FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Person Classification with Convolutional Neural Networks

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Visual Computing

by

Georg Sperl, BSc
Registration Number 1025854

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Robert Sablatnig
Assistance: Univ.Lektor Dipl.-Ing. Dr.techn. Roman Pflugfelder

Vienna, 25th February, 2016

Georg Sperl	Robert Sablatnig

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Personenklassifikation mit Faltenden Neuronalen Netzwerken

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

Visual Computing

eingereicht von

Georg Sperl, BSc
Matrikelnummer 1025854

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Robert Sablatnig
Mitwirkung: Univ.Lektor Dipl.-Ing. Dr.techn. Roman Pflugfelder

Wien, 25. Februar 2016

Georg Sperl	Robert Sablatnig

Erklärung zur Verfassung der Arbeit

Georg Sperl, BSc
Borromäumstraße 34, 2362 Biedermannsdorf, Austria

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

Wien, 25. Februar 2016

Georg Sperl

Acknowledgements

I would like to thank Roman Pflugfelder, my advisor at the Austrian Institute of Technology (AIT), for friendly and open-minded supervision and enabling work on this thesis at the AIT. Next, I want to thank the AIT for sponsoring my work and providing me with a workplace and the necessary hardware. I am also grateful to Csaba Beleznai for supplying me with a steady flow of information on state-of-the-art trends and interesting links on convolutional neural networks. Furthermore, I want to thank Robert Sablatnig of the Technical University of Vienna for his supervision and the well-conducted seminars on scientific writing and presentation, in which I have gained helpful knowledge for this thesis. Lastly, I am grateful to Sebastian Zambanini for resolving several questions and doubts.

Danksagung

Ich möchte mich bei Roman Pflugfelder, meinem Betreuer am Austrian Institute of Technology (AIT), für den freundlichen Umgang und die aufgeschlossene Betreuung bedanken und für die Ermöglichung der Durchführung meiner Arbeit am AIT. Außerdem bedanke ich mich beim AIT für die finanizielle Unterstützung und dafür, dass ich mit einem Arbeitsplatz und der notwendigen Hardware ausgestattet wurde. Weiters bin ich Csaba Beleznai für den stetigen Strom an Trends und interessanten Links zum Stand der Technik von faltenden neuronalen Netzwerken dankbar. Ich möchte mich auch bei Robert Sablatnig bedanken. Seine Betreuung und die gut geführten Seminare über wissenschaftliches Schreiben und Präsentieren haben mir wertvolles Wissen für diese Arbeit vermittelt. Schließlich bin ich noch Sebastian Zambanini für das Klären einiger Fragen und Zweifel dankbar.

Abstract

Large-scale object recognition challenges such as the ImageNet Large Scale Visual Object Recognition Challenge or the Microsoft Common Objects in Context challenge have shown that convolutional neural networks achieve state-of-the-art performance on computer vision problems like object detection and image classification. Convolutional neural networks benefit from datasets of hundreds of thousands of images, which cover more intraclass variabilities and aid in learning robust and invariant features. However, these datasets are designed for general object recognition and no dataset of similar dimensions exist for person recognition. Therefore, data is collected from over 30 datasets for person detection, classification, segmentation and tracking to form a pool of data sources for person recognition. A method of extracting application-specific data from this pool and training a convolutional neural network for binary person classification is proposed. Additionally, performance improvements of subclass labeling are analyzed for the nonperson class and an error rate of 2.82% is achieved. Results demonstrate that using our person recognition dataset as a pre-training set for person classification tasks with training sets of only up to a few thousand images leads to an increase in accuracy of over 8% to a total accuracy of over 99%. The quality of our dataset is demonstrated by additional evaluation. Furthermore, results emphasize the complexity of convolutional neural network architecture choice and indicate increased robustness in training with subclass labeling with regards to initialization and solver algorithms.

Kurzfassung

Großangelegte Objekterkennungswettbewerbe wie zum Beispiel das ImageNet Large Scale Visual Object Recognition Challenge oder Microsoft Common Objects in Context haben gezeigt, dass faltende neuronale Netzwerke den Stand der Technik der Leistung in Computer Vision Aufgaben wie Objektdetektion und Bildklassifikation erreichen. Faltende neuronale Netzwerke profitieren von Datensätzen aus hunderttausenden Bildern, die mehr Intra-Klassen-Variabilitäten abdecken und helfen, robuste und invariante Merkmale zu lernen. Allerdings sind dies Datensätze für allgemeine Objekterkennung und es existiert kein Datensatz für Personenerkennung, der ähnliche Ausmaße hat. Daher werden Daten von über 30 Datensätzen für Personendetektion, Personenklassifikation, Personensegmentation und Personenverfolgung gesammelt, um einen Topf von Datenquellen für Personenerkennung zu formen. Es wird eine Methode für das Extrahieren von anwendungsspezifischen Daten aus diesem Topf und das Trainieren von einem faltenden neuronalen Netzwerk für binäre Personenklassifikation vorgeschlagen. Weiters werden die Leistungsverbesserungen durch Subklassenannotation der Nicht-Personen-Klasse analysiert und eine Fehlerquote von 2.82% wird erreicht. Resultate zeigen, dass die Verwendung von unserem Personenerkennungsdatensatz als Vor-Trainings-Datensatz für Personenklassifikationsaufgaben, welche Trainings-Datensätze von nur wenigen tausenden Bildern haben, zu einer Genauigkeitssteigerung von über 8% führt, was in Folge zu einer Gesamtpräzision von über 99% führt. Die Qualität unseres Datensatzes wird weiters durch zusätzliche Evaluierung nachgewiesen. Darüber hinaus betonen die Ergebnisse die Komplexität der Auswahl einer geeigneten Architektur eines faltenden neuronalen Netzwerks und bezeugen die erhöhte Robustheit beim Training durch die Verwendung von Subklassenannotationen bezüglich Initialisierung und Lösungsalgorithmen.

Contents

CHAPTER 1

Introduction

It has been shown that Convolutional Neural Networks (CNN) are robust classifiers [LBD+89, KSH12]. With efficient, parallelizable GPU implementations CNNs have been applied to large datasets with up to one million images and have achieved state-of-the-art performance [KSH12, RDS+15]. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) was introduced in 2010 [DDS+09], and since 2012, CNNs have won the annual classification, localization and detection challenges [RDS+15]. The results of classification and localization challenges are shown in Figure 1.1. Error rates were reduced by 10% with the introduction of CNNs from 2011 to 2012 and by another 10% until 2014 to a 6.7% top-5 error by GoogLeNet [SLJ+15]. Furthermore, Zhou et al. show the significance of large, problem-specific datasets [ZLX+14].

Pedestrian or more general person classification is an inherent part of person detection, which is one of the challenges of surveillance and autonomous driving [AKV15, SA15, RHGS15]. Several person detection or classification datasets exist, such as the Caltech or Daimler datasets. However none achieve the proportions of the ILSVRC dataset as demonstrated in Table 5.1. The task of classification differs from detection in that an image is mapped completely to an image class, i.e. the existence of one object is assessed but no object extents or Regions of Interest (RoI) are computed, whereas detection aims to find bounds of multiple objects in one image. CNNs also achieve state-of-the-art performance in object detection, where their strong visual features aid in RoI classification [GDDM14, Gir15]. Additionally, high-level CNN features can be used for RoI proposal generation and in combination with classification thus perform state-of-the-art detection [RHGS15].

The aim of this work is to create *PersonData*, a dataset for binary person classification, i.e. classification of images as those of a person or not, of similar proportions to the ImageNet dataset. Additionally, a high-accuracy binary classifier is trained on this data. CNNs are chosen as the type of classifier because they achieve state-of-the-art performance as discussed above. We demonstrate that our data is advantageous and yields better

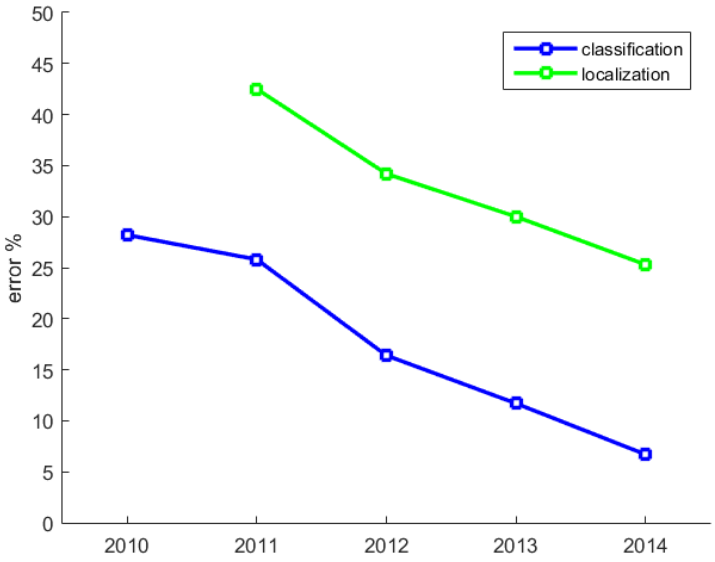


Figure 1.1: Winning error rates of the ILSVRC classification and localization challenges from 2010 to 2014. Data was taken from [RDS+15].

results for applications with small training data by first pre-training the model with our data. Moreover, we analyze and verify the benefit of additional labels to define the non-person class. We provide information about the data sources and propose a method of interpreting multiple datasets as a pool on which application-specific views and output datasets can be computed, allowing, for example, the combination and utilization of detection, segmentation and classification datasets for classification purposes.

Time and resources have limited this work to classification; it does not extend to detection. Furthermore, training is strictly supervised, and a single CNN architecture is used, which is only modified for different numbers of output classes and by adding Mean-Variance-Normalization (MVN) layers. Other deep methods, such as deep forests or deep belief networks, are left for future research.

Our work differs from state of the art in classification in that it is focused on person classification, whereas the state of the art was benchmarked on a 1,000-class challenge. GoogLeNet achieved a winning 6.7% top-5 error on the ILSVRC2014 [SLJ+15, RDS+15]. In comparison, we achieve an error rate of 2.82%. Note that this is an indirect comparison, since different training and test data for different classification tasks was used and ours is a binary error rate, i.e. comparable only to top-1 error for which no results are publicly available. So far no dataset of comparable size has been published for person classification. PersonData, therefore, constitutes a viable asset for training models for person recognition with over 300,000 person images. These images are gathered from more than 30 different datasets and thus cover a larger range of the person class's intraclass variability than any

of the data sources alone.

First, individual source datasets for person recognition were sought out and collected in one pool. This includes data for pedestrian detection, image classification and tracking. Datasets vary in their annotation format and structure and require separate handling for data extraction. Therefore, a two-step approach is used to first extract data from different sources and merge it into a collective, application-specific output. Datasets may be compressed or readable only with proprietary tools such as Matlab and have to be converted to readable formats beforehand. A CNN classifier is then trained using the extracted data for binary person classification. The CaffeNet [JSD+14] architecture is trained with stochastic gradient descent on the 300,000 person images and two types of images as the negative class. First, random negative crops from explicitly labeled negative images of the source datasets are explored. Afterwards, the 79 labeled nonperson object classes or their respective 11 categorical labels from the MicroSoft Common Objects in COntext (MSCOCO) dataset are utilized as a collective negative superclass.

We show that our classification network trained on PersonData achieves 97.18% accuracy and demonstrate its generalizability to new data by achieving over 94% accuracy on four extra datasets. Additionally, our results show that using additional labels for subclasses of the negative class yields 1.5% or 2% higher accuracy for 79 or 11 labels, respectively, indicating the benefits of subclass labeling. Furthermore, using PersonData as a pre-training set for pedestrian classification on the PETS dataset improves accuracy by over 8.37%.

Thus, the main contributions of our work are, first, the creation of a large-scale dataset for person recognition of over 600,000 images, second, the training of a binary person classifier with 97.18% accuracy and, third, the evaluation of subclass labeling of the catch-all negative class, showing that subclass labeling leads to increased robustness in initialization methods and solver choice and therefore higher accuracy.

The rest of this thesis is organized as follows. First, we explain the basic functionality of CNNs starting from a single perceptron and extending it to neural networks and finally CNNs. Next, a brief overview on person recognition with a focus on pedestrian detection is given. Afterwards, related work and the state of the art in the areas of person recognition, object detection and CNNs are discussed. Then, methodological details on the creation of PersonData and the training of our CNN classifier are explained. After this, implementation is discussed with regard to hardware, software and frameworks, smaller algorithms and training details. Next, the data and classifier are evaluated with regard to the number of labels, performance, overfitting and generalizability, and respective results are shown. Finally, the thesis is summarized, conclusions are drawn and strengths and weaknesses of the proposed methods and potential future work are discussed.

CHAPTER 2

Convolutional Neural Networks

CNNs have a long history starting with the perceptron algorithm in 1958 [Ros58]. Although CNNs can be explained intuitively as models that learn visual filters to recognize high-level image features, understanding the functionality of neural networks in a first step will facilitate understanding how CNNs work. In the following, the perceptron, its extension to the multi-layer perceptron, feed-forward neural networks and the basics of CNNs are explained.

2.1 The Perceptron

Rosenblatt designed the first learning neural computer, the *perceptron*, in 1958 in an effort to mimic human learning [Ros58]. A neuron's dendrites are modeled by weights, which are multiplied with input values. Additionally, a bias value is added to model a neuron's required activation potential. Afterwards, the multiplied values and bias are summed up in the cell body and passed through an activation function to produce an output representing the firing rate in the neuron. The architecture as described is shown in Figure 2.1.

The perceptron is a linear classifier defined by weights w_i, a bias b and an activation function f according to Equation 2.1. It is possible to merge the bias with the weights by using homogeneous coordinates, i.e. putting the bias at the bottom of the weight vector and adding a constant 1 to the input vector $\mathbf{x}$. Different activation functions, which are discussed in the next section, can be used, but the original perceptron uses the heavy-side step function, resulting in a binary output. Therefore, the perceptron is a linear classifier and its weights define a hyperplane as a linear decision boundary between classes. Thus, its representational power is limited; it is not possible to model, e.g., an XOR function using a perceptron since no line can be drawn to separate the classes as shown in Figure 2.2.

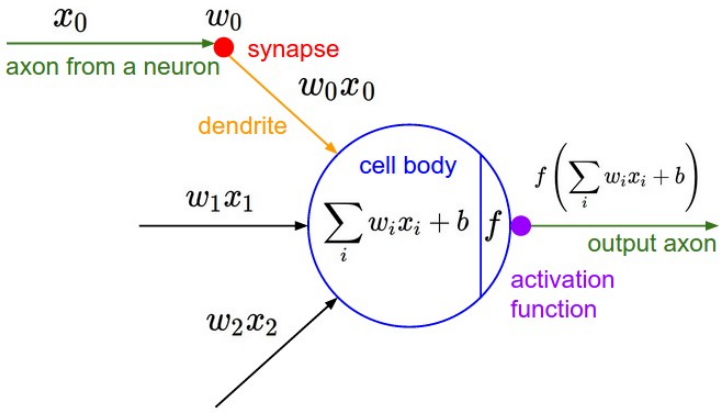


Figure 2.1: The perceptron model with weights w_i, inputs x_i and bias b. Image from [Kar16].

$$f\left(\sum_i w_i x_i + b\right) \tag{2.1}$$

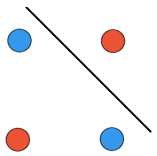


Figure 2.2: The XOR function creates a not linearly separable set. The two classes red and blue cannot be separated by a single straight line.

2.2 Neural Networks

To overcome the limited representational capabilities of a single perceptron, neural networks were developed. In a neural network, or artificial neural network, multiple perceptrons, also commonly referred to as units or neurons, are connected in acyclic graphs, although one of the most common architectures is that of the multi-layer perceptron in which the neurons are organized in layers. There are input and output layers and additional hidden layers that increase network size and complexity. Figure 2.3 shows an example of this layered architecture.

Multi-layer perceptrons are universal approximators, i.e. they can approximate any continuous function, and thus do not suffer the same limitations as a single perceptron [Cyb89].

Typically, the layers are fully connected, meaning that every neuron is connected to each neuron of the previous and next layer. However, neurons of the same layer are not

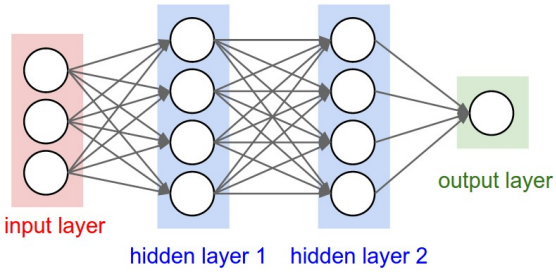


Figure 2.3: A multi-layer perceptron with two hidden layers. Image from [Kar16].

connected to each other. The number of units in each layer and the number of layers have to be chosen and result in networks that approximate functions of varying complexity. It can be difficult to find a good architecture, as using too many units and layers may result in a complex function that overfits training data, and choosing too few units can produce a biased function that is too simple to represent the data distribution well.

