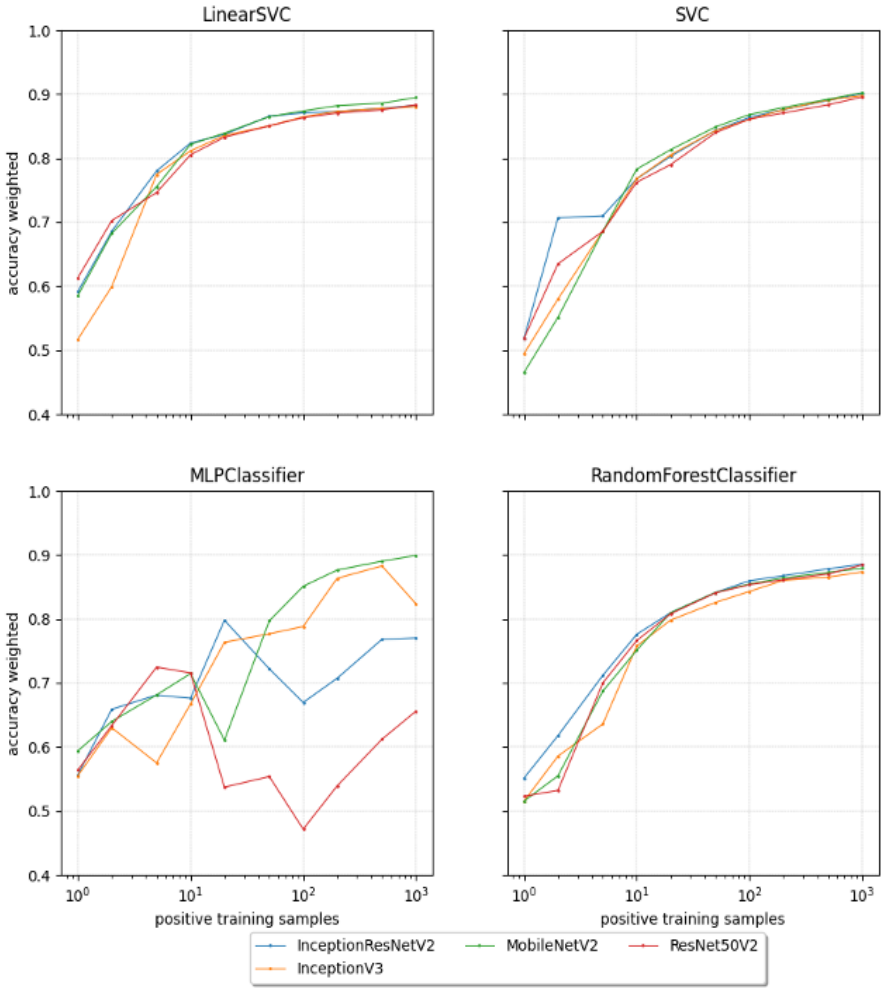


Figure A.15: Weighted Accuracy scores for selected classifiers of the Coco data set

A.2.2 Results CocoBox Data Set

This section reports additional result plots for the CocoBox data set.

