One-shot Detektion in kunsthistorischen Bildern

One-shot detection in art historical images

Maurice Frank

May 18, 2017

Ruprechts-Karls-Universität Heidelberg

Fakultät für Mathematik und Informatik
Heidelberg Collaboratory for Image Processing
Computer Vision Research Group

Bachelorarbeit

One-shot Detektion in kunsthistorischen Bildern

One-shot detection in art historical images

Maurice Frank
3174235

Reviewer Prof. Dr. Björn Ommer
Fakultät für Mathematik und Informatik
Ruprechts-Karls-Universität Heidelberg

Supervisors Dr. Miguel Ángel Bautista

May 18, 2017

Maurice Frank
One-shot detection in art historical images
One-shot Detektion in kunsthistorischen Bildern
Bachelorarbeit, May 18, 2017
Reviewers: Prof. Dr. Björn Ommer
Supervisors: Dr. Miguel Ángel Bautista

Ruprechts-Karls-Universität Heidelberg
Computer Vision Research Group
Heidelberg Collaboratory for Image Processing
Fakultät für Mathematik und Informatik
Berliner Str. 43
69120 and Heidelberg

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, May 18, 2017

Maurice Frank

Abstract

This thesis introduces a one-shot object detector that was developed with the objective to retrieve similar objects from images given a single reference object. We show that using a neural network which was pretrained for image classification, it is possible to train a object detector from small numbers of examples in very little time. Our results show the connection and correlation between the number of examples and the quality of the detector. Furthermore we show how, using intensive preprocessing, a single example can be enough to extract similar instances.

We outperform a simple classical object detector on the PASCAL-VOC dataset detecting parts of different objects with each a two-class Fully Convolutional network detector.
Lastly we show qualitatively how this method can be applied in the humanities specifically art history to automatically discover other depictions of the same object.

Zusammenfassung

Diese Abschlussarbeit führt einen Objektdetektor ein, der mit dem Ziel entwickelt wurde ähnliche Objekte in Bildern zu finden mit einer einzelnen vorgegebenen Referenz. Wir zeigen, dass es unter der Verwendung eines für Bildklassifikation trainierten Neuronalen Netzwerks möglich ist, einen Detekor auf Basis weniger Beispiele in kürzester Zeit zu trainiern. Unsere Resultate verdeutlichen den Zusammenhang, zwischen der Anzahl der verschiedenen Beispiele und der Qualität des Detektors. Weiterhin wird gezeigt, wie mit umfangreicher Vorverarbeitung der Referenzbeispiele schon ein einzelnes Bild reichen kann, um ähnliche Instanzen zu finden.

Unsere Method liefert bessere Ergebnisse auf dem PASCAL-VOC Datenset als ein einfacher klassicher Objektdetektor unter der Aufgabe verschiedene Teile von verschiedenen Objekten zu detektieren. Wir erreichen dies jeweils mit einem zwei-Klassen Fully Convolutional Network. Zuletzt zeigen wir qualitativ, wie unsere Methode in den Geisteswissenschaften, speziell der Kunstgeschichte, verwendet werden kann, um automatisch Darstellungen eines bestimmten Objektes in Bilddatenbanken zu finden.

Contents

1 Introduction

Today more and more work processes can be supported, or replaced through intelligent machines. Computer Vision proves itself capable of understanding images and video on a near-human level or even outperforming normal human capabilities in some visual tasks. One reason intelligent machines are absolutely needed today and in the future are the enormous masses of data the *digital natives*, companies or scientists generate and store. Independent of the performance difference between computer and human, it its often impossible to work through catalogs or databases of millions or more images, music recordings, scientific data records et cetera. To automate such tasks does not only relieve from monotonous work but also enables humans to achieve more in less time.
At the very latest with the development of analog and digital mass storage systems scientists began to base their research on findings from mass data. No different scientists of the humanities move their work into a digital form. Being able to quickly look up any painting as a digital image or shifting through unique, in some archive locked, manuscripts from the middle ages makes drawing connections and research considerable easier. As we can assume these databases to grow or already have grown over any humans capabilities, digital humanities are a prime example for the application of visual learning and automation.

1.1 Problem Statement

For this thesis we approach one of possible applications of Computer Vision in the digital humanities, in particular art history. We assume a historian doing work on a huge set of digitized images $\mathbb{I}$. Those could be scans of medieval writings, architecture sketches or photos of historic paintings. The images are assumed to be not at all annotated meaning we do not have any metadata containing e.g. the creator.

Now the historian wants to compare all images $I = \{i_1, i_2, \ldots, i_n\} \subset \mathbb{I}$ conjunct in-that they all share some visual pattern p, may it be they depict the same person or show all a similar scenery. We want to provide the user of this database a method to automatically generate the set I. This should be done giving a image τ which the historian states for, that it contains p. Therefore we need a function $C(i)$ which can

decide for a image $i \in \mathbb{I}$ whether it possesses p and therefore is part of the seeked set I. The only data we can use to adjust the operator C is the sample image τ. Proposing one approach for this C_τ and a generator function $G(\tau) = C_\tau$ is the problem of this thesis.