Historically, the sigmoid function has been used as activation function for the neurons, as it can be easily interpreted as firing rate of a neuron. The hyperbolic tangent function has a similar form but is zero-centered. Since 2010, the Rectified Linear Unit (ReLU) has become the de-facto standard, as it improves training speed and does not suffer from gradient vanishing [NH10, DSH13]. Output units do not have any activation function, so that they can produce arbitrarily valued output.

Neural networks can be trained, i.e. the network can learn weights, by defining a loss function (also called cost or objective function) and using the backpropagation algorithm to adapt weights [RHW88]. An example of a loss function is the sum-of-squared-distances. The backpropagation algorithm works by calculating the loss from the current network output and a ground truth and computing a weight update from it by passing the error backwards through the network [RHW88]. This is straightforward for the output layer, as the error can be used directly. For hidden layers, however, a sensitivity for the unit has to be calculated using the activation function's gradients. The sensitivity is then used to distribute the error backwards through the network.

2.3 Convolutional Neural Networks

Standard neural networks do not handle shifts and distortions in images, which occur frequently in image datasets, since objects are usually not perfectly aligned or appear multiple times in the same image. However, despite these drawbacks neural networks should still be able to learn robust features. Therefore, LeCun et al. developed CNNs [LBD+89], which are designed specifically for computer vision applications introducing shift and distortion invariance. A sample network is shown in Figure 2.4.

Using convolutional layers, the networks are able to learn to recognize local features, such as edges or corners, by restricting the receptive fields of hidden units to local connectivity

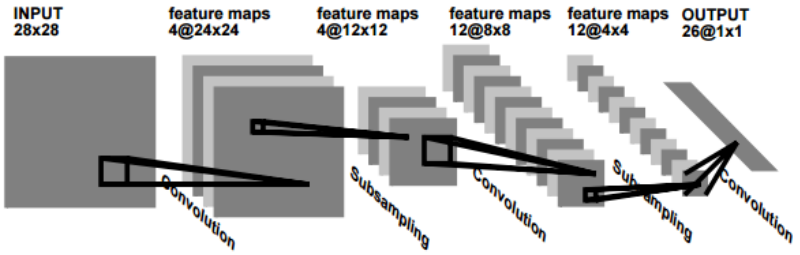


Figure 2.4: The Lenet CNN architecture with convolutional and pooling layers. Image from [LBD+89].

and to add shift invariance by enforcing spatially shared weights. Furthermore, spatial or temporal subsampling in the form of pooling layers reduces sensitivity to shifts and distortion.

The architecture of CNNs is made up of distinct layers that are usually arranged in multiple stages [LKF10]. The basic layers include convolutional layers, nonlinearities, pooling layers and fully connected layers. One stage may comprise a convolutional layer to learn filter banks a non-linearity identical to the activation function in standard neural networks and a feature-pooling layer. In the first level, such a stage typically learns simple visual features like edges or colour blobs. The second stage then combines the previous level's features, e.g. learning corners as combinations of edges. Adding more stages results in more complex high-level features, such as faces, depending on data and application.

Convolutional layers consist of multiple filters that are defined by their weights. The layer defines the number of filters and their kernel size, the stride in which they are applied and the amount of padding to handle image borders. Example filters learned in the first convolutional layer are shown in Figure 2.5. The convolved output of a filter is called a feature map and a convolutional layer with n filters creates n feature maps, which are the input for the next layer. For backpropagation, the gradient of the convolution is required, which is the forward-pass convolution with weights flipped along each axis.

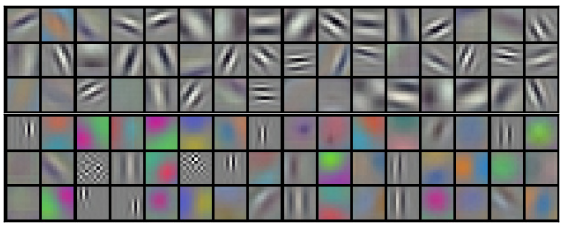


Figure 2.5: Filter weights learned by the first convolutional layer in AlexNet. Image from [KSH12].

Pooling layers reduce feature map resolutions and thereby the sensitivity to shift and

distortions, as exact feature location is discarded and only relative and approximate location information remains. Max-pooling selects the maximum output from a local receptive field and is applied in a sliding-window fashion similar to convolutions [GMB13]. Figure 2.6 shows the result of max-pooling. To obtain the gradient it is necessary to store the original location of the selected maximum value since maximum operations act like a routing mechanism in neural networks. An additional benefit of pooling layers is reduced memory cost. For example, 2×2 max-pooling results in output feature maps with half the input's width and height.

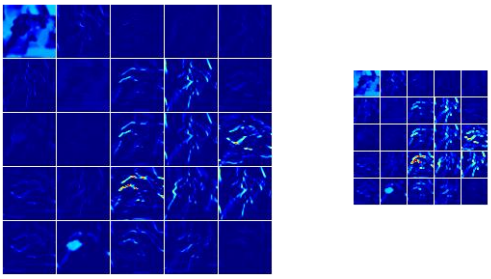


Figure 2.6: Max-pooling of 25 feature maps.

Fully convolutional layers work identically to hidden layers of a standard neural network. They can be used at the end of a CNN after several stages to compute arbitrary features and output scores (cf. universal approximators) [KSH12].

One method that is applicable to CNNs is transfer learning, where a network is pre-trained with a dataset and fine-tuned to another dataset afterwards [YCBL14]. In this case, a network is pre-trained ideally on a large dataset to learn robust filters and features that are generalizable to new data and then trained with application-specific data benefiting from features identical or similar to the pre-trained ones.

Since 2010 CNNs have been applied to various computer vision tasks and have been developed further [RDS+15]. Various architectures have been published and open-source frameworks have been developed, creating large user bases [RHGS15, JSD+14]. Furthermore, several important robustness-increasing techniques have been developed, such as dropout and batch normalization, which allow higher learning rates for faster training while minimizing the degree of overfitting [SHK+14, IS15]. These and other improvements are discussed in the related work.

2.4 Chapter Summary

In this chapter the history of CNNs is discussed. The perceptron is introduced as a model of human neurons with its activation function output resembling the rate of fire. Next, multi-layer perceptron neural networks are explained as an extension and the backpropagation algorithm to train them is discussed. Afterwards, CNNs are explained,

covering the types of layers used and features that are learned. Lastly, pre-training of CNNs and techniques such as dropout are discussed.

CHAPTER 3

Person Recognition

Person recognition is a challenging task in computer vision with applications such as surveillance, autonomous driving, robotics and virtual reality [ZC12, MG06]. The task of person recognition comprises classification, detection and segmentation problems and is a special case of the more general object recognition. The person class is specifically difficult to model, since pedestrians or faces exhibit high intraclass variability, because of variations in, for example, pose, clothing and background [OPS+97, MG06]. The main difference between person detection and person classification is that in detection only the person class is looked for and people have to be differentiated from the rest of the world, whereas in classification the class is one of a select pool of fixed classes [OPS+97]. Person detection has a long history; In 1985, for example, Tsukiyama et al. detect people in image sequences [TS85]. In the following, an overview on types of pedestrian detection methods will be given, including the state of the art.

3.1 Pedestrian Detection

The methods of pedestrian detection can be split roughly into *static* and *dynamic* methods. Static methods are applied on still images or single frames, whereas dynamic methods require video sequences for motion information [ZC12]. Another type of method is the use of combinations of models and features to produce a more robust detector. Finally, there are part-based methods, which aim to handle partial occlusions. These method types are explained in the following.

3.1.1 Static Pedestrian Detection

Static methods detect people on a single image by either searching the entire image with a model or by calculating features, which are then classified. Model-based methods, therefore, require a robust pedestrian model able to deal with problems, such as pose

variation and occlusion. Examples of model-based methods include histogram of human appearance [CRM03], where localization is formulated using the basin of attraction of a smoothed similarity function, template-trees [Gav07], where a variety of shape exemplars is matched using trees that are clustered with stochastic optimization, and statistical field models [WY06], which capture local nonrigidity.

Most pedestrian detection methods use feature extraction and classifiers on regions of interest [ZC12]. A detection window is selected from the entire image, often by the use of sliding window. Features are computed for this window, which are then fed into classifiers. The classifiers are trained on sample windows labeled as person and nonperson. An example of a classifier is support vector machines.

Histogram of Oriented Gradients (HOG) [DT05] is one such feature commonly used for feature-based detection [ZC12]. The detection window is tiled into overlapping blocks. In these blocks, computed gradients are aggregated into spatial and orientation cells and are additionally contrast-normalized. Another common feature is the Scale Invariant Feature Transform (SIFT), since it is robust and scale invariant [Low04]. SIFT computes high-dimensional vectors representing image gradients similar to HOG. It can be combined with principle component analysis to reduce computational costs [ZC12]. Furthermore, other important features include Local Binary Patterns (LBP) [MYL+08], where binary codes are computed by thresholding local information, and shape context [YLL10], where points inside the shape and on its boundary are used to measure shape similarity.

3.1.2 Dynamic Pedestrian Detection

Assuming that the camera has a fixed position and orientation, i.e. there is no relative motion between the camera and the background, motion features are a strong indicator for pedestrian movement [ZC12]. Thus, multiple dynamic methods have been developed based on video data. For example, Histogram of Oriented Flow (HOF) combines motion-based descriptors with HOG appearance descriptors to detect moving and standing people [DTS06].

3.1.3 Combined Methods

The combination of complementary features can improve detection performance [ZC12]. Walk et al. combine HOG with a self-similarity feature and motion features derived from optical flow to construct a robust detector, showing that feature combinations handle detection challenges better than single features [WMSS10].

3.1.4 Part-Based Pedestrian Detection

A main challenge in pedestrian detection are occlusions, since pedestrians in, for example, street scenes occlude each other or are occluded by cars, street signs, animals, or other objects [WNL08]. The idea of part-based detection is to separate the object in question into parts, which are detected separately. The separate parts can then be combined using

known geometrical constraints, such as that one part has to be connected to another specific part in a certain angle. In the case of pedestrians, arms, legs, torsos and heads can be detected separately to robustly handle partial occlusions. Wu et al. use 11 hierarchically clustered body parts in a joint likelihood framework to detect occluded pedestrians [WNL08].

3.2 State of the Art

Before CNN-based region proposals, the state of the art in pedestrian detection was based on manually designed features, such as the filtered channel features introduced by Zhang et al. [ZBS15]. The method of Zhang et al. involves computing feature channels from an input image, calculating feature vectors by sum-pooling over rectangular regions and feeding them into a decision forest. The sum-pooling can be seen as a convolution with a rectangular filter bank, thus defining visual features. The decision forest consists of short trees, i.e. weak classifiers, and is trained using Adaboost. However, CNNs have improved object detection since 2015 [RHGS15]. CNN-based detection is static, feature-based detection, which benefits from high-level and robust visual features [KSH12]. A breakthrough is the work of Ren et al. [RHGS15], in which the robustness of CNNs is additionally exploited for region proposal generation. The method is discussed in the next chapter along with other related work.

3.3 Chapter Summary

This chapter gives an overview on person detection. The challenges of person recognition are explained. Then, four types of methods for pedestrian detection are discussed, including static, dynamic, combined and part-based methods. Lastly, the state of the art in person detection is explained.

CHAPTER 4

Related Work

CNNs are models capable of being trained on large amounts of data and benefit from strong robustness in computer vision tasks [KSH12, RDS+15]. The power of CNNs in image classification was shown already in 1989 when LeCun et al. classified handwritten digits for zip code recognition with 5% test error [LBD+89]. In the following, techniques for increasing robustness, examples of CNN architectures, detection capabilities and other related work on CNNs are presented.

Dahl et al. show that using ReLUs and dropout improve performance of deep neural networks [DSH13]. Dropout is a method that reduces overfitting by randomly dropping units and their connections during training, thus preventing correlation between units [SHK+14]. Units are dropped with a probability p, where p can be different for every layer. Krizhevsky et al. suggest dropout rates of less than 20% for input layers and of about 50% in higher layers. At test time, no units are dropped, but their outputs are multiplied with their respective dropout probabilities, since not dropping any units else results in a proportionately higher expected value. This leads to a heuristic exponential model averaging, resulting in less overfitting. Also, when fine-tuning a network using pre-trained weights from a network trained with dropout, Krizhevsky et al. suggest pre-multiplying the weights with their dropout probabilities for the same reason.

In 2015, Ioffe et al. introduce batch normalization, which is another technique of making neural networks more robust [IS15]. During training with stochastic gradient descent, minibatches are normalized regarding their mean and variance, avoiding the problem that neurons receive input of varying distributions at every pass and have to adapt their weights accordingly. A moving average and variance are calculated and final values are used at test time. Training is, therefore, less sensitive to initialization and learning rates, which can thus be set to higher values than without batch normalization, leading to five times faster training speed while still increasing accuracy by 2% [IS15].

Data augmentation is a model-agnostic method for learning invariances. Paulin et al. state the importance of learning class invariances and proposed an explorative algorithm to find the best combination of data transformations [PRH+14]. Training data is transformed with the transformations expected to happen at test time. These transformations reflect extrinsic variation, such as lighting conditions and point of view. Flipping and random cropping were shown to be data augmentation methods that increase robustness [PRH+14]. The augmentation can also be applied at test time. Paulin et al. show that there are, however, also transformations that do not increase or can possibly even decrease accuracy while increasing computational load [PRH+14].

Another method of enhancing performance in CNNs is to use recurrent convolutional layers as proposed by Liang et al. [LH15]. Recurrent convolutional layers are based on recurrent connections in the human visual system and not only use the output from the previous layer but also cycle their own output into themselves. In practice this is done by unfolding the recurrent layers for a fixed amount of time-steps. Unfolding does not introduce new parameters to the network, as the same connections are used multiple times for recycling output. Thus, the network can be made deeper by unfolding, but the number of parameters remains constant. Additionally, multiple paths are created, since the input is passed to each unfolded iteration. This results in larger receptive fields, where the longest path results in the largest receptive field. Hence, multiple levels of local context are included in one recurrent layer, i.e. one for each path depth, leading to higher accuracies compared to a similarly deep cascade of normal convolutional layers with the same amount of parameters.

The first CNN applied to the ILSVRC-2010 challenge is AlexNet, with a top-1 error rate of 37.5% on the 1,000-class classification challenge [KSH12]. An efficient GPU implementation and the large ImageNet dataset are used to train a neural network consisting of five convolutional layers interleaved with max-pooling layers and three fully connected layers on top. Furthermore, Krizhevsky et al. note the importance of dropout, data augmentation and network size. Eigen et al. introduce the OverFeat architecture for classification, localization and detection challenges of the ILSVRC-2013 challenge, winning the localization challenge [SEZ+13]. The model uses the same layers as AlexNet but discards local contrast normalization and varies several hyperparameters such as filter sizes and strides. Eigen et al. also show that using more layers and creating deeper networks results in better performance. Szegedy et al. achieve state of the art on ILSVRC-2014 with a 6.7% training error by developing a 22-layer deep model built from multiple so-called inception modules that comprise several parallel paths through combinations of convolutional and pooling layers [SLJ+15]. Szegedy et al. explicitly use 1×1 convolutions to increase depth with little computational costs and model the architecture to approximate optimal local sparse structures of CNNs with dense components. Simonyan et al. extend on the idea that deeper architectures improve accuracy and proposed the VGG architecture [SZ14]. Their model contains up to 19 weight layers, 6 max-pooling layers and three fully connected layers using only 3×3 convolutions in the convolutional layers. Simonyan et al. show that simply increasing

the network depth results also increases accuracy and achieved first place in localization and second place in classification in the ILSVRC-2014 challenge. Other improvements to CNN architecture have been proposed for various tasks such as the deformation pooling layer [OWZ+15], which learns deformation penalties for object detection, or sibling convolutional layers with shared weights, which learn person features passed together into a layer that learns a similarity measure for person re-identification [AJM15].

Fully convolutional networks are introduced by Long et al. [LSD15] for image segmentation and by Springenberg et al. [SDBR14] for image classification. Fully convolutional networks do not contain fully connected layers, since fully connected layers require fixed size input but convolutional layers allow arbitrary input dimensions due to their sliding window application. Additionally, Long et al. propose a method for converting fully connected layers to convolutional layers by interpreting them as convolutions with kernels that span the entire input region [LSD15]. Thus, models trained for classification with fully connected layers can be refactored to models that output classification heat maps on arbitrary images.

CNNs trained for classification are shown to implicitly learn localization information even for image-level labeling [OBLS15]. However, only approximate object location is learned but not the extents, as our own experiments also demonstrate. Therefore, using CNNs built specifically for object detection leads to better detection performance than using their classification counterparts. OverFeat is originally applied to object detection by using classification and regression loss in a sliding-window detector [SEZ+13] and is further extended with a recurrent LSTM layer for more robustness in crowded scenes [SA15]. Sermanet et al. train a CNN for pedestrian detection using multistage features, i.e. the network contains connections or paths from low-level and mid-level layers to layers of the classification stage, improving accuracy, with unsupervised training by sparse coding [SKCL13]. As CNNs display low speed even with GPUs, Angelova et al. propose large-field-of-view deep networks for pedestrian detection [AKV15]. The network is trained to find multiple detections at the same time to save time. More precisely, in a first stage, the image is classified in a 4 by 4 grid by a four-layer, then each grid element is classified with a deeper network and, finally, a third, even deeper network is used as a last and accurate check. Liu et al. introduce spatially weighted max-pooling to model spatial relation of semantic features in pedestrian detection [LHYS15]. CNNs using their method are, for example, able to learn optimal positions for heads.