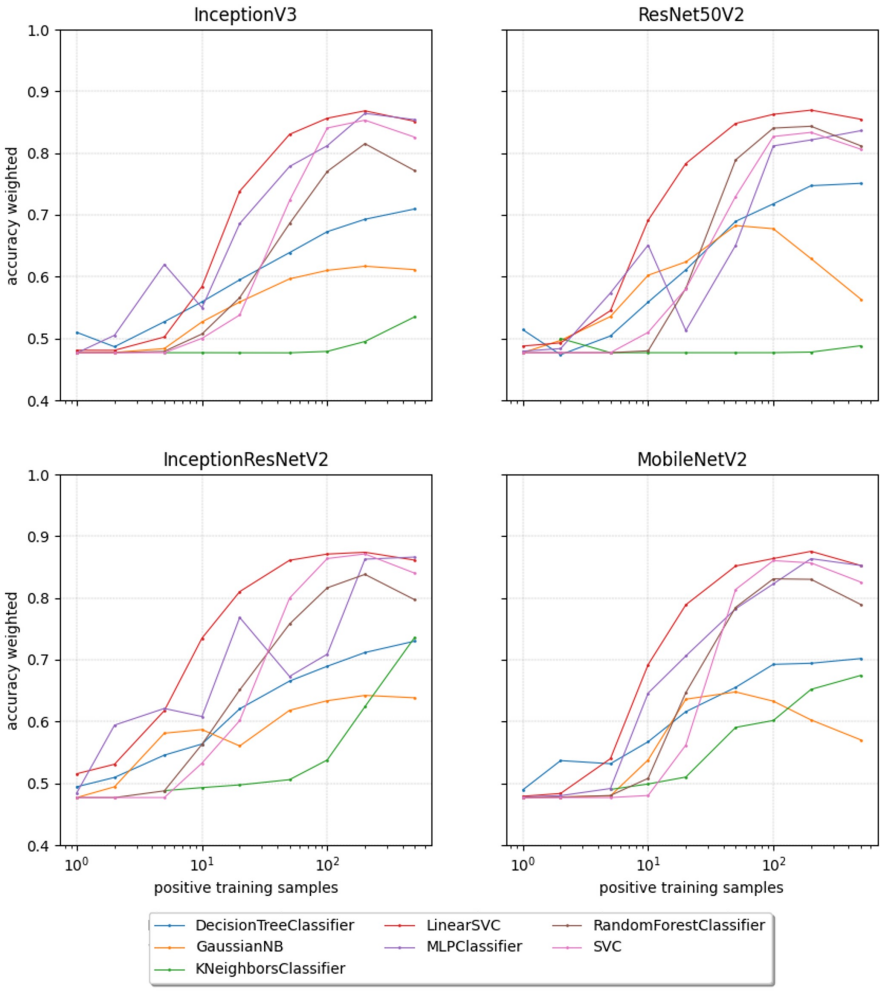


Figure A.16: Weighted Accuracy scores for each feature extraction model for the CocoBox data set

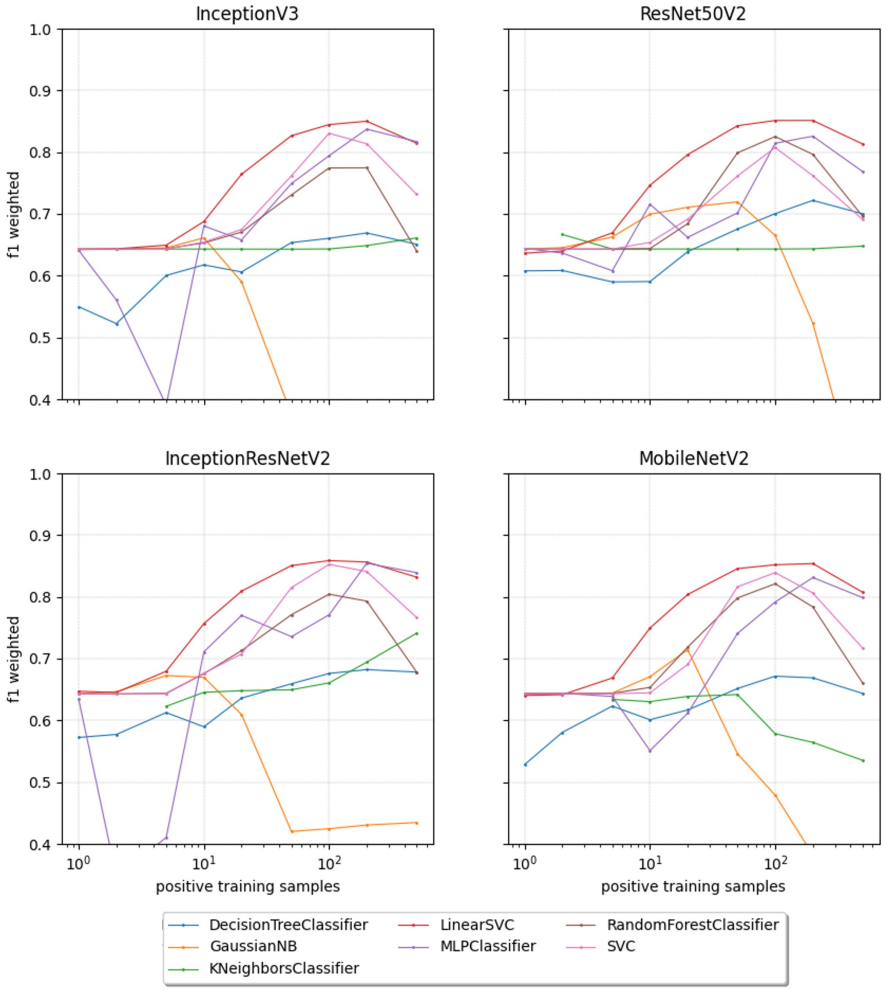


Figure A.17: Weighted F1 scores for each feature extraction model for the CocoBox data set

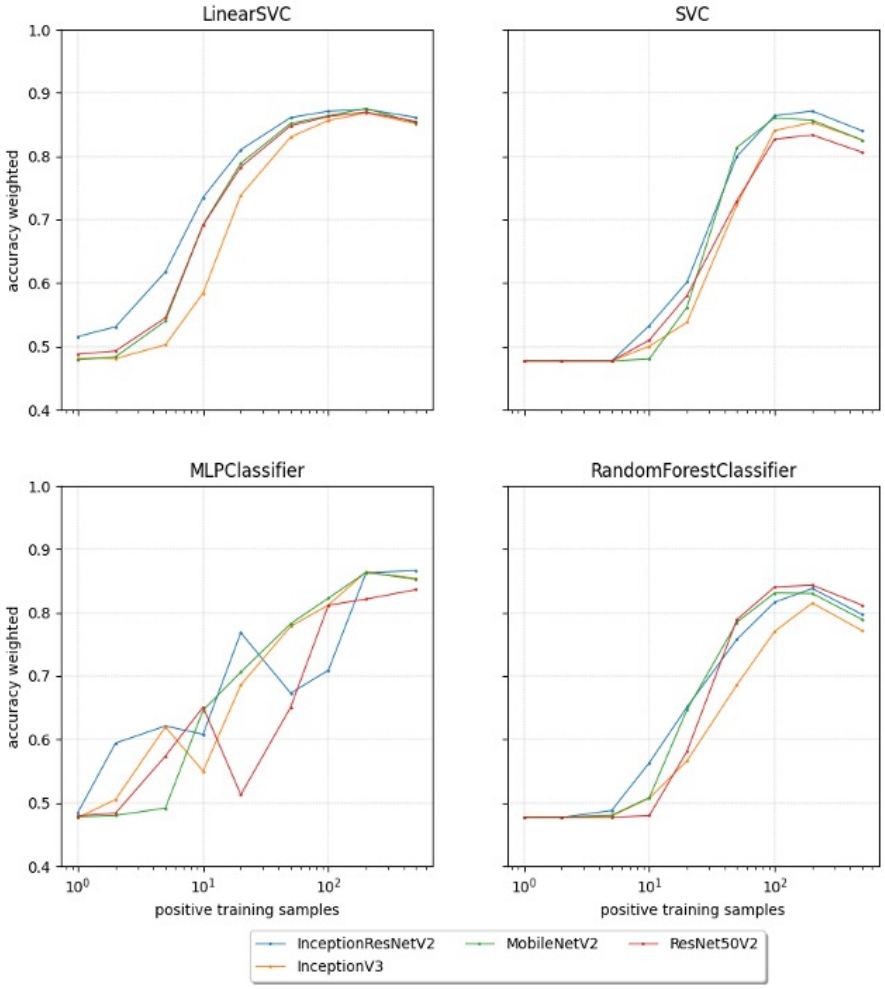


Figure A.18: Weighted Accuracy scores for selected classifiers for the CocoBox data set

A.2.3 Results TF_Flowers Data Set

This section reports additional result plots for the tf_flowers data set.