Exploiting the speed and robustness of low-level feature region proposals such as Selective Search, Girshick et al. propose the Region Convolutional Neural Network (RCNN) detection pipeline, achieving 53.3% mean Average Precision (mAP) on the VOC 2012 challenge [GDDM14]. The system first extracts about 2,000 region proposals from the input image calculated with Selective Search. Next, features are extracted from a deep CNN such as AlexNet. Lastly, the image patches are classified by class-specific Support Vector Machines (SVM) using the CNN features. In 2015, He et al. propose Spatial Pyramid Pooling (SPP) for CNNs, where fixed size representations are computed from arbitrarily sized feature maps, thus creating the necessary fixed-length input for fully

connected layers [HZRS14]. SPP is robust to deformations and has the advantage for object detection that convolutional features only have to be computed once per image as compared to RCNN. The convolutional features can then be pooled into fixed length and used in detectors, resulting in speeds over 20 times faster than with RCNN. Lenc et al. combine RCNN and SPP into a detection network by integrating SPP as a layer and adding bounding box regression [LV15a]. Additionally, they speed up training by using a constant region proposal scheme, only training images on one scale as compared to multiple scales in the original SPP method, and by replacing SVM classification with a softmax output. The constant region proposals are pre-calculated by statistical analysis of the underlying PASCAL VOC 2007 dataset. They show that replacing the SVM and only evaluating one scale results in mAP drops of only 1.3% and 1.1%, respectively, but speeds up training significantly.

Coincidentally, Girshick propose a different improvement on RCNN, resulting in a method ten times faster than SPP [Gir15]. Using RoI pooling, fixed size vectors can be computed from arbitrary regions similarly to SPP. In fact, RoI pooling is the special case of SPP pooling with a one-level pyramid. A region is divided into a fixed number of grid elements, and inside each element, values are max-pooled to an output. RoI pooling is applied on top of convolutional features using image-level region proposals as computed by Selective Search or EdgeBoxes. The pooled regions are passed through fully connected layers to create a feature vector. This vector is then passed into sibling fully connected layers, leading to a softmax layer for classification scores and a bounding box regressor, respectively.

Ren et al. further improve on this design by introducing Region Proposal Networks (RPN) [RHGS15]. Exposing region proposals as the bottleneck of the RCNN pipeline, RPNs are designed to compute robust region proposals fast. The RPN is used for region proposals that are then used in the detection network. To reduce costs, both networks share their convolutional layers for features. On top of these features, the RPN now uses a sliding window, which is natively implementable as a convolutional layer, to create an intermediate output, which is then fed into two convolutional sibling layers such that the whole network is fully convolutional for arbitrary image sizes. These sibling layers calculate objectness scores and bounding box coordinates, respectively. Additionally, both are calculated for nine anchors in the input image simultaneously, since receptive fields can span almost the whole input image. The anchors are defined by three aspect ratios and sizes to cover nine possible region proposal shapes. The objectness score is then used to find relevant region proposals used in the detection network. This procedure only requires evaluating the shared convolutional layers once and then the different top parts of the RPN and detection network, resulting in speeds of, for example, 5 fps for the deep VGG network. Additionally, fewer region proposals are needed than before, i.e. 300 instead of 2,000, to still achieve a higher 70.4% mAP on the PASCAL VOC 2012 challenge. However, a drawback of the method is training, as it requires four steps. First, the RPN is trained by fine-tuning a model that is pre-trained with ImageNet. Then the region proposals from the first phase are used to train the detection network, which is

initialized equally. Next, the convolutional layers are initialized with the weights from the detection network and only the unshared RPN layers are trained. This fixes the shared weights between both networks. Lastly, the top layers of the detection network are fine-tuned using the region proposals from the third step.

CNNs benefit from training datasets of multiple hundred thousands of images to achieve high performance and are commonly evaluated on benchmark challenges providing such large datasets [RDS+15, LMB+14]. Caltech101 is a classification dataset from 2005 for 101 object classes [FFFP07]. The Caltech Pedestrian Detection Benchmark contains a dataset with 350,000 person annotations in 250,000 frames and has been used for the evaluation of over 15 detection approaches [DWSP12]. The PASCAL Visual Object Classes (VOC) Challenge, which has been held annually from 2005 to 2012, comprises classification, detection and segmentation challenges on 20-class data with over 31,000 annotations in over 11,000 images and additional challenges such as action segmentation or person layout.

In 2010, the ILSVRC for image classification and object detection has been introduced and is being held annually [DDS+09, RDS+15]. With the use of Amazon Mechanical Turk to achieve large-scale annotation, approximately 1.2 million images are organized according to the WordNet hierarchy. The individual challenges are 1,000-class image classification, 1,000-class single-object localization, where the bounding box of one object per instance per image has to be calculated, and 200-class object detection. Only not occluded object parts are used for bounding box annotation. Classification accuracies are measured as top-1 and top-5 error rate, i.e. the percentage of images where the ground truth was not among the top 1 or 5 output classes, respectively, sorted by classification confidence. The winning score of the 2014 classification challenge is GoogLeNet with a 6.7 % top-5 error.

The MSCOCO dataset is published in 2015 [LMB+14]. It contains 2.5 million annotations in 328,000 images. The images are annotated with instance-level polygonal segmentation masks. In total, there are 91 object classes; however only 82 of them have more than 5,000 labeled instances. The 2014 release contains only 80 classes due to incomplete or ambiguous labeling. The labels are grouped into 12 supercategories, where the person class is its own category. Contrary to Imagenet, fewer classes are provided; however, there are more images per class. Additionally, the amount of images per class is equally distributed.

To address the lack of understanding of how CNNs work or why they perform so well, Zeiler et al. introduce deconvolution networks. These networks are able to visualize image parts and features that lead to activations for each neuron [ZF14]. Deconvolution networks invert the operations of the network by deconvolving, using the transposed filters and unpooling with switch variables for input locations. These visualizations help to understand which neuron learns which features. Nguyen et al. [NYC15] use evolutionary algorithms to deliberately fabricate incorrect images that still achieve a classification score of 100%. They find that deep networks do not learn global structure, since feature repetitions are allowed and the best way to handle this problem is to use

larger datasets with more classes such as in the ILSVRC, thus covering more variability. Mahendran et al. invert image representations including CNN features and found that deeper features become more invariant and abstract [MV15]. Lenc et al. analyze the equivariance, invariance and equivalence of CNN features by using stitching layers, which are able to swap network parts [LV15b]. Low-level features are found to transform along with the input image and are equivalent among different network architectures, while deeper layers lose these properties. In 2015, Yosinski et al. develop a tool for visualizing intermediate activations of CNNs using a video feed, which allows activations to be analyzed using dynamic, moving data [YCN+15]. Additionally, they propose a regularized optimization technique to create recognizable and interpretable images that lead to high activations for each neuron.

Since ground truth data may contain labeling noise, Vo et al. propose a training method that is able to deal with this problem by using weakly supervised re-ranking and network fine-tuning [VGLBP15]. They use this algorithm to train a CNN on noisy web data, and their results suggest that given more research, large quantities of data that have not been controlled manually can be used.

CNNs have also been applied to other tasks such as artistic style transfer [GEB15]. Gatys et al. use feature map activations for content reconstruction and feature map correlation, defined as a matrix of feature map products, for style reconstruction. Higher layers contain less information about content but more about style. They propose style transfer by creating an output image via backpropagation using a combined content and style similarity loss function.

4.1 Chapter Summary

In this chapter related work on CNNs is introduced. The techniques of dropout, batch-normalization, data augmentation and recurrent convolutional layers for increased robustness are explained. Then, common architectures such as AlexNet are discussed and fully convolutional networks are explained. Afterwards, CNNs for detection from weak localization to specialized RPNs for high-level RoI proposals are discussed. Lastly, several benchmarks, methods for visualizing and understanding CNNs and for dealing with labeling noise are introduced.

CHAPTER 5

Methodology

Deep learning typically benefits from large amounts of data, e.g. datasets from hundred thousands to millions of images, as can be seen from the results of the annual ILSVRC challenge, where CNN detectors and classifiers achieve state-of-the-art performance on a 200-class detection and a 1,000-class classification task with a dataset of approximately 1.2 million training images [RDS+15]. Therefore, we create a dataset pool designed specifically for the task of person classification. In the following, details on the data sources and merging are discussed. Then, the training of a CNN on this data is explained and, lastly, localization based on feature maps is examined.

5.1 Dataset

When working with the person image class, its high intraclass variability has to be taken into account. Additionally to lighting conditions, occlusions and other extrinsic variability, people vary further due to strong differences in pose, clothing and accessories, skin color and more. Therefore, for the task of binary person classification, i.e. classifying an input image as person or not person, a large dataset of hundred thousands annotations for persons and a catch-all negative class is required. Many datasets with images of persons are available; however, they were designed specifically for certain tasks and publications, such as tracking, detection, pose estimation, action recognition and others. Furthermore, these datasets, although varying in size, are comparably small, containing only a few hundred to a few thousand images. Datasets such as the ILSVRC and MSCOCO datasets, which are commonly used for deep learning, consist of multiple hundred thousand images.

We therefore collect annotated person images from 31 different datasets, creating a large-scale dataset specifically for person classification, which is crucial to training a deep CNN with high accuracy [RDS+15]. The dataset, *PersonData*, comprises about 600,000 images and is explained in more detail in the following.

5.1.1 Dataset Pool

In order to create the PersonData, over 50 other datasets were examined. The foremost problem is the definition of what a person image is. In the case of PersonData, the person class is defined implicitly from all collected data. Since there is too much data to analyze manually or to prune source images not fitting a manually selected definition, we decide to heuristically describe this implicit definition. For the sake of simplicity, a person image is defined as an image that contains at least part of the human body. Note that although this means that a picture of a finger counts as a person image, the tendency in all datasets is full-body pedestrian images. Examples of different types of person images are shown in Figure 5.1. Some images may be labelled as a person although the focus seems to be on another object as, for example, in Figure 5.1b. Apart from perspective and pose, different datasets also exhibit differences in the choice of margin around the person, color space and image sizes as exemplified in Figure 5.2.

(a) Pedestrian images

(b) General person images

Figure 5.1: Images labeled as person. Most datasets provide pedestrian images such as (a) while other datasets contain general person images such as (b).

Due to their variety in applications and sources, the collected datasets are inherently different. Their folder structures, image and annotation formats can be vastly different, resulting in difficult combined usage. To address this issue, we propose a method of extracting data in a standardized manner from different sources by implementing individual dataset parsers that return standardized information. This requires almost no modification of source datasets and allows for easy integration of new sources as well as simple addition of new queries. The only alterations to source datasets are to decompress

Figure 5.2: Examples of image differences among datasets. (a) shows a monochrome and an RGB, (b) different margins and (c) different scales and crops.

them, split videos into separate image frames and convert binary Matlab files to plain text. The individual parsers handle different folder structures, file naming conventions, and annotation formats, such as XML-based or text file annotations and annotation-to-image-file correspondence, since one annotation file can include information for only one image or even an entire dataset. Another script uses the returned standardized information for a specific task, which in our case is image classification. Image classification requires image-level labeling, which means that images containing multiple annotated persons have to be cropped to each annotated bounding box. Therefore, bounding box coordinates and class labels are returned from the parsers and are used to crop and store labeled images in another folder holding the complete, parsed output dataset. Note that bounding boxes can be calculated on the fly from segmentation maps or polygonal annotations, which facilitates the use of segmentation datasets for classification. The results of this procedure can be interpreted as a view of the base dataset pool.

In the use of tracking datasets for classification, image redundancy has to be considered. Video data may show a person walking on a street. Frame by frame annotation results in a sequence of images of the same person in a walking cycle. Since the person's pose repeats itself and the background stays relatively similar, e.g. a sidewalk, this means that individual images are similar. The fact that these multiple, similar images can outnumber single, nonrepetitive still image of from a different dataset indicates a potential of overfitting to video data. For PersonData information is, therefore, only extracted

from every 100^{th} frame. This constant is selected in an attempt to not include individuals more than four times and thereby including only four distinct poses in walking cycles, which were empirically found to be different enough to hold new information. Figure 5.3 depicts the small changes in pose variation of a person walking.

Figure 5.3: Images extracted from a video of a person walking exhibit little pose variation and, therefore, lead to redundancies.

To further improve the general quality of the dataset, person annotations smaller than 40×60 pixels are discarded, as these were empirically found to not be recognizable. As a comparison, evaluation on a subset of the Caltech dataset shows that its bounding boxes range from 32×32 pixels to 243×480 pixels with an average of 54×130 pixels. Furthermore, data augmentations such as flipped images are discarded, since this is done on the fly in the training phase. Complete datasets are discarded, if they lack annotations, are partially or completely included in other datasets or if the amount of data contained (cf. video redundancy) is not worth the effort of implementing a possibly complicated parsing algorithm. For example, this is the case with the Grand Central Station or Mall datasets, which only include trajectory annotations without bounding boxes.

The result is a pool of 31 selected datasets yielding over 300,000 labeled person images. The list of source datasets is shown in Table 5.1. The datasets can be more or less suitable depending on their size. For example, MSCOCO provides roughly 130,000 person images, and the Daimler datasets sum up to approximately 100,000 person images. This means that these two datasets provide one third of the total data each, and the remaining 29 make up the final third. Note that this is only the positive class of the binary problem. In the first experiments, the negative, i.e. nonperson, data available from these datasets is also used. Negative data is also diverse, ranging from explicit negative cropped images to empty scene images such as streets, and some datasets do not provide any nonperson data at all. Random image crops are computed from this negative data pool by calculating random bounding box sizes and positions, given a few constraints such as the minimum bounding box dimensions and an aspect ratio between 0.7 and 1.3. The amount of negative images thus created is set to the amount of positive images extracted

to create a 50:50 distribution. It is suggested to mimic the application's distribution in the training data to minimize bias toward any class. Since our application is simply classification itself, we settle on an equal distribution. As we have little control over the random negative crops' contents and only a subset of datasets contains labeled negative images, we additionally choose to augment the MSCOCO dataset without extracted person images as a different base of comparison. MSCOCO provides equally distributed segmentation and bounding box labels for 80 classes divided into 12 categories within over 300,000 images. The image distribution over the MSCOCO dataset is shown in Figure 5.4. Augmenting the person class with our collected data yields 343,814 images, while the remaining 79 classes total to 268,770 images, resulting in a 56:44 distribution. The 79 classes are used in combination as the negative nonperson class. This has the benefit of deeper labeling than random negative crops, since class and category labels are provided. Additionally, this allows for evaluation of which objects in the non-person class can be classified well and which not.

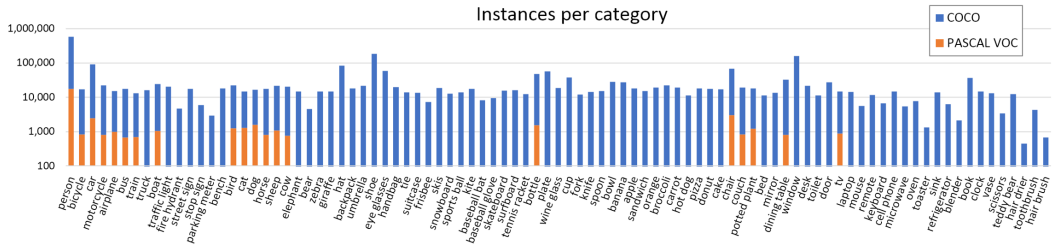


Figure 5.4: Distribution of images over the MSCOCO dataset in comparison to the PASCAL VOC dataset. Image from [LMB+14].