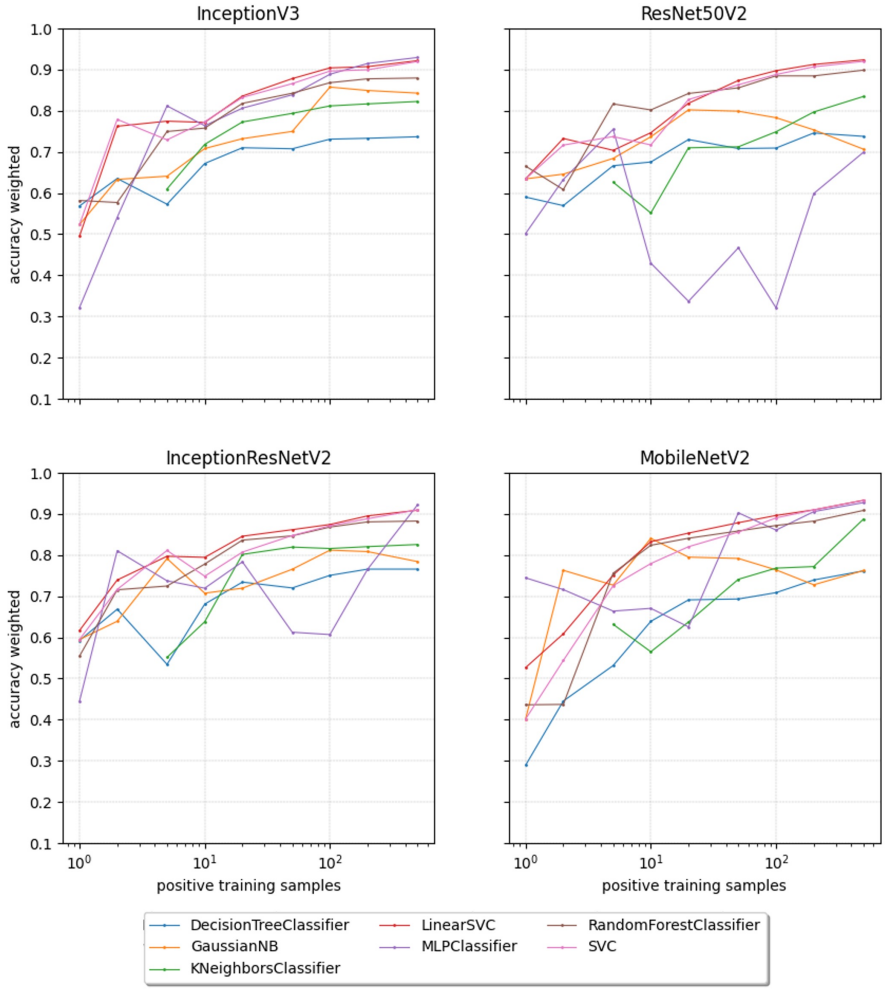


Figure A.19: Weighted Accuracy scores for each feature extraction model for the tf_flowers data set

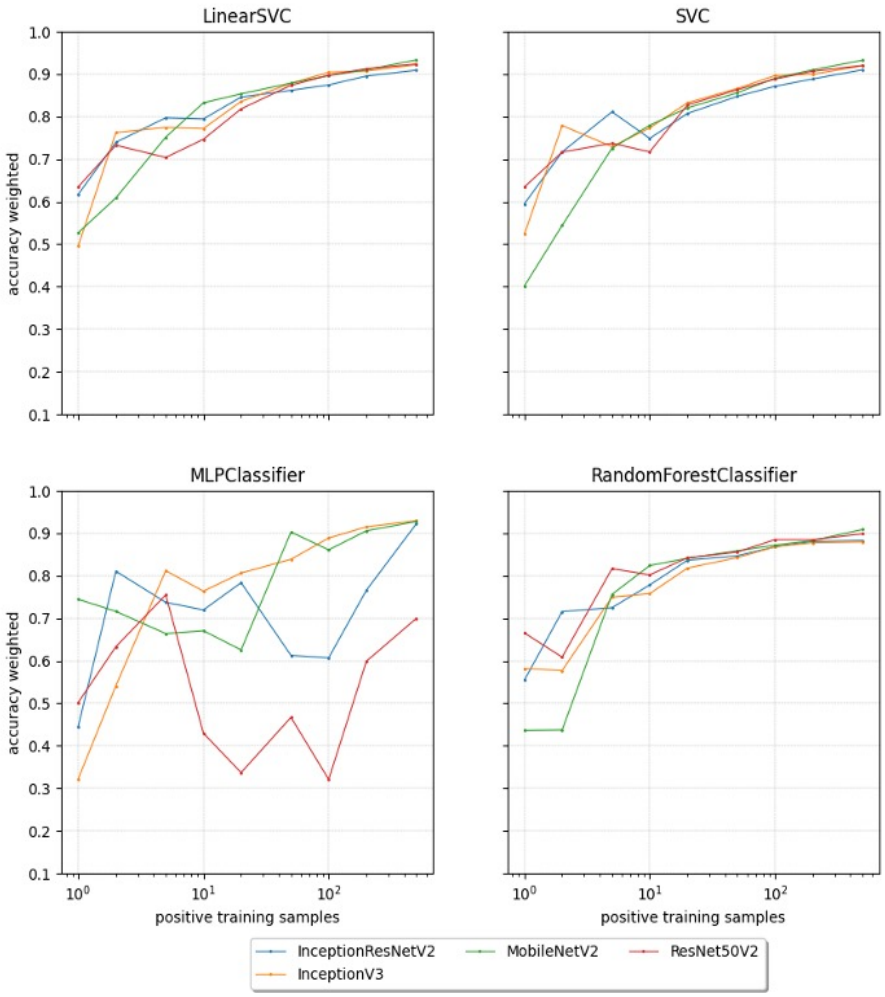


Figure A.20: Weighted Accuracy scores for selected classifiers for the tf_flowers data set

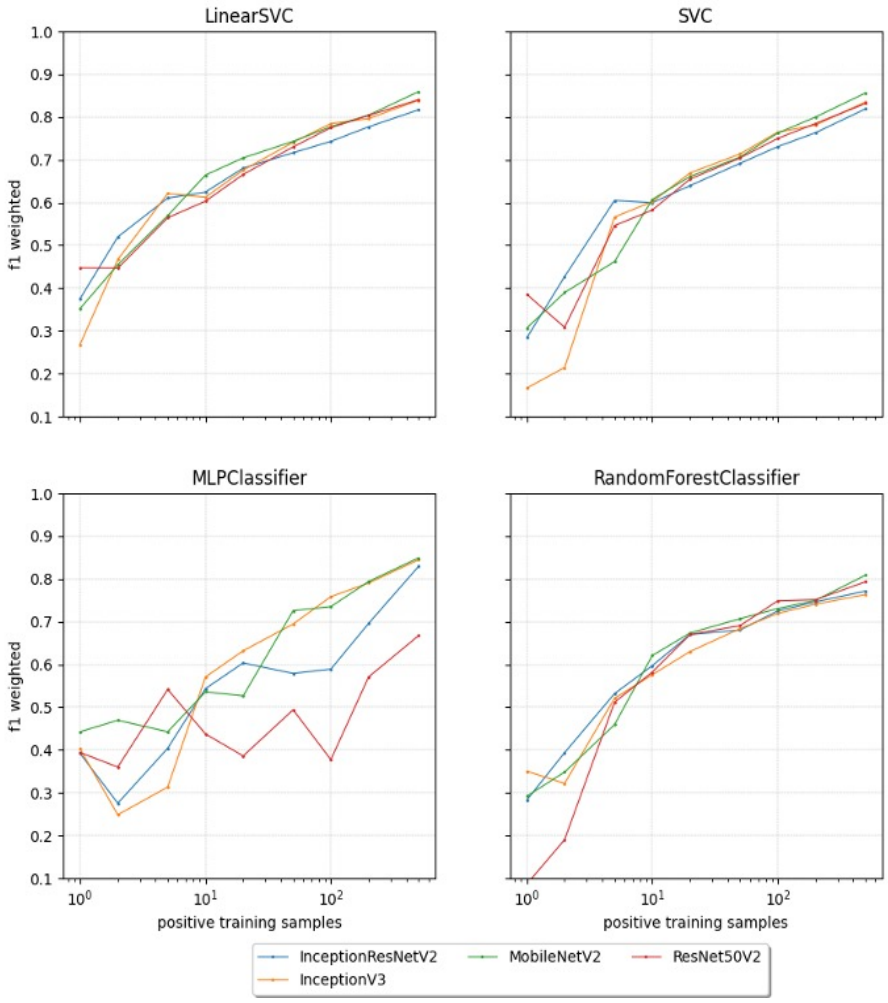


Figure A.21: Weighted F1 scores for selected classifiers for the tf_flowers data set

A.3 Further Results of the Pipeline Evaluation

This section reports additional evaluation results from Chapter 7

A.3.1 Performance Results Over All Object Categories

This section presents the average weighted Accuracy scores over all object categories