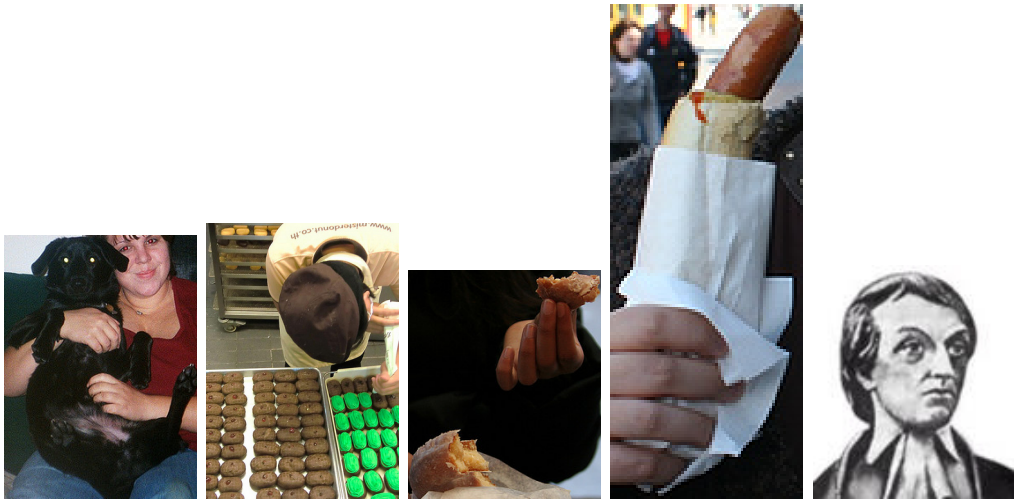
A problem inherent in large datasets is *labeling noise*, since the sheer amount of images makes such datasets difficult to examine manually. E.g., when calculating only 2 seconds per image and a work schedule of 38 hours, it would take over a month to look at 300,000 images. Labeling noise refers to incorrect or imperfect annotations, for example, a picture of a dog with the label person. During training, however, ground truth annotations are assumed to be correct and labeling noise, therefore, negatively affects the model. Reasons for labeling noise are automatic methods, where results are only sparsely checked, but also differences in perception and labeling type and effort. In the LabelMe dataset, for example, people can be tagged with synonymous tags and may occur under labels such as *person, pedestrian, boy, child, father*, etc. Furthermore, segmentation to bounding box conversion is problematic because the image contains multiple objects, as the following example demonstrates. An image of a person's lap with a pet on it, where the person's legs are labeled results in a converted bounding box labeled as person containing the whole pet but only a part of the human. Examples of inclusion and perception problems are depicted in Figure 5.5.

Dataset	# ext.img.
MSCOCO	130,654
Daimler Mono Ped. Detection Benchmark Dataset	16,071
Daimler Multi-Cue Occluded Ped. Classification	88,880
Daimler Ped. Segmentation Benchmark (PedCut)	505
2D MOT 2015	548
Caltech Pedestrian Dataset	232
CAVIAR Data Set 2	270
CBCL Pedestrian Database #1 (Pedestrian128x64)	924
CBCL StreetScenes	1,373
Crowdsourced Annotation Dataset	100
CVC-02 Pedestrian Dataset (Classification)	1,083
CVC-03 Virtual-Pedestrian Dataset	839
CVC-04 Virtual-World Pedestrian Dataset 2	1,208
CVC-05 Partially Occluded Pedestrian Dataset	1,357
CVC-06 Partially Occluded Virtual-World Pedestrian Dataset	1,339
CVC-07 DPM Virtual-World Pedestrian Dataset	2,534
ImageNet n02472293 & n00007846 (Man & Person Synsets)	526
INRIA Person Dataset	1,696
KITTI Object Detection Eval 2012	2,057
KITTI Object Tracking Eval 2012	110
LabelMe	11,642
mpi-inf (Training & Test)	10
mpi-inf2 (TUD Multiview Pedestrians, TUD Stadtmitte)	673
Multi-view Multi-class Detection dataset	129
NICTA Pedestrian Dataset	44,223
PASCAL VOC 2012	13,540
Pedestrian Attribute Recognition At Far Distanec (PETA)	16,587
PNNL Parking Lot 1 & 2	100
PNNL Pizza	184
PRID 2011 Dataset	459
Pedestrian Parsing in Surveillance Scenes Dataset (PPSS)	3,961

Table 5.1: List of source datasets used for the PersonData dataset additionally showing the number of extracted person images per dataset.

5.2 Person Classification

PersonData is created as explained in the previous section for the main task of person classification with as much data as possible. A deep CNN is trained to classify an image as containing a person or not. The count of persons in the image is ignored and only the existence of a person is of importance under the assumption that input images only



(a) (b)

Figure 5.5: Examples of labeling noise issues. (a) shows problems due to inclusions, where people appear in object images or vice-versa. (e) shows a drawing labeled as person.

contain one person. This means that a picture containing multiple people, e.g. due to overlaps, is also simply classified as person. Contrary to object detection, localization is not part of the problem. Therefore, no RoI proposal generation or bounding box calculation or refinement has to be done. It is assumed that the images used in training already are potential RoI and further only have to be classified. The trained deep network can be extended naively to a detector with a sliding-window routine. However, we focus solely on creating classifier and analyzing its potential in relation to the data provided. The network was trained with a few variations on architecture or input data to demonstrate robustness and analyze the benefit of our dataset and the choice of catch-all negative class labeling. First, the choice of the neural model is explained. Afterwards, the training of the network is described.

5.2.1 Network Architecture

Deep CNNs are made up of different types of layers. These layers include convolutional layers, pooling layers, fully connected layers and others. All of these layers may have additional hyperparameters such as filter size, padding, stride for the convolutional layers and the number of neurons for the fully connected layers. Note that parameters of the network architecture and training procedure are called hyperparameters to distinguish them from the weights and biases learned by the network during training, which are collectively called parameters. The amount of layer types along with their additional hyperparameters and their effect on training speed and quality make it difficult to choose an architecture. State of the art therefore usually relies on or modifies existing published architectures such as AlexNet [KSH12], GoogLeNet [SLJ+15] or VGG [SZ14], since they

perform well on benchmarks such as the ILSVRC.

We use CaffeNet [JSD+14], because the hardware has limitations: It is not capable of fitting very deep networks such as VGG into memory. CaffeNet is a modification of AlexNet citeAlexNet without relighting data augmentation and where the order of pooling and normalization layers is switched. An outline of the CaffeNet architecture is shown in Figure 5.6. CaffeNet is made up of five convolutional layers, three fully connected layers and a softmax output layer. It uses ReLUs as activation functions and employs dropout in the first two fully connected layers. Additionally, pooling layers are used after the first, second and fifth convolutional layers; furthermore, Local Response Normalization (LRN) layers are used after the first two pooling layers. CaffeNet has the common architecture of convolutional layers followed by fully connected layers without SPP or RoI pooling and therefore accepts only fixed size input. The output softmax function interprets the data as a probability distribution and output is, therefore, in the range of 0 to 1, summing up to a total of 1. In the binary case, this leads to results such as $(1, 0)$, $(0.2, 0.8)$ or $(0.5, 0.5)$. The numbers represent a confidence of classifying the image as a certain class, and the highest number is chosen as the output class. The higher the confidence, the safer the classification is presumed to be. In the worst case, multiple classes share the same or a similar confidence value and are thus ambiguous. It is possible to only consider classifications with a confidence over a certain threshold, e.g. 0.6. However, we do not use any threshold.

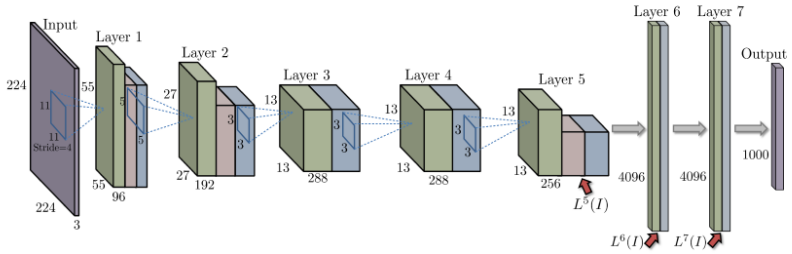


Figure 5.6: The CaffeNet architecture. Layers 1 to 5 are the convolutional layers combined with pooling and LRN and layers 6 and 7 are the fully connected layers. Image from [BSCL14].

5.2.2 Training

Various solvers can be used to train neural networks. Stochastic gradient descent, Nesterov's accelerated gradient descent and Adagrad are, for example, three available solvers, all of which are used in combination with backpropagation to modify and learn network parameters by minimizing an objective function. In stochastic gradient descent, training data is passed through the network in small partitions, called minibatches. This has the advantage that parameters are updated multiple times in one epoch, i.e. one iteration of the entire dataset, and that fewer data has to be held in memory for a training step, which is especially useful for GPU implementations. A minibatch is passed through

the network, computing intermediate outputs along the way as well as a final output, which in our case is the softmax function. This output is compared to the ground truth by means of a loss function such as the Euclidian loss or multinomial logistic loss, resulting in an error measure. Since the error is computed from the last layer, it is directly usable for this layer. For earlier layers, however, the error has to be propagated backwards through the entire network by means of backpropagation using gradient sensitivities. Thus the parameters of every neuron can be modified using individual propagated errors. This is done by multiplying the error with a learning rate and subtracting it from the current value. To achieve convergence, the learning rate is adapted during training to become smaller, leading to less modification of the parameters and thus converging values. An extension of standard stochastic gradient descent is standard stochastic gradient descent with momentum. Momentum calculates a new update as a convex combination of the previous update and the current gradient. This is interpretable as a momentum of the current model state moving on top of the loss function and helps in breaking out of shallow ravines, i.e. local optima.

It is important to note that this procedure converges to a local optimum of the high-dimensional loss function depending on the entirety of learnable parameters; therefore, an adequate initialization is necessary. The standard that is used per default in the Caffe framework is initializing neuron weights with random values from a Gaussian distribution and setting the biases to zero. Sample random weights are shown for the first convolutional layer in Figure 5.7. Random values are used for what is called symmetry breaking, as they lead to random initial activations at each neuron, while using the same values would lead to equal activations and errors and thus redundancies [KSH12].

Figure 5.7: Random initial values for weights of the first convolutional layer are sampled from a Gaussian distribution.

A second method of initializing parameters is to use a pre-trained model. A network is trained with a large dataset with the goal of learning versatile and generalizable filters in the case of CNNs and use them in the second network. It is possible to use only a subset of pre-trained parameters and use random initialization for the rest as before. This technique is called fine-tuning, which is used in literature for object recognition with the 1.2 million image training dataset for the ILSVRC 1,000-class classification challenge [DDS$^+$09, JSD$^+$14, RHGS15]. The network can then be modified by changing layers and even switching out the top layer to train and use it for a different task, e.g.

pre-training for classification and fine-tuning for style recognition. This works well, as low-level convolutional features such as colors and edges are shared among different tasks. Examples of low-level features are shown in Figure 2.5. PersonData is one such dataset usable for pre-training for tasks requiring person images. Its benefit is evaluated in the results chapter. Fine-tuning can be controlled by setting different learning rates for different neurons or layers. One approach is to fix low- and mid-level layers by setting their learning rates to 0 and to retrain only high-level or switched-out layers. It is also possible to give selected layers a higher learning rate to retrain the network with a focus on newly initialized parameters.

Another important aspect of training is to shuffle the training data to have approximately the same distribution in minibatches as in the entire dataset. In case one has binary labeled data, with the first half having only one label and the second the other, the network can overfit to the first class. Thus, training with thousands of images with the same class can train the network so that any input image is of that class, since zero-valued weights can be learned to always achieve the respective label and since these weights cannot be updated to adapt to another class because they always produce zero-valued output due to multiplication.

Learning rates, their decay and solver algorithms are hyperparameters which have to be chosen, resulting in a problem similar to that of choosing a network architecture. One way is to use the default settings also used in the training of benchmark winners and to adapt them when necessary. Our choice of hyperparameters is explained in the implementation chapter.

There are various methods of data augmentation that aim at increasing the size of the training dataset using knowledge about its contents [PRH+14]. The most commonly used data augmentation for object recognition is horizontally flipping images [PRH+14]. A horizontally flipped image of a person, car or many other classes is still a completely valid image, which is, however, a different image for the model to train. Using horizontal flips allows for doubling the size of the training data in these cases. Data augmentation methods differ in computation time, generalizability of their underlying assumptions and effectiveness. For example, applying a translational jitter to input images is relatively irrelevant for CNNs compared to flipping because CNNs display translation invariance due to convolutions [PRH+14]. Other methods include rotation, scale jittering, color information jittering and random cropping. Figure 5.8 depicts some examples. Furthermore, some augmentations are applicable on the fly during training. This has the benefit of not having to precompute and store such augmentations, potentially saving a lot of disc space. In our training we use random horizontal flipping and random cropping on the fly. Our network expects images with a width and height of 227 pixels. However, we store PersonData images as 256×256 pixels and therefore use the random cropping mechanism to directly crop images to the network's required input dimensions, so they do not have to be rescaled.

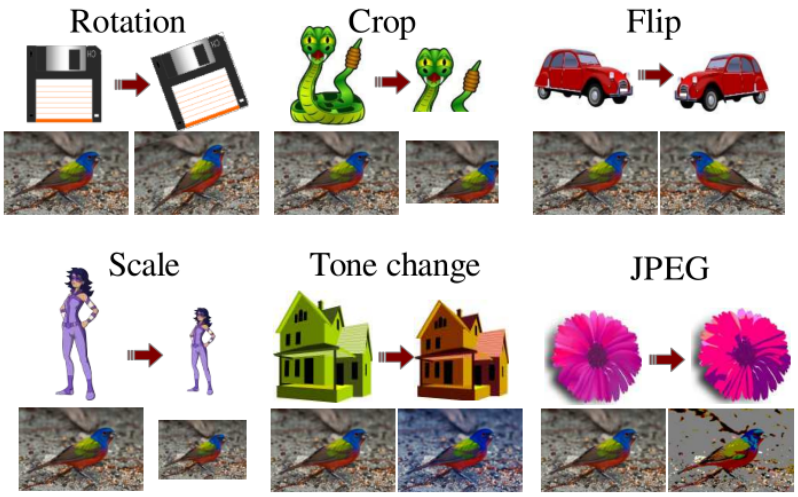


Figure 5.8: Examples of data augmentations. Images can be rotated, cropped, flipped and scaled while retaining their original labels. Image from [PRH+14].

Binary Classification with Random Negatives

The PersonData dataset yields over 300,000 person image crops, which are used for training purposes. Only about one third of the datasets provide explicitly labeled nonperson images, often in the form of empty scenes, i.e. camera photos of street and other outdoor scenes not containing any people. Negative data is collected from these images by randomly selecting images and then randomly cropping out image parts. This is not ideal, as there is little control over the results of the random choices. There are too many to check manually, lacking additional annotation. Sampling some negative crops resulting from this procedure shows that they are not representative enough. Figure 5.9 shows sample negative crops, which include mostly parts of cars, sidewalks and houses, given that the images represent empty street scenes, which make up the majority of the provided negative data. Training our CNN as described in the previous sections yields a testing accuracy of 99%. In addition, sampled false positives as shown in Figure 5.10 suggest that the network overfits, e.g. to pets. Evaluation is made difficult by the fact that the random cropping procedure can lead to the same or very similar images in training and in the test set as the same image is often used to create crops multiple times in the process. We conclude from our results that a classifier trained like this learns a form of objectness score rather than a separate person versus nonperson classification.

5.2.3 Binary Classification with Additional Labels

As random negative iamges create problems, a second type of negative class is used to allow the evaluation of the PersonData dataset and the trained network. More precisely, the negative class is defined by multiple subclasses. The MSCOCO dataset contains

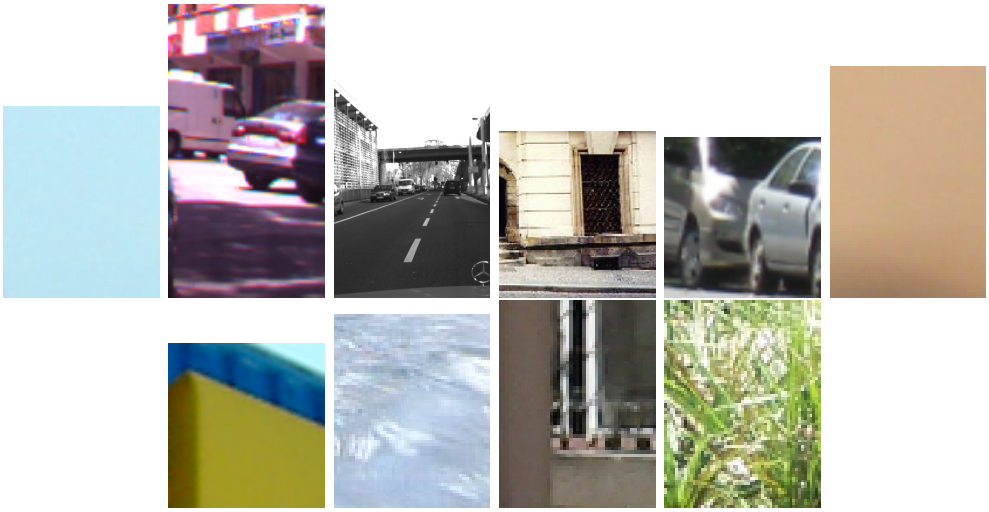


Figure 5.9: Random crops from explicitly labeled negative images usually include cars, walls and homogeneous areas such as skies and asphalt.