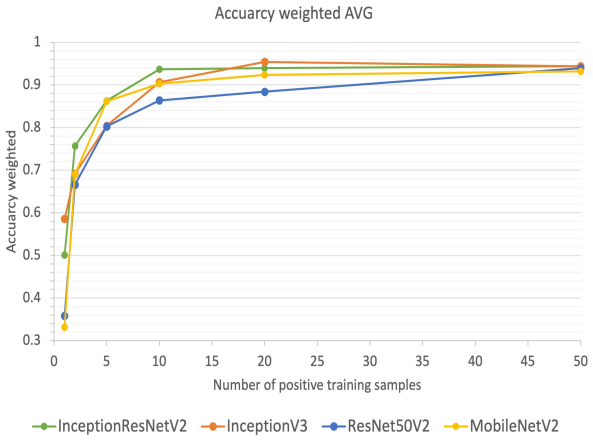


Figure A.22: Accuracy for each feature extraction model over all object object categories

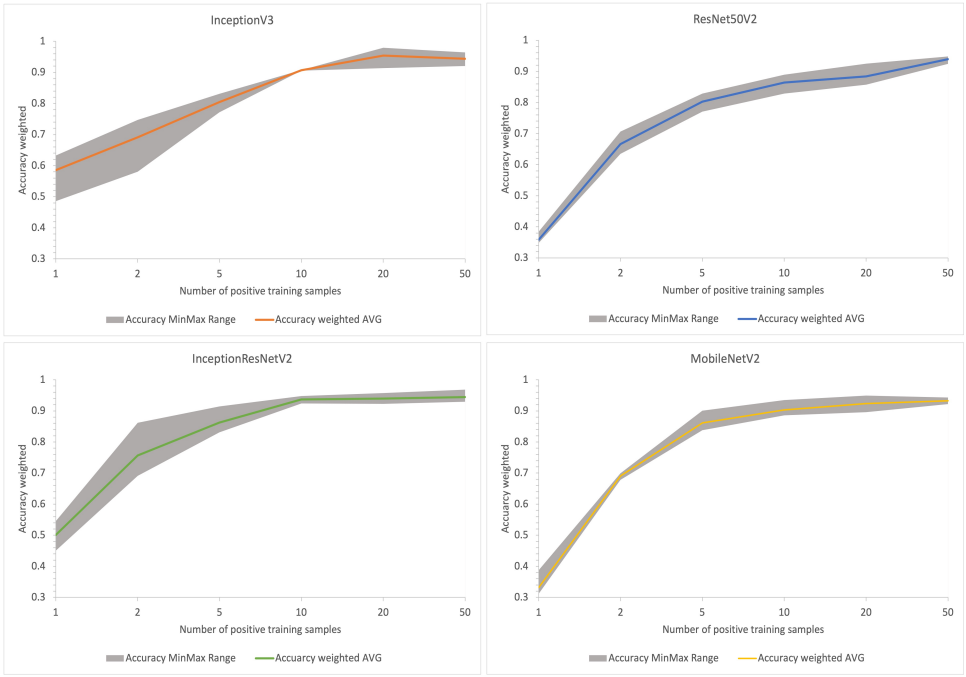


Figure A.23: Average and MinMax range of the Accuracy scores over three separate evaluation runs

A.3.2 Classifier Selection Results

This section presents an additional graphic for Section 7.4.4.

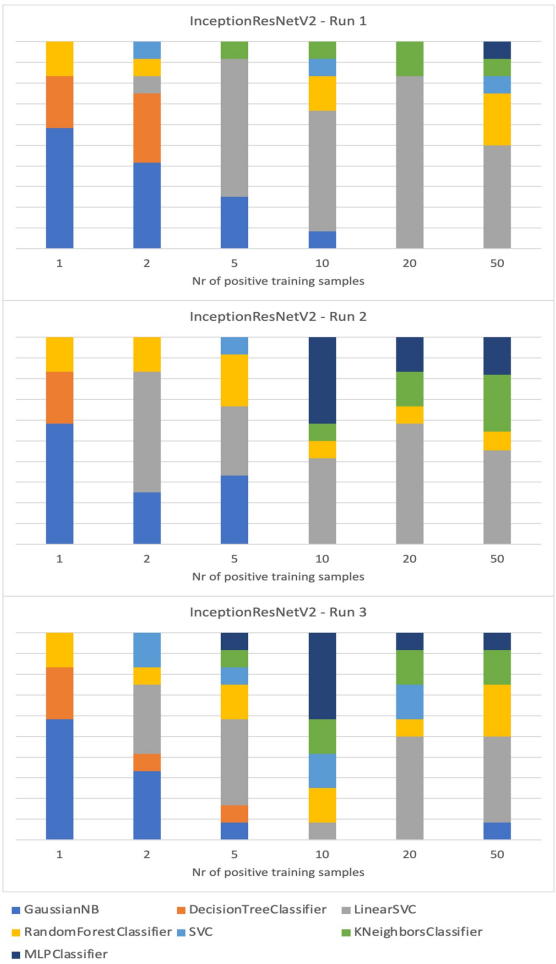


Figure A.24: Ratio of selected classifiers for the InceptionResNetV2 model over 3 runs