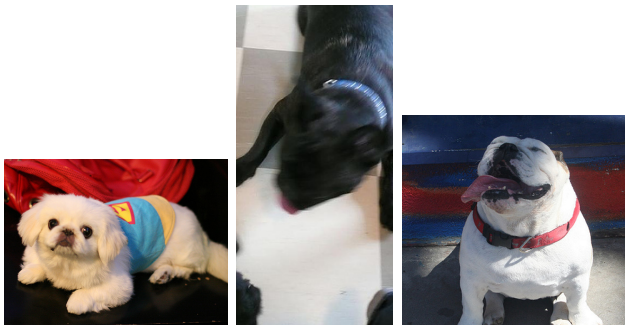


Figure 5.10: False positives of the classifier trained with random negative crops suggest overfitting to pets most likely due to inclusion labeling noise.

segmentation and bounding box annotations for 80 classes divided into 12 categories. The main benefit is that the negative class now is not made up of mostly car parts and crops of unlabeled empty street scenes but instead has more detailed labels such as backpack, chair, etc. An additional benefit is that the person class is its own category, allowing three views of the data to be evaluated while keeping the person class constant. Thus, the negative class consists of either either the remaining 79 nonperson classes or the remaining 11 nonperson categories, or the entirety of nonperson data is used. Note that PersonData also contains the person images of MSCOCO. Thus, the combination can also be seen as augmenting the MSCOCO person class. Furthermore, the amount of negative images in the 79 classes is roughly similar to the amount of positive images. To be precise, there are 343,814 positive and 268,770 negative images resulting in a distribution of $56 : 44$. Moreover, the number of images per class in MSCOCO is uniformly distributed, thus not favoring any one class over another. Since the softmax function is defined for a

set amount of classes, the networks for the binary, categorical or multiclass view naturally have different numbers of output neurons, i.e. 2, 12 or 80.

5.3 Feature-Map-based Region Proposals

CNN feature map visualizations give insights into the type of features learned. Analyzing feature maps of dynamic video feeds reveals information about how such video feeds change with movement. In higher layers, high-level features such as faces as shown in Figure 5.11 can be detected by spotting the feature maps that show strong activations correlating with the expected movement, e.g. head movements. This suggests that the features learned in higher convolutional layers are already able to represent objects well. Additionally, the results of Yosinski et al. [YCN+15] show that features of the fifth convolutional layer understand complex shapes like animal bodies or object classes, as is demonstrated by regularized optimal input stimuli in Figure 5.12. Motivated by this fact, we try to develop a region proposal algorithm based on feature maps usable in a Fast R-CNN detection pipeline. However, as empirically discovered and additionally confirmed by Oquab et al., feature maps generally contain only approximate information about object localization [OBLS15]. That is to say, for example, that when training a network for person classification the network can hypothetically learn that the existence of a head suffices for accurate classification. In this case, no other features such as torsos, legs or feet have to be learned and have no activation in feature maps. For an accurate bounding box, features that encompass the entire body have to be learned. Therefore, as information is missing, no bounding box can be computed of the entire person but only of their head. However, as a head feature is learned, it can be used as an approximate location. What is more, there is no general prior information about what feature is learned in a specific feature map and also about in which feature map a feature is learned. Therefore, it is not possible to find, for example, the feature map containing the head with only prior knowledge, as it may not exist or vary in position or can be combined with another feature, e.g. as an upper-body feature. For the same reason manually constructing a detector around selected features using statistical analysis of feature maps is not guaranteed to work well, e.g. features containing the head and feet of a person are needed for an accurate bounding box, however, they are not guaranteed to exist in the first place.

Therefore, several methods to heuristically generate region proposals from feature maps are examined. First, the receptive fields of the strongest activations in arbitrary feature maps are calculated. Receptive fields of higher layers are, however, relatively large because the extents are increased by reversing convolutions and pooling (cf. Figure 5.13) and may span almost the entire image, making them impractical for region proposals, as they can span multiple objects.

Another idea is to aggregate multiple features maps together as a pixel-wise sum, maximum or average, allowing for random combination of features in separate feature maps that are important in combination to find object extents, e.g. heads and feet. On these

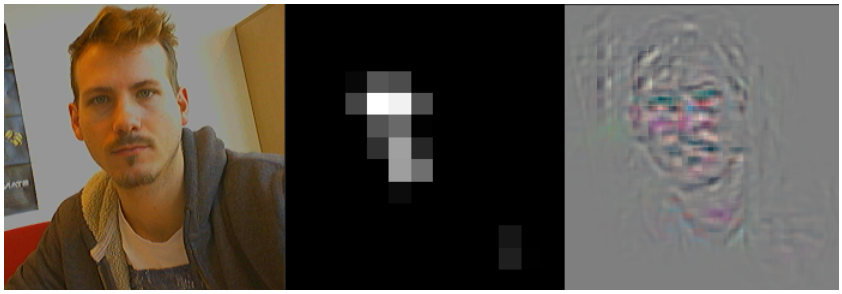


Figure 5.11: Left: Input image. Middle: Feature map of the fifth convolutional layer with high activations in the are of the head. Right: Deconvolution of the feature map.

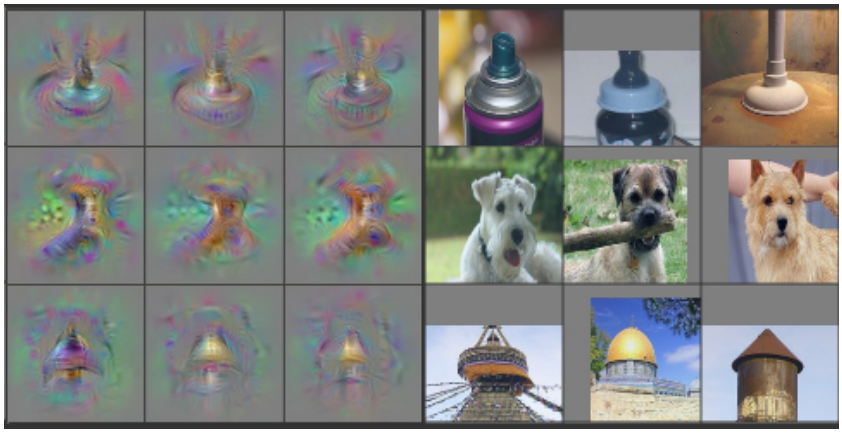


Figure 5.12: Each row shows an object class and regularized optimal input-stimulus images, which lead to strong activations in neurons, showing class-specific features. Images from [YCN+15].

aggregated feature maps, region growing is used, and growth is based on the activation value, resulting in structures encompassing high activations of multiple features depending on the amount of feature maps combined. Difficulties with this method are the choice of thresholds as the stopping condition or the number of feature maps to combine. Figure 5.14 shows sample RoI computations for one image based on multiple random feature map aggregates of the fifth convolutional layer. These results demonstrate that arbitrary feature maps do not suffice for this task, since relevant information is lost, e.g. there are no strong activations around the left person.

The advantages of feature maps for region proposal generation are that feature maps in higher layers contain robust high-level features and that they are smaller than the original image due to convolutions and pooling without border padding. The disadvantages are as discussed above, i.e. missing information and no prior knowledge. The region proposal algorithms Selective Search and EdgeBoxes work on low-level features of the input image and thus do not have the problem of missing object parts. Furthermore, RPNs are able

Figure 5.13: Receptive field of a high-level neuron that covers a large part of the input image. Image from [HZRS14].

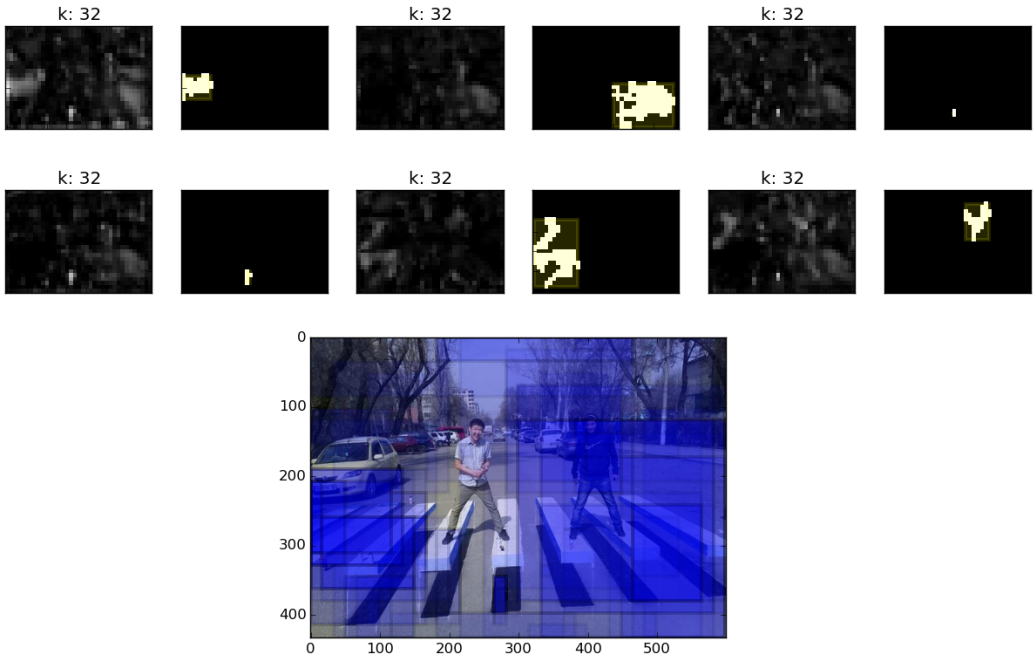


Figure 5.14: Top: Multiple aggregates of 32 random feature maps of the fifth convolutional layer and region growing performed on them, respectively. Bottom: RoI bounding boxes for the computed regions.

to use high-level features to calculate objectness scores and multiple region proposals at every feature map pixel. The combined training of object extents, objectness, object class and region proposals ensures that features able to predict accurate object bounds are learned and achieves state of the art in object detection with CNNs [RHGS15]. However, RPNs were published after our experiments and because they are not straightforward to implement or use, there was not enough time to use them for additional evaluation.

5.4 Chapter Summary

This chapter introduces the methodology of our work. First, our dataset pool *PersonData* is introduced, which consists of 31 datasets for person recognition or object recognition with person annotations. The method of extracting images and handling issues such as video data overfitting are explained and the distribution of extracted images are demonstrated. For classification, the selected CNN architecture is introduced and the stochastic gradient descent solver as well as initialization methods are discussed. Afterwards, negative class composition by either random cropping or subclass labeling are discussed. Finally, the idea of RoI proposal generation based on high-level feature maps using receptive fields or strong activations is explained.

CHAPTER 6

Implementation

Deep CNNs contain millions of parameters that are learned by training. All parameters as well as gradients and error updates have to be held in memory during training. Forward passes through the network and training with backpropagation require computations on all of these parameters and neuron gradients. Exploiting mathematical properties of convolutions, parallel and efficient GPU implementations have been developed, such as the Nvidia Cuda library, which can be augmented even further for deep learning with the cuDNN library. This allows for training times two orders of magnitude smaller than on the CPU. However, specific hardware designed to work with these libraries and with enough capacity to hold the data is required.

6.1 Hardware

We use an Nvidia GTX 960 in our setup to benefit from faster training times in deep learning frameworks with the support of Cuda and cuDNN. We also choose this particular model due to budget restrictions and Cuda compatibility. It has the minimum requirements for basic CNNs. For example, we find that its 4 GB memory is almost used completely during training the CaffeNet model with a batch size of 64, which is another important reasons for the model choice, since - although there are better-performing models such as GoogLeNet and VGG - adding more layers would require even smaller batch sizes and would reach hardware limitations more quickly. A 256 GB SSD is used for quick access to applications and disk space for small data such as documents. A 1 TB HDD is used for larger data such as stored, trained network parameters, which for CaffeNet are around a few hundred MB each, and datasets, which quickly sum up due to copies and modifications. The CPU used is an Intel Xeon E5-1607 v3 with four cores and 3.1 GHz.

6.2 Software and Frameworks

The operating system chosen is an Ubuntu Linux distribution, as it is freely available and as it is the primary supported operating system of the Caffe framework. To utilize the GPU's capabilities, we install the libraries Cuda 7.5 and cuDNN v3. Scripts for dataset creation, merging and modification as well as scripts for file conversion and more are written in Python 2.7 as it is free, works well for rapid prototyping and the Caffe framework also provides python wrappers and scripts.

There are various deep learning frameworks capable of creating, training and using CNNs. These frameworks vary in speed, abstraction level, modifiability and creation of new layers. Table 6.1 lists various frameworks along with their programming language. While preferences may vary, web research concludes that there is no single best network. We opt for Caffe because of its high abstraction level, which makes network architecture definition, solver and application relatively easy. Theano allows for easier modification, as networks are completely defined in Python code and symbolic gradients make it possible to do rapid prototyping of new layers without having to worry about manually calculating gradients for backpropagation as is the case in Caffe, where Cuda or c++ implementations of forward and backward passes have to be written. However, creation of new layers is not our goal; therefore, Caffe's simpler usage is more beneficial. Tensorflow is a promising new framework maintained by Google and open to the public since November 2015. At its time of publishing, however, it was still slower than Caffe and did not support the newest Cuda version. Its benefits are Python and interactive-graph-based control of optimized c++ code.

Framework	Programming Language
Caffe	c++, Cuda, Python
Theano	Python
Torch	Lua
matconvnet	Matlab
cuda-convnet	c++, Cuda
Deeplearning4j	Java, Scala
tiny-cnn	c++
Tensorflow	python

Table 6.1: List of deep learning frameworks with programming languages used and additional information.

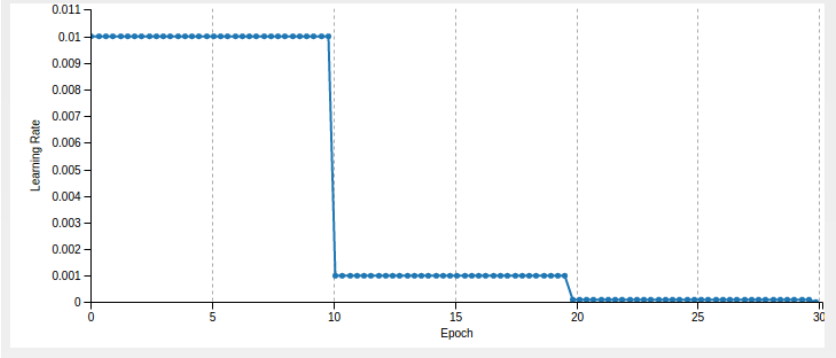
When working with Caffe directly, most programming is done using the Python wrapper, abstracting c++ functionality for quick prototyping possibilities. However, Nvidia DIGITS is used as a second framework that works on top of Caffe, or more precisely a separate Caffe fork by Nvidia of the official repository. DIGITS is a job-based library for simplified and multi-threaded preparation of datasets for use with Caffe in database formats such as LMDB and for training with visualizations of current training loss and

validation accuracies. Since CNNs such as CaffeNet that have fully connected layers require fixed size input, DIGITS provides options during database creation to either use cropping, resizing or filling with Gaussian noise to refactor images to identical dimensions. For training, DIGITS provides functionality for on the fly data augmentation using random crops with a fixed size, e.g. randomly taking 227×227 regions from a 256×256 image in our case, and random horizontal flipping. Caffe solver definition requires the amount of total training iterations, where one image forward and backward pass counts as one iteration. DIGITS facilitates solver definition by an abstraction to epochs, where one epoch defines iterating over the entire dataset once. Furthermore, DIGITS visualizes learning rate decay as shown in Figure 6.1a. Visualizations of training loss, validation loss and validation accuracy in an interactive, constantly updating graph make it easy to see whether or not training is going well. One example of such a graph is depicted in Figure 6.1b. During or after training, it is possible to select a stored snapshot of the network state and use it to classify one image and visualize layer weights, activation distributions and feature maps of the forward pass as shown in Figure 6.2. Additionally, classifications for multiple images can be calculated along with presentation of the top results per class. A disadvantage is that, because DIGITS works on a fork of Caffe, new developments of the main Caffe repository are merged into it only when the developers of DIGITS decide to do so.

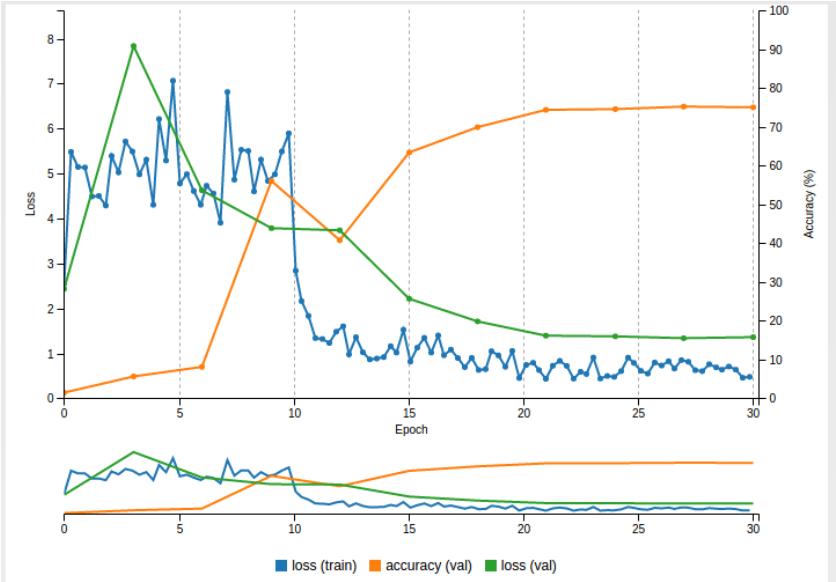
The DeepVis toolbox [YCN+15] is used to help gain intuitive knowledge about the inner workings of convolutional neural networks. The toolbox visualizes feature maps of all layers of the CaffeNet model. This includes convolutional, pooling and fully connected layers. Furthermore, a specific feature map in a layer, i.e. output of one learned filter, can be selected and a deconvolution [ZF14] can be calculated to visualize how this filter sees the input image. An example of such a visualization is shown in Figure 5.11. Also, pre-calculated regularized optimizations of the network's neurons for the ImageNet dataset can be shown, as demonstrated in the paper of Yosinski et al. [YCN+15]. These optimizations visualize what types of images lead to the highest activation of a particular feature map or neuron (cf. 5.12). Another feature is the possibility of dynamic video input using a webcam. This method allows us to explore activations with dynamically changing input and enables empirical recognition of feature map features such as, for example, a feature map that learns human heads, as the activation moves similarly to the human head in the input feed.

6.3 Scripts

Iterating datasets in our collected dataset pool, analyzing folder structures and file information, doing hard negative and positive mining, converting annotation formats and other smaller algorithms are implemented using Python scripts. Evaluating test sets and analyzing the potential of feature maps for region proposals is done utilizing Caffe's Python wrapper.



(a)



(b)

Figure 6.1: Top: Learning rate decay is visualized over training epochs. Here, a step function decay is used and the learning rate is divided by 10 after one-third and two-thirds of training. Bottom: Training loss and validation loss and accuracy are plotted over training epochs, allowing immediate understanding of how well training is currently proceeding.

6.3.1 Data Preparation

Since datasets have different formats of underlying data and annotations, it can be necessary to convert files for easier usage. Our dataset iterator scripts use Python and rely on the Python OpenCV library for image processing. Therefore, understandable input consists of images and text-based formats such as plain text or XML annotations. Because the Caltech and Daimler Multi-Cue Occluded Pedestrian Classification Benchmark datasets do not have such formats, their data has to be converted. Caltech contains street

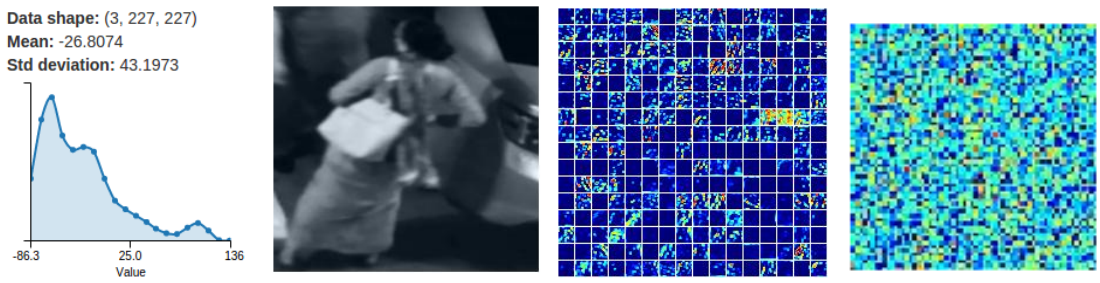


Figure 6.2: From left to right: Input value distribution, input image, feature maps and activations of a fully connected layer.

scenes as image sequences in a SEQ format, which are split into separate image files, and annotations in a VBB file with a Matlab library to read them; these are converted to plain text using the provided Matlab library in Octave. The Daimler dataset contains image collections stored as MAT files, which are also split into separate images using Octave.

The PETS 2009 dataset is used for evaluation of generalization and the benefit of pre-training with PersonData. As no ground truth annotation is provided with the original dataset, annotation is used from Andriyenko et al. [ARS11] Specifically, the scenes *S2L1V1* and *S2L2V1* are used. However, as no negative, i.e. person-less, images are available, a negative image is constructed manually for both scenes by stitching together image parts that do not contain persons, which is possible since the camera has a fixed position. Sample images and a stitched image are shown in Figure 6.3. From the stitched images, crops of the size 50×100 pixels are cut out on a regular grid. This crop size was chosen because this is the approximate size of people in the images. The grid is constructed with horizontal and vertical steps of one-quarter the width and height of the crops, respectively. This results in 1,239 overlapping crops of a 768×576 image. The number of person annotations are 4,650 and 10,292 for *S2L1V1* and *S2L2V1*, respectively. Samples of cropped person and background images are shown in Figure 6.4.

6.3.2 Dataset Merging

We propose a method of using a collected pool of datasets to extract application-specific data. This is done by using a main script that utilizes individual scripts for iterating the datasets and returning the information necessary to create an application-dependent view of the entire data. We implement this with Python scripts exploiting the *yield* functionality to iteratively return results from one method usable in another while maintaining the first method's state, which is beneficial for parsing complicated folder structures and returning only bits of information. First, datasets to be parsed are declared in the main script with their names, the path to the dataset's folder, the specific iterator script, a variable to define how many images of a video sequence should be used, a constraint for minimum bounding box dimensions and potential extra parameters. The

Figure 6.3: In the top row, three sample images of a scene in the PETS dataset are shown. On the bottom, the stitched empty background image can be seen, which is manually stitched together from multiple sample images.

datasets are then iterated one after the other, returning the class label, the path to the image file, bounding box coordinates and the video frame number of that image if available. Next, video data is filtered by taking only every n^{th} frame according to the variable set for the current dataset to avoid redundant information, like images of the same person, e.g. in a walking cycle, that are too similar. Since it is possible that annotations exist for missing images, the images' existence has to be checked. Afterwards, the bounding box coordinates are reordered and cut to image bounds, because annotations can have bounding boxes exceeding outside image bounds and because coordinates are not always necessarily in the correct order. In the next step, the adapted bounding box is compared to the minimum bounding box requirement, which in our case is 40×60 pixels. Lastly, the image is cropped to the bounding box and written to the output directory.

The individual dataset iteration scripts accept the dataset root path and, if required, additional arguments as parameters. They then iterate over the specific folder structures and read the specifically formatted annotation. To allow variable ordering and easy addition or modification of information that is returned, it is returned as a dictionary. Additionally, an information type variable is returned, such as *crop* for image crops in our case, which allows passing various types of information that can be used differently in the main script for other applications than just image classification. Together, these scripts are able to create a modifiable application-specific view of the dataset pool. The

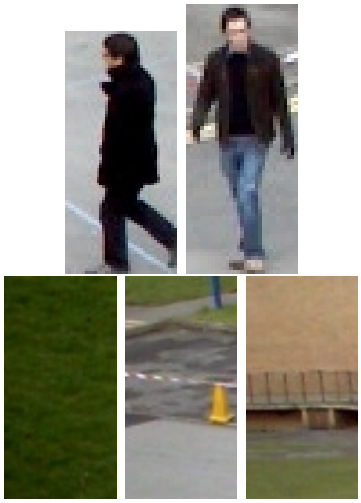


Figure 6.4: Samples of image crops from the PETS dataset for positive images (top) and negative images (bottom).

view allows for simple extensions through generalization and reuse of code.

6.3.3 Training, Validation and Test Set Creation

The labeled image crops for classification extracted from the dataset pool have to be split into training, validation and test sets to train and evaluate the neural networks. DIGITS accepts text files for each split, where each line is a ground truth annotation containing the image file path and its label. Therefore, the following procedure is used to create these files. First, parameters are given for the percentages of each split. In the evaluation of the difference between binary, categorical and full 80-class labeling we use the rule of thumb according to the Pareto Principle and select 20% as the test set, and split the remainder again into a 80% training set and a 20% validation set. Image paths are stored in a list per class label and shuffled to randomize the dataset source per class. Next the specified percentages for training, validation and test splits are selected per class and combined into one list for each split. These lists are then shuffled again to randomize class order. The final lists are then written into text files with absolute image paths, and the class label separated by a white space. The class labels are numbers; a separate label text file is used to map these numbers to class names.

6.3.4 Classification Evaluation

For the evaluation of different, trained networks on various test sets, an evaluation algorithm is implemented using Caffe's Python wrapper. It is able to load a model, load ground truth data, classify the test set and compare the results with binary accuracy, recall and precision. It is also able to calculate confusion matrices and select the best and worst classifications.

To fully initialize the model in the framework, trained weights, which are stored as a *caffemodel* file, a mean image, the test-time definition of the network architecture in the *protobuf* format and a file containing the ordered labels to map ground truth label numbers to names are needed. Figure 6.5 shows an example of a mean image, in which a weak silhouette can be seen. This silhouette demonstrates that the majority of data are pedestrian images. Additionally, the silhouette is symmetrical, which confirms the assumption that person images are invariant to horizontal flipping. For the ground truth input, a text file containing a list of ground truth items as explained in the previous section is expected.

Figure 6.5: Mean images of the PersonData dataset on the left and the PETS crops on the right.

Using the caffe *net* class, the network is initialized with the required data and a *transformer* is initialized for pre-processing input images as the network requires. Pre-processing includes resizing the image to the required fix size, switching the color channel order to BGR, reordering the image dimensions to $channel \times y \times x$ and subtracting a per-pixel mean value calculated from the training set's mean image. This is identical to the pre-processing done in training.

Next, the test images are loaded, pre-processed and classified in batches. Interestingly, although the batch size is limited to 64 images during training due to GPU memory limits, at test time a batch size of 256 images only produced about a half the memory load, i.e. 2 GB. This may be due to the additional data required during training such as gradients and errors per neuron. Even higher batch sizes were tested; however, we did not notice any speedup, the reason for which may be the bottleneck of CPU-to-GPU memory transfer limited by bandwidth.

The classification results have different dimensions depending on the model, i.e. 80, 12 or 2 values for the 80-class, 12-class or 2-class model respectively. The output is binarized, i.e the 79 or 11 negative classes are merged into one negative label and true positives, true negatives, false positives and false negatives are computed for correct and incorrect person and nonperson classifications, respectively. Accuracy, recall and precision are calculated using these values according to the formulas of Equation 6.1, Equation 6.2

and Equation 6.3 where TP, TN, FP and FN are true and false positives and negatives, respectively.

$$Accuracy := \frac{TP + TN}{TP + FP + TN + FN} \tag{6.1}$$

$$Precision := \frac{TP}{TP + FP} \tag{6.2}$$

$$Recall := \frac{TP}{TP + FN} \tag{6.3}$$

Afterwards, a confusion matrix is constructed to encode classification results. We use the highest winning class of the softmax output as the classification, because softmax scores are interpreted as probabilities and thus the highest score is the highest class probability. The confusion matrix encodes the results using the input class as the y-axis, the output class as the x-axis and the percentage of the respective classifications as the cell value. Note that our ground truth test set is always labeled with the full 80-class labeling unless otherwise specified. This permits comparing even binary classification results to the more strongly annotated ground truth, i.e. visualizing the percentage of, e.g., backpacks that are classified as person or background. Figure 6.6 depicts three such confusion matrices for different input models. Additionally, another matrix is calculated, where softmax probabilities are summed up directly rather than summing up classifications. These probabilities can show classification tendencies not visible in the confusion matrix. For example, for a softmax output of $[0.5, 0.4, 0.1]$, the confusion matrix would lose the similarity of the first and second output, as it would instead use $[1, 0, 0]$. However, given enough data, both matrices converge to a similar result and the confusion matrix suffices for evaluation.

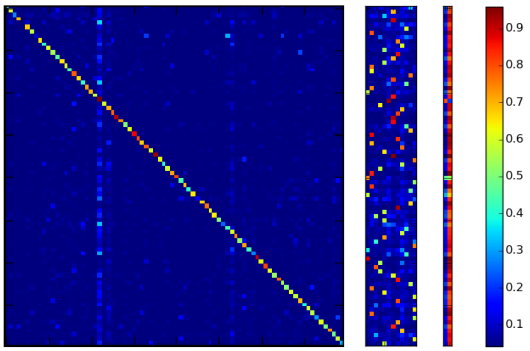


Figure 6.6: Confusion matrices for ground truth with full 80-class labeling and 80-class, 12-class and 2-class output.

Lastly, to extract the best and worst classifications, prediction errors are calculated by subtracting the classification scores from the ground truth. The data is sorted accordingly and the images resulting in the 100 highest and lowest scores are stored in an output folder.

6.4 Training Details

The CNNs are trained using the DIGITS framework for a manageable and supervisable training pipeline, which provides a better overview than the pure console log of Caffe. The data is required to be in a database format, for which we choose the default LMDB, and is converted using DIGITS as well. For a created database, the framework also shows class distribution and computes an image mean as shown in Figure 6.5.

The model used is the CaffeNet model with modifications such as the number of output neurons. The detailed layer parameters and layer oder is shown in Table 6.2.

Layer	Parameters
Convolutional 1	n 96, k 11, s 4, p 0, ReLU output
Max-pooling 1	k 3, s 2, p 0
Local response normalization	local size 5, α 0.0001, β 0.75
Convolutional 2	n 256, k 5, s 1, p 2, ReLU output
Max-pooling 2	k 3, s 2, p 0
Local response normalization	local size 5, α 0.0001, β 0.75
Convolutional 3	n 384, k 3, s 1, p 1, ReLU output
Convolutional 4	n 384, k 3, s 1, p 1, ReLU output
Convolutional 5	n 256, k 3, s 1, p 1, ReLU output
Max-pooling 5	k 3, s 2, p 0
Fully connected 6	n 4,096, dropout 0.5, ReLU output
Fully connected 7	n 4,096, dropout 0.5, ReLU output
Fully connected 8	n 80 or 12 or 2, softmax output

Table 6.2: The CaffeNet network architecture. The order of layers is the order in which data is passed through. n is the amount of output feature maps or neurons, k is kernel size, s is stride and p is padding.

The layer weights are initialized with random values from a Gaussian distribution and biases are initialized as zero per default. Additionally, biases have a learning rate multiplier of 2; therefore, propagated errors have more effect on biases than on weights.

The solver used for training is stochastic gradient descent with momentum. The learning rate is chosen as the default 0.01 and is stepped down after 33% of training to 0.001 and after 66% to 0.0001 (cf. Figure 6.1a).

6.5 Feature-Map-Based Region Proposals

The network is loaded and data is pre-processed in the same way as in the evaluation algorithm. A single forward pass is executed for an input image and the activations in the feature maps of the convolutional, pooling and LRN layers are stored as matrices. The Python library *numpy* is useful for efficient handling of such large matrices since it is built around N-dimensional array objects and linear algebra. To allow arbitrarily sized input images, the fully connected layers were cut out of the model definition. The remaining layers can natively process input of arbitrary dimensions.

6.6 Chapter Summary

In this chapter the implementation details of the applied methods are discussed. The utilized hardware is explained with respect to storage space, GPU and GPU libraries and the CPU. Next, the software components such as the operating system, scripting language and the CNN frameworks Caffe and DIGITS are introduced. Then, the algorithmic details for dataset preparation and merging, the creation of training and test splits and for evaluation and computation of confusion matrices are explained. Afterwards, details on the hyperparameters of the chosen CNN are discussed and further information on the implementation of the region proposal generation is given.

CHAPTER 7

Evaluation and Results

Data from more than 30 sources is collected and merged to create PersonData, yielding over 300,000 labeled person images for image classification training. We evaluate the performance and properties of CaffeNet trained with PersonData to prove the following hypotheses. First, we show that binary classification is more accurate when defining the catch-all negative class by multiple subclasses. Second, we analyze the generalization capabilities of the classifier network and changes to the amount of data provided. Lastly, we show that classification on a small dataset of a few thousand images can be improved by initializing the weights of the network with pre-trained weights using the PersonData dataset.

7.1 Binary Classification with Multiple Labels

Taking the MSCOCO dataset, nonperson classes as the negative class for the binary person classification task has the significant advantage of stronger labeling. The negative class is provided as 79 separate, labeled classes such as *chair* or *car* and is additionally annotated with category labels such as *indoor*, since MSCOCO has 80 classes, one of which is the person class. Therefore, it is possible to train the network with three views of the data, i.e. as an 80-class, 12-class or 2-class classifier, while having the full 80-class ground truth labeling for evaluation purposes.