A.4 The VIA application - Tutorial

This section contains brief instructions on how to use the box application VIA by Dutta et al. [Dutta et al., 2016] and [Dutta and Zisserman, 2019]. Figure A.25 illustrates the box annotation process.

1. Navigate to the **Annotation** section on the entry page.
2. Enter the number of randomly selected images to be annotated and click the 'Start Annotation' Button. The VIA application starts automatically.
3. Select an image for annotation.
4. If the object is unknown to the system, define an **id** and a **label**. **IMPORTANT: The id must be a continuously increasing integer number.** VIA cannot handle gaps between ids. Furthermore, do not delete the first entry with **id: 1**.

5. Draw a box and select the correct label. Continue until all object of interest are annotated.

6. Finish the process by either clicking on the rightmost icon or navigate to *Process – End Annotation Process*.

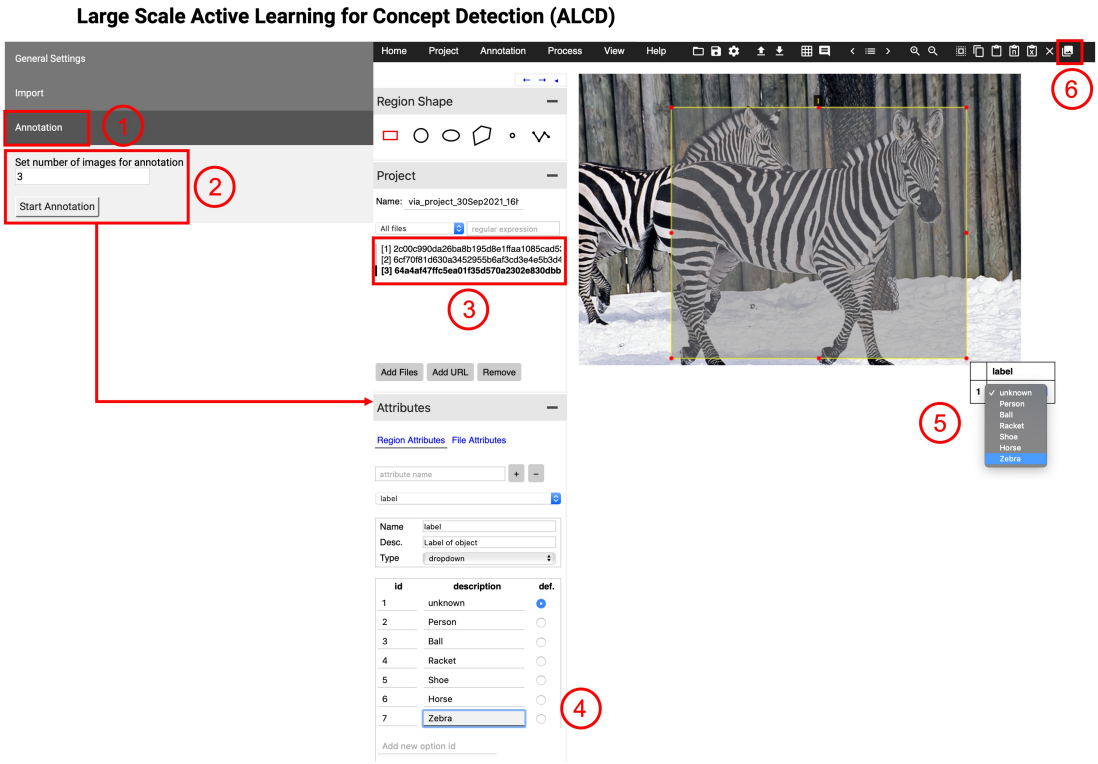


Figure A.25: Tutorial for Box annotation with the VIA application

List of Figures

List of Tables