The network is trained as described in the previous chapter, i.e. with stochastic gradient descent similar to literature [KSH12] and the specified training, validation and test data splits for 30 epochs. 30 epochs were chosen to limit training time, which was about 15 hours for each network, and still achieve a high converging accuracy as demonstrated below. Table 7.1 shows the results for the differently trained networks. A problem occurres when training the binary network, as it does not learn, producing 56% accuracy, which is the same as random guessing, as the distribution of positive and negative images is 56:44. Visualizing the weights of the first convolutional layer shows that the weights

almost do not change during training, as seen in Figure 7.1. Base learning rates of 0.1, 10 and 100 times the initial value as well as polynomial decay instead of step function decay do not change the fact that the initial random weights remain the same. When using category or full labeling this problem does not occur. A possible explanation is that the 79 classes are too diverse to be trained with one label. Minibatches of 64 images can never include all classes at once, which shifts learned weights toward the classes contained in the batch. At the same time, it is difficult to learn weights for one label containing multiple dissimilar classes, since a backpack has different features than a car but both share the same negative label. Therefore, additional options are considered. First, initializing the binary network with the weights from the fully labeled one results in equally good results. However, this needs the fully labeled data for pre-training. Nevertheless, as the network trains correctly when initialized not randomly, this suggests that the problem is indeed the network's initialization and a possible workaround is to initialize it with a different large dataset such as ImageNet. Second, initializing the binary network randomly like the others but using the Adagrad solver instead of stochastic gradient descent yields at least better results of 86.15%. Adagrad is an extension of gradient descent where per-parameter learning rates are used and adapted so that rarely updated parameters get higher learning rates. However, Adagrad is designed for convex problems, which neural network training is not, and therefore results in less optimal solutions. Lastly, including MVN layers in the network, i.e. adding an MVN layer after the third convolutional, the last pooling and the second fully connected layer, and using Adagrad results in a more comparable accuracy of 94.24%. MVN aids in training by normalizing distribution of input or activations by doing away with the need to re-adapt the following neurons to differently distributed data at each update. This is especially beneficial in this case, where multiple classes are merged into one, because the problem is using the same label for different classes.

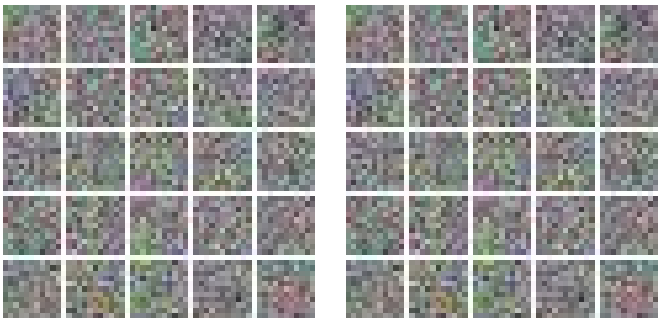


Figure 7.1: Weights before (left) and after (right) training a 2-class model with standard stochastic gradient descent. There is almost no change, showing that the network does not learn features.

The validation accuracy provided in the table is calculated for the number of classes of the respective models. Therefore, the validation accuracy of binary models is almost the same as the test set accuracy, whereas there is a greater gap when looking at multiclass

model	val. acc. %	acc. %	recall %	precision %
80-class	82.60	95.51	**98.01**	94.21
80-class (MVN)	67.41	91.56	87.17	97.53
12-class	87.27	**96.05**	97.38	95.66
12-class (MVN)	75.11	90.24	83.31	**99.16**
2-class	56.11			
2-class (80 pre-trained)	*96.49*			
2-class (Adagrad)	86.15	85.63	82.50	91.06
2-class (Adagrad and MVN)	**94.24**	94.07	93.94	95.42

Table 7.1: Results of models tested for 80-class, 12-class and 2-class labeling. Validation accuracy is the accuracy measured at the last epoch of training and is calculated using all classes of the model while test set accuracy, precision and recall are calculated for the binary problem.

models. In the multiclass validation accuracy classifying one negative class as another, e.g. classifying a car as a backpack, counts as a misclassification, whereas these intranegative misclassifications are ignored in the binary measurements, as it is not important for the task of person nonperson classification.

The 12-class model without MVN layers achieves the highest binary test set accuracy. Interestingly, MVN layers only achieve higher accuracy for the 2-class model trained with Adagrad and only improves precision for the other models, which shows that one cannot simply claim that MVN improves performance in general. In the first case, MVN can help, as described above, i.e. neurons do not have to adapt strongly for different classes. However, for the 12- and 80-class models, it leads to drops in recall of over 10% and increases in precision of less than 4%. Figure 7.2 shows for the 80-class model that after adding MVN layers, the first convolutional layer weights are much more sparse and have less contrast than without MVN layers. This extends further into the network, and higher layers also have strong activations and weights only in half of the available space.

Comparing the accuracies of the individually best-performing models for each amount of classes shows that the 12-class model performs best with 96.05%, which is 0.54% higher than the 80-class model and 1.98% higher than the 2-class model. Considering that after adding MVN, the 80-class model achieves the highest accuracy than the results of the previous section, it is difficult to say which amount of labeling is the best since several components have to be taken into account. such as hyperparameters, solver, frameworks and implementations. However, we hypothesize that there is a best level of labeling that depends on architecture and data, which in our case is categorical labeling. In the general case, using more labels has the main benefit of avoiding the initialization and training problems experienced with the 2-class model.

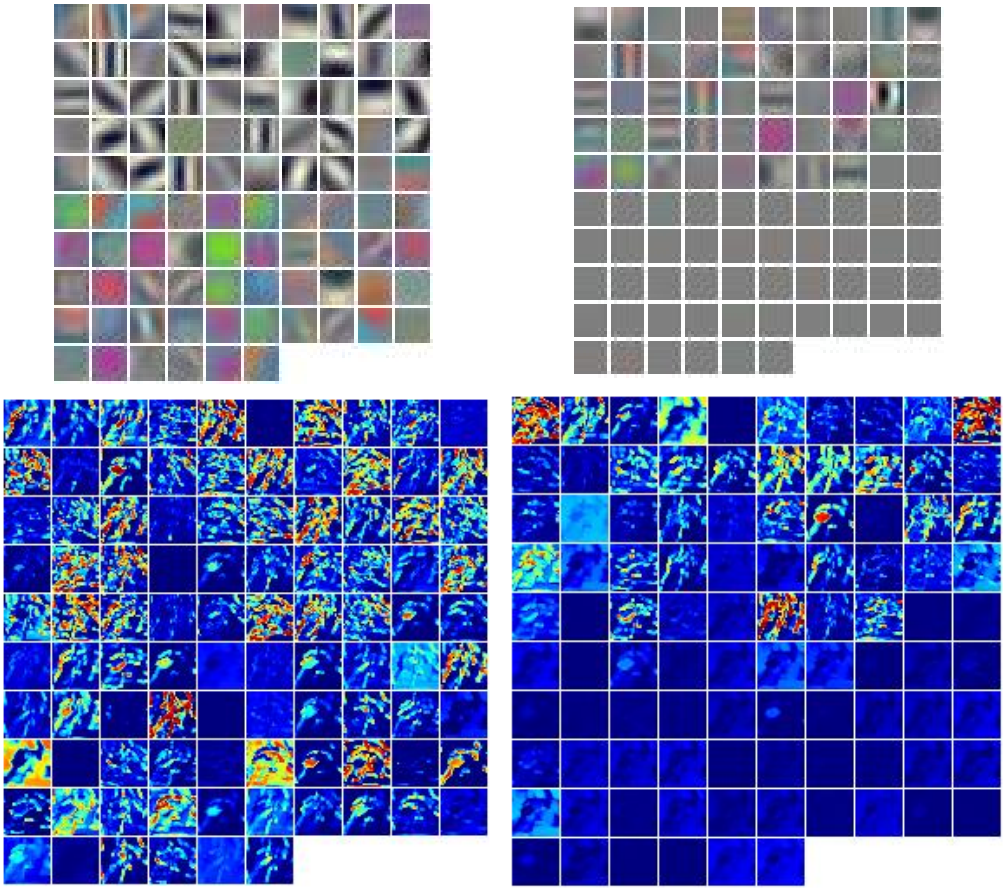


Figure 7.2: Left: Weights of the first convolutional layer and feature map activations for a 80-class model with the standard CaffeNet architecture. Right: Weights and feature map activations are much more sparse after extending the architecture with three MVN layers.

7.2 Performance Evaluation

The following tests are performed using an 80-class model trained for 50 epochs including the MVN layers as explained in the previous section and using a different percentage of training data, i.e. using 90% of PersonData as the training set and the remaining 10% as the validation set in an effort to maximize the use of the data to achieve higher accuracy. As the validation set was not used for any hyperparameter tuning, which would overfit the model to the validation set, it is usable directly as the test set as well.

The model is used to evaluate which classes lead to misclassifications, overfitting on video data, generalization and pre-training benefits, as discussed in the following.

7.2.1 Classification Analysis

It is important to analyze correct and incorrect results of the classification network to gain insights into the data, the network and their respective strengths and weaknesses. Therefore, we evaluate the classifications on the CaffeNet model extended by MVN layers on PersonData, the results of which are shown in Table 7.2, by computing the confusion matrix and showing the the top classification scores for person images and the percentage of input images per class that are classified as person in Table 7.3. The confusion matrix is shown in Figure 7.3.

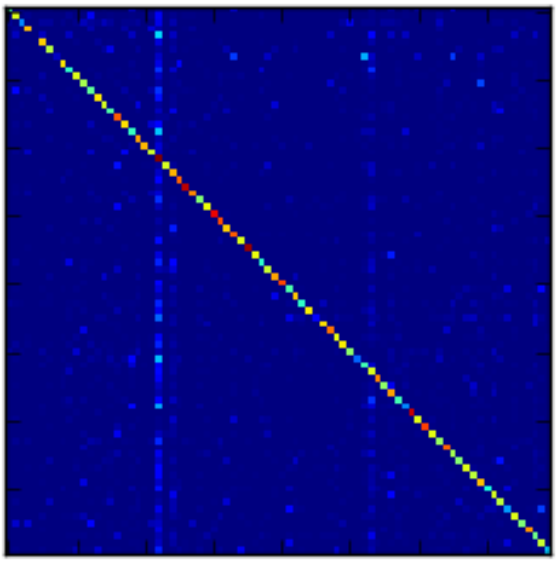


Figure 7.3: Confusion matrix of the 80-class MVN model with 90% training data.

model	val. acc. %	acc. %	recall %	precision %
80-class	85.89	97.18	97.87	97.13

Table 7.2: Results on the test set after training 50 epochs with 90% of PersonData as the training set.

Person images are classified correctly 97.87% of the time. However, due to the training data distribution, there is a bias toward the person class, as can be seen in the confusion matrix as a vertical line through class 22, which is the number of the person class. Over 10% of toothbrushes, handbags and backpacks and over 20% of hair dryers are classified as person. As seen in Figure 7.4a, the images of these classes can have people in them because they are items that are worn, held or used by people. This also occurs between two negative classes, although it is not important for the task of binary classification, since both are interpreted as negative. Another source of errors is labeling noise as seen in Figure 7.4b. Naturally, there are images that are simply classified incorrectly such

Classifications of person images (%))				
person 97.87	chair 0.31	couch 0.16	dining table 0.16	motorcycle 0.10
Images classified as person (% of input class)				
person 97.87	hair dryer 22.22	backpack 14.77	handbag 14.04	toothbrush 13.56

Table 7.3: Top classifications for images with the input or output class person. E.g. 0.31% of person images are classified as chairs and 14.77% of backpacks are classified as person.

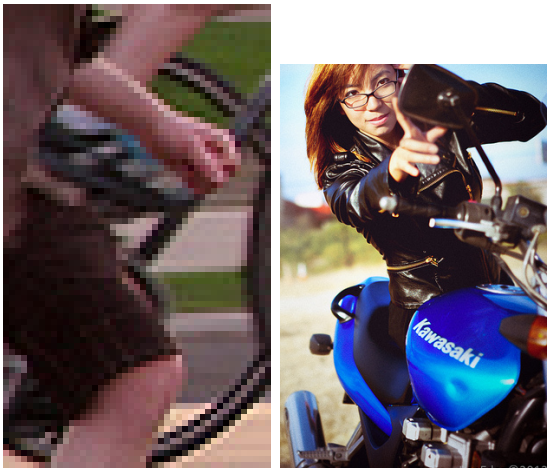
as the images in Figure 7.4c, which shows that the trained classifier is not perfect. A possible explanation for this is the training with data containing labeling noise and the high intraclass variability of persons.

Comparing these results to the results from the previous section emphasizes the importance of data and length of training. Using an additional 26% of data and training for 20 epochs longer results in an accuracy increase of about 1.5%. The binary error is 2.82% compared to the 6.7% of the winner of the classification challenge in ILSVRC2014, i.e. GoogLeNet with 1.2 million images. These errors are only partially comparable, since the ILSVRC error is provided only as a top-5 error, i.e. the ground truth class only has to be among the 5 top classifications, and since there is a difference between 2-class and 1,000-class classification, since random guessing for 2 classes yields 50% but for 1,000 classes only 0.1% accuracy. Nevertheless, our results achieve similarly high accuracies, showing comparable quality of the PersonData dataset, though for a different task with different data.

7.2.2 Generalization to New Data

Evaluating generalization capabilities of the model and dataset is important to show their worth for new applications. However, as any significant dataset is used to create PersonData, it is difficult to find additional test sets. As a result, the datasets Penn-Fudan Database for Pedestrian Detection and Segmentation, CUHK Person Re-Identification Dataset, CUHK campus, CUHK-03 Dataset and two tracking scenes from PETS2009 are chosen for evaluation. However, only positive annotation is provided for these datasets. Therefore, for the first datasets, only positive classifications are evaluated and for the PETS scenes, negative samples are cropped out, as described in the previous chapter. Table 7.4 lists the results of these tests. Note that since the PETS data only has a nonobject negative class, an arbitrary nonperson label is assigned for evaluation purposes using the 80-class model.

Results of over 94% are achieved, which is comparable to the results on the validation and test data and, therefore, underlines the generalization capabilities of the classifier. On Penn-Fudann and CUHK, over 99% accuracy are achieved, which again indicates a



(a) The bicycle and motorcycle are classified as person since the images also contain people.

(b) The image is labeled as banana but is classified as a bowl due to ambiguity.

(c) A cow is incorrectly classified as couch and a pizza as book.

Figure 7.4: Images may be misclassified for different reasons, such as labeling noise and ambiguity, but also because of an imperfect classifier.

bias toward the person class. This is also demonstrated by the recall scores, which are higher than the precision scores, as they contain more false positives than false negatives.

7.2.3 Overfitting to Video Data

PersonData contains person images extracted from tracking datasets, i.e. video data. Even after reducing redundancy by not selecting every frame, multiple images of the same people remain. As a result of the random shuffling of the data in the process of creating training, validation and test set splits, the same person can appear in multiple

dataset	# pos.	# neg.	acc. %	recall %	precision %
Penn-Fudan	423	0	100	100	100
CUHK (combined)	12,614	0	99.97	99.97	100
PETS2009 S2L1V1	4,650	1,239	94.35	99.81	93.47
PETS2009 S2L2V1	10,292	1,239	96.19	99.56	96.30

Table 7.4: Results for generalization to new datasets and the number of positive and negative images per class.

splits. As the person may be trained multiple times in the training set and then also appears in the test set it, is naturally classified more easily. To analyze how much the model overfits to such cases due to video data, the test set is split into two parts, i.e. into images from video data sources and images from other sources. Note that still image datasets also contain multiple images of the same person often without annotation about any source video or image sequence frame number and therefore may overfit similarly. However, analyzing video and still image data separately still gives insights into the strength of overfitting. Figure 7.5 shows examples of the difference in data source types.

Figure 7.5: Examples of person images for datasets consisting of still images (left) or video sequences (right). In video datasets, the same people appear more often and in less distinct pose variation than in still image datasets.

data	# images	acc. %	recall %	precision %
Still images	31926	97.72	97.72	100
Video data	2456	99.83	99.83	100

Table 7.5: Results of the analysis of splitting positive test data by video or still image source dataset.

Table 7.5 shows the results of both tests. Note that these tests were performed using only positive data because the negative class only comes from the MSCOCO dataset. As expected, the accuracy is higher for the video data, with a difference of 2.11%. The

effect on the entire data should be small considering that video data is only about $\frac{1}{12}$ the amount of still image data. The results depend strongly on not using every frame of a video dataset, as increasing the ratio of video to still image data would increase the degree of effect of the overfitting. The results of the previous evaluation, however, show that the model does not generally overfit and that the high accuracy is justified.

7.2.4 Training Set Size

A main hypothesis is that more training data lead to better results, as smaller quantities are overfitted more easily. Four tests are performed using differently sized portions of the training data on an identical validation dataset to demonstrate the importance of the size of PersonData. Table 7.6 shows the results for tests using 25%, 50%, 75% and 100% of the training data with identical training.

% data	# images	multiclass acc. %	acc. %	recall %	precision %
25	137,792	3.09	43.91	0	
50	275,626	73.23	94.02	91.82	97.36
75	413,436	79.76	95.83	94.50	98.00
100	551,286	83.02	96.70	96.14	97.94

Table 7.6: Results for differently sized portions of training data.

The benefit of using more data is clearly visible. The results are plotted in Figure 7.6. Increasing the training data from 25% to 50% results in 70% higher accuracy, but the increase from 50% to 75% only raises accuracy by about 6%; the increase from 75% zo 100% improves accuracy by less than 4%. This shows that for a certain network and data, adding more and more data from the same sources does not improve performance at the same rate. One possible explanation is that data from the same sources covers only one part of the range of intraclass variance and that it is necessary to add qualitative new data, i.e. data containing information about the target class that has not yet been provided by other sources.

7.3 Benefit of Pre-Training with PersonData

The PersonData dataset is designed to aid in training for person-specific tasks such as person classification or detection. We show that using PersonData as the pre-training data set and later fine-tuning it to a specific application achieves better results than simply using the application-specific data alone. Two scenes of the PETS2009 dataset are prepared for image classification, as described in the previous chapter. The resulting two datasets are positive and negative images for person classification from the same camera viewpoint but at different times with different lighting conditions and different numbers of people. Therefore, training a classifier using the first scene and testing it on the second mimics developing a classifier for person classification or detection for surveillance

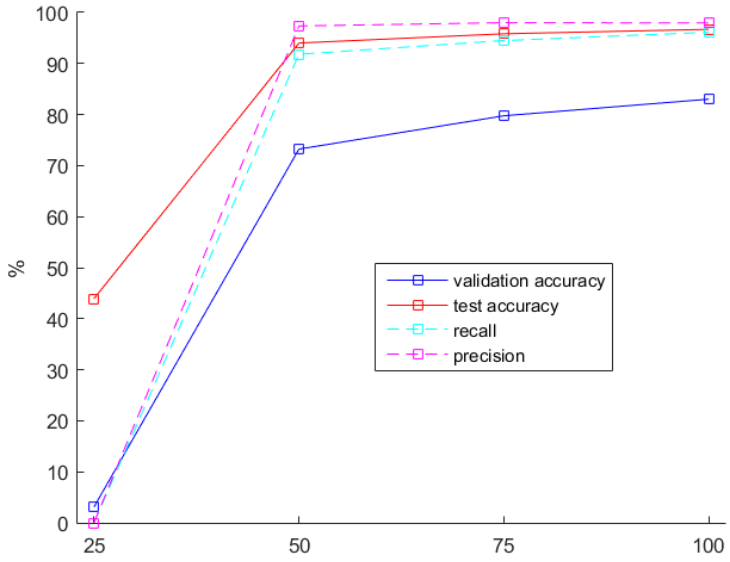


Figure 7.6: Accuracy, recall and precision are plotted over the percentage of data used. More data achieves higher values, which appear to converge for larger amounts.

purposes of a fixed viewpoint with little data. The results listed in Table 7.7 show the performance benefit of pre-training the model with PersonData and then fine-tuning it to PETS data. Fine-tuning was done by initializing the network with the weights learned from PersonData and dividing the learning rate by 10 for all layers below the fully connected layers and by 2 for the fully connected layers of the CaffeNet architecture. The model is trained in each case with 90% of the PETS S2L1V1 data as the training set and 10% as the validation set and tested on the complete PETS S2L2V1 set.

training data	initialization	acc. %	recall %	precision %
PETS	random	91.16	94.46	95.59
PETS	PersonData	99.53	99.61	99.86

Table 7.7: Results on the PETS2009 S2L2V1 dataset after training with PETS2009 S2L1V1 using random weights or weights pre-trained on PersonData.

Pre-training with PersonData results in a performance gain of over 8%, which clearly demonstrates the benefits of pre-training with our data. The pre-trained filters and features are generalizable and lead to better results than random initialization. This result is additionally confirmed by the speed of training. It takes 20 epochs to achieve 100% accuracy on the small validation set of 589 images using random initialization, whereas the fine-tuned network achieves 100% after only 5 epochs.

7.4 Chapter Summary

The results and evaluation of our dataset and classifier are demonstrated in this chapter. First, validation accuracy and test accuracy, recall and precision are shown for different types of subclass labeling, i.e. the 80-class, 12-class and 2-class models with and without MVN layers showing that subclass labeling increases training robustness. The problem of initialization and training the 2-class model is explained. Next, analysis of incorrect classifications and effects of labeling noise are discussed. Then, results are demonstrated regarding generalizability and video data overfitting of the network trained on PersonData showing the quality of the dataset. Afterwards, the amount of data required for training is analyzed indicating the importance of large-scale datasets. Lastly, the benefits of pre-training datasets of only a few thousand images with PersonData is demonstrated.

CHAPTER 8

Conclusion

In conclusion our contributions are threefold. First, we created a large-scale dataset for person recognition with over 600,000 images for person classification. Data was collected from over 30 datasets and a method of viewing such heterogeneous data sources was proposed with respect to their annotation structure, format and type to be able to extract data as available and required for a specific application. Second, we trained a CNN using this data for binary person classification, i.e. classifying the existence of a person in an image as true or negative, with an error rate of less than 3%. Third, we demonstrated that subclass labeling aids in training the classifier more robustly and avoiding problems of solver choice and initialization by having stronger labeling of the catch-all negative class. We additionally evaluated the benefit of the data as a pre-training training set for fine-tuned applications.

8.1 Summary

To summarize, we first explained what CNNs are, beginning with the perceptron and extending to neural networks and finally convolutional networks. Next, an overview on person recognition was given. Afterwards, we discussed the related work. Specifically, examples were brought for several CNN architectures and techniques such as dropout or batch normalization, and recurrent CNNs and fully convolutional networks were explained. Next, CNNs for object and pedestrian detection were introduced along with the RCNN pipeline, which is further extended with RPNs for fast and accurate object detection. Then, benchmarks and datasets with large amounts of labeled data for comparison of state-of-the-art methods like the ILSVRC and MSCOCO were described. Also, visualization techniques and analysis of CNNs and additional applications were discussed.

Afterwards, we explained in detail how the PersonData dataset pool was created from 31 source datasets. Since these datasets vary strongly with respect to annotation and content,

we introduced a two-part method of viewing the dataset by extracting information from individual, different datasets and then using this data in a second step. In our case, we extracted over 300,000 labeled person image crops of person classification and used the MSCOCO nonperson classes to create a classification dataset of over 600,000 images, where the negative class is defined by 79 labeled subclasses having additional category labels for 11 supercategories.

From this dataset, we trained a person classifier using the CaffeNet architecture and experimented with the model, training data and labeling. We tested methods for generating region proposals on feature maps for object localization and discussed that only approximate location can be computed. Next, we evaluated the strengths and weaknesses of the trained classifier and showed that stronger labeling is beneficial for training, since training works more robustly with regard to the solver choice and parameter initialization. Moreover, we showed that the classifier trained on PersonData generalizes well to new data and discussed the problem of overfitting to video data as well as the benefit of using larger amounts of data. Finally, we demonstrated the benefit of using PersonData as a pre-training dataset for applications.

8.2 Discussion

Our results show that the choice of CNN architecture is complex. Adding the three MVN layers, which are essentially intended to aid in training by not making neurons adapt to varying distributions, helped to achieve higher accuracy in the 2-class model, whereas the 12-class and 80-class models experienced significant drops when using 80% as the training data. Therefore, adding additional MVN layers does not generally improve the model. This can be further analyzed by comparing it with batch normalization layers, which differ from MVN layers in that they compute moving averages and variances at training time and use these at test time.

For training, using the additional labels in the 12-class and 80-class models compared to the binary approach had the advantage of not having to worry about the initialization and solver, unlike in the case of the 2-class model. Therefore, we hypothesize that supplementary subclass labels make training more robust. The exact amount or depth of labeling is a hyperparameter that has to be examined using a validation set. In our case, 12 classes lead to higher accuracy than 80 classes without MVN layers but to a lower accuracy using MVN layers.

The highest achieved accuracy of 97.18% indicates that PersonData is a dataset where the person class is well-represented and intra-class variances are covered well and that the model used is sufficient to train a classifier on this data. This 2.82% error is smaller than the winning top-5 error of the ILSVRC2014 1,000-class classification challenge. The generalizability to new datasets also underlines the quality of the PersonData set. We showed that the accuracy converges for the amount of data in our dataset for the selected architecture. This leads us to believe that further data needs to add to the total quality by, for example, covering more intraclass variances rather than only adding to quantity.

Another possibility to improve the data is to manually prune noisy data caused by the existence of labeling noise. Our results have shown misclassifications caused by labeling noise, which emphasizes the importance of data checks. For completely new data, this can be done by exploiting redundancy, e.g. having two people label the same image and when there are differences use annotations provided by a third person to resolve issues.

In retrospect, adding the datasets used for evaluation purposes and using gained knowledge about the data sources already used, it is possible to extract an unknown amount of additional data from the dataset pool, e.g. instead of searching the ImageNet synsets directly, images can be used from the detection challenge, where a person class is provided. Although this additional data mainly adds to the quantity of data, it should also improve quality, since it is likely that the data covers more intraclass variances, given the large heterogeneity of the person person class resulting from different poses, clothing, ethnicity, etc. Also, the bias toward the person class can be analyzed further by choosing a subset of the entire data and altering the distribution to find if distribution is indeed a cause for biases or if different negative data is needed. Generally, we suggest that the data distribution be the same as the expected distribution at test time, i.e. that the same distribution be used as occurs in applications.

Our classifier is naively extendable to a sliding-window detector. However, we conclude from our localization experiments and the literature [OBLS15, RHGS15] that specifically designed detectors exploit advantages of shared computation and that features learned to understand object extents, whereas CNNs for classification learn to recognize features important only for the existence of objects. The PersonData dataset can be used for detection training as well. Data from detection or tracking sources is directly usable but classification data can also be utilized to create a larger dataset. Classification datasets in our pool mostly have fixed margins around the objects in their images. This fixed margin is usable to automatically define a bounding box for each image and even if no margin is annotated, the image can be used for detection training by defining a whole-image bounding box.

8.3 Future Work

As ImageNet is commonly referenced as a pre-training dataset [RHGS15], possible future work includes comparison of pre-training data with ImageNet or with PersonData for tasks containing people, e.g. detection. Since low-level features are equivalent among different architectures [LV15a], we suggest that the following multistage fine-tuning could be beneficial. First, the model is trained with ImageNet and learns highly robust low-level features simply due to the immense amount of data. Second, the model is then trained with PersonData to learn mid-level features for person recognition, for example, by limiting or disabling learning rates for low-level features. Lastly, the model is fine-tuned for an application, where application-specific high-level features are learned.

As labeling noise is apparent in PersonData, instead of exhaustively, manually pruning the data, it is also possible to adapt the model or the training to explicitly handle it

similarly to training with noisy web images [VGLBP15]. Semisupervised methods can be used to identify noisy labels and adapt them. Additionally, prior information about data source that have more labeling noise than others can be utilizedto identify and adapt noisy labels.

Another way of improving classifier performance is, for example, hyperparameter tuning, i.e. finding the best CNN architecture with layer sizes and properties, and finding the best amount of video redundancy from tracking sources. The latter can, e.g., be done by using image similarity measures to select distinct images from a video sequence. Furthermore, additional data augmentation can be exploited during training but also during testing.

Further evaluation can be performed by taking the negative or background class into consideration. The necessity and benefit of separate objects, an additional catch-all class, randomly generated data, and object-less data, i.e. streets, walls, floors, etc., can be analyzed.

Finally, PersonData can be used outside of the scope of CNNs. For example, deep forests or deep belief networks can be trained using our data.

Bibliography

[AJM15] E. Ahmed, M. Jones, and T.K. Marks. An improved deep learning architecture for person re-identification. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3908–3916, June 2015.

[AKV15] A. Angelova, A. Krizhevsky, and V. Vanhoucke. Pedestrian detection with a large-field-of-view deep network. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 704–711, May 2015.

[ARS11] A. Andriyenko, S. Roth, and K. Schindler. An analytical formulation of global occlusion reasoning for multi-target tracking. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 1839–1846, Nov 2011.

[BSCL14] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky. *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, chapter Neural Codes for Image Retrieval, pages 584–599. Springer International Publishing, Cham, 2014.

[CRM03] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):564–577, May 2003.

[Cyb89] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[DDS+09] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, June 2009.

[DSH13] G.E. Dahl, T.N. Sainath, and G.E. Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8609–8613, May 2013.

[DT05] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893 vol. 1, June 2005.

[DTS06] N. Dalal, B. Triggs, and C. Schmid. Human detection using oriented histograms of flow and appearance. In *Computer Vision – ECCV 2006*, pages 428–441. Springer, 2006.

[DWSP12] P. Dollar, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(4):743–761, April 2012.

[FFFP07] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59 – 70, 2007. Special issue on Generative Model Based Vision.

[Gav07] D.M. Gavrila. A bayesian, exemplar-based approach to hierarchical shape matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(8):1408–1421, Aug 2007.

[GDDM14] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, June 2014.

[GEB15] L.A. Gatys, A.S. Ecker, and M. Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[Gir15] R. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[GMB13] I.J: Goodfellow, D. Warde-Farley M. Mirza, and A. Courville Y. Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[HZRS14] K. He, X. Zhang, S. Ren, and J. Sun. *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part III*, chapter Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition, pages 346–361. Springer International Publishing, Cham, 2014.

[IS15] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[JSD+14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 675–678, New York, NY, USA, 2014. ACM.

[Kar16] A. Karpathy. CS231n Convolutional Neural Networks for Visual Recognition. `http://cs231n.github.io/neural-networks-1/`, 2016. [Online; accessed 29.01.2016].

[KSH12] A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[LBD+89] Y. LeCun, B. Boser, J.S. Denker, D. Henderson R.E. Howard, W. Hubbard, and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[LH15] M. Liang and X. Hu. Recurrent convolutional neural network for object recognition. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3367–3375, June 2015.

[LHYS15] F. Liu, Y. Huang, W. Yang, and C. Sun. High-level spatial modeling in convolutional neural network with application to pedestrian detection. In *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 778–783, May 2015.

[LKF10] Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 253–256, May 2010.

[LMB+14] T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C.L. Zitnick. *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, chapter Microsoft COCO: Common Objects in Context, pages 740–755. Springer International Publishing, Cham, 2014.

[Low04] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[LSD15] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, June 2015.

[LV15a] K. Lenc and A. Vedaldi. R-CNN minus R. *CoRR*, abs/1506.06981, 2015.

[LV15b] K. Lenc and A. Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 991–999, June 2015.

[MG06] S. Munder and D.M. Gavrila. An experimental study on pedestrian classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1863–1868, Nov 2006.

[MV15] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5188–5196, June 2015.

[MYL+08] Y. Mu, S. Yan, Y. Liu, T. Huang, and B. Zhou. Discriminative local binary patterns for human detection in personal album. In *2008 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2008.

[NH10] V. Nair and G.E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[NYC15] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436, June 2015.

[OBLS15] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Is object localization for free? - weakly-supervised learning with convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 685–694, June 2015.

[OPS+97] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio. Pedestrian detection using wavelet templates. In *1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 193–199, Jun 1997.

[OWZ+15] W. Ouyang, X. Wang, X. Zeng, S. Qiu, P. Luo, Y. Tian, H. Li, S. Yang, Z. Wang, C. Loy, and X. Tang. Deepid-net: Deformable deep convolutional neural networks for object detection. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2403–2412, June 2015.

[PRH+14] M. Paulin, J. Revaud, Z. Harchaoui, F. Perronnin, and C. Schmid. Transformation pursuit for image classification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3646–3653, June 2014.

[RDS+15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[RHGS15] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[RHW88] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Cognitive modelling*, 5:3, 1988.

[Ros58] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[SA15] R. Stewart and M. Andriluka. End-to-end people detection in crowded scenes. *CoRR*, abs/1506.04878, 2015.

[SDBR14] J.T. Springenberg, A. Dosovitskiy, T. Brox, and M.A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[SEZ+13] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013.

[SHK+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[SKCL13] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[SLJ+15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.

[SZ14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[TS85] T. Tsukiyama and Y. Shirai. Detection of the movements of persons from a sparse sequence of tv images. *Pattern Recognition*, 18(3–4):207 – 213, 1985.

[VGLBP15] P.D. Vo, A. Ginsca, H. Le Borgne, and A. Popescu. Effective training of convolutional networks using noisy web images. In *2015 13th International Workshop on Content-Based Multimedia Indexing (CBMI)*, pages 1–6, June 2015.

[WMSS10] S. Walk, N. Majer, K. Schindler, and B. Schiele. New features and insights for pedestrian detection. In *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1030–1037, June 2010.

[WNL08] B. Wu, R. Nevatia, and Y. Li. Segmentation of multiple, partially occluded objects by grouping, merging, assigning part detection responses. In *2008 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2008.

[WY06] Y. Wu and T. Yu. A field model for human detection and tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(5):753–765, May 2006.

[YCBL14] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3320–3328. Curran Associates, Inc., 2014.

[YCN+15] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579, 2015.

[YLL10] F. Yang, Y. Lu, and B. Li. Human body detection using multi-scale shape contexts. In *2010 4th International Conference on Bioinformatics and Biomedical Engineering (iCBBE)*, pages 1–4, June 2010.

[ZBS15] S. Zhang, R. Benenson, and B. Schiele. Filtered channel features for pedestrian detection. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1751–1760, June 2015.

[ZC12] G. Zheng and Y. Chen. A review on vision-based pedestrian detection. In *2012 IEEE Global High Tech Congress on Electronics (GHTCE)*, pages 49–54, Nov 2012.

[ZF14] M.D. Zeiler and R. Fergus. *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, chapter Visualizing and Understanding Convolutional Networks, pages 818–833. Springer International Publishing, Cham, 2014.

[ZLX+14] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 487–495. Curran Associates, Inc., 2014.