1.2 Thesis Structure

The thesis is structured as follows. Firstly in **Section 2** we give a quick overview over the topics and techniques used in this work including a short introduction to the inner workings of Convolutional Neural Networks (CNNs). In **Section 3** we list related works that tackle different subtasks of our method. Next in **Section 4** we explain in depth our pipeline and report extensive test results in **Section 5**. Finally in **Section 6** we show how this method can be practically applied in the real world and give a short conclusion and outlook in **Section 7**.

2 Concepts

> *Q: What's the difference between machine learning and statistics?*
> *A: ConvNets*
>
> **— Bored Yann LeCun**
> via Twitter

In this chapter we quickly introduce the core techniques and concepts this thesis is build upon. First we discriminate four different tasks of describing objects in images in 2.1. Next in 2.2 we give overview over Neural Networks and what kind of computations they perform. Further we explain two specific architecture called Fully Convolutional Network (FCN) and Residual Network (ResNet) in 2.3 and 2.4, respectively. The last section 2.5 describes two methods to evaluate our approach.

2.1 Task definitions

Although the posed task for this thesis is pretty clear from its formulation, we want to quickly distinguish the similar but different image understanding tasks that are solved using these techniques.

The simplest task that was tackled is **image classification**. In image classification the input are images I containing a single object that fills most of the image. Hence the task is to find the class label c for the singular object o which therefore labels the whole image: $f(I) = c_o$. Since 2015 CNNs can surpass the performance of humans doing classification [He+15].

Next **object localization** was addressed. Localization means that we still have only one relevant object per image I but here the object fills only a part of the image. Instead of only predicting the class c for this object o the localizer also has to predict translation t and scaling s for the object: $f(I) = (c, t, s)_o$.

Harder to solve but also way more applicable is **object detection**. Now we take complicated images I, containing a variable number of possibly overlapping objects

$O = \{o_1, o_2, \ldots, o_n\}$, each of varying class c. The detector has to identify and localize all objects inside the image: $f(I) = \{(c,t,s)_1, (c,t,s)_2, \ldots, (c,t,s)_n\}$.

Lastly we have **image segmentation**. Here we aim to not only identify multiple different objects o in an image I but also to find their boundaries on the pixel level: f(I) =$\{c_i \; \forall i \in I\}$. As such each pixel of a found object in the image is labeled with its class label resulting in multiple binary masks each masking out one individual object.

2.2 Neural Networks

Neural networks are a specific method to approximate a function $f(x) = y$, which yields some desired output from input data. For example this could be the task to return the name of the breed given images of dogs as input. They are called networks because they are a composite of chained functions, each computing one step of the overall function. Data flows from the top, the first computation, to the end getting further processed:

$$f(x) = f_n(f_{n-1}(\ldots f_1(x))) = y \tag{2.1}$$

When the network is connected on a acyclic graph, meaning the flowing data does not go through any loops, it is called a **feedforward network**. The nodes in this graph, representing the functions f_i are also called neurons showing the connection to the human brain on which neural networks are loosely based. Nodes, like neurons, can have different levels of connectivity. While we focus here on feedforward networks, not each deeper node has to be connected to all nodes sitting on top of it. Thereby these networks can have multiple branches with input data being split and flowing through different function chains until being gathered in a single output.

In modern neural networks the f_i are mostly linear operators but are, most of the time, incorporated with non-linear processing operators. Working on raw pixel values of images, it will, in the most cases, not be possible to find a linear decision function which separates inputs according to the task [LBH15]. To discriminate such representations we need a non-linear decision boundary and thus a neural network should be a composite of linear and non-linear operations, so that it, at least in theory, can approximate the non-linear representation.
The most classical operation for f_i inside a neural network are functions returning a linear combination of the inputs. Nodes using such a function are called Fully Connected (FC) layers. The FC layer is connected to all nodes from the next upper

layer of the network. If we assemble all outputs from the previous nodes in a single matrix X as the input for the layer, its function

$$f_{fc}(X) = XW + b \tag{2.2}$$

can simply be expressed as a matrix multiplication with a weight matrix W and a bias vector b.

2.2.1 Optimization with Stochastic Gradient descent

We give a quick overview how neural networks are trained.
At the end of a neural network, in training, are normally a score function S and a loss function L. The score function assigns the input to a semantic label, defined for a specific task. The loss function tells how good these assignments are, thus rating the current data pass through the network. We want to modify the parameters of our network to a point where the loss is minimal. The process of finding these parameters is called optimization.
The basic idea for optimization is to gradually, step by step, approach this optimal parameter set. While the optimizer does that, it moves over the function space of the loss function L to find one of its minimums. To reach a local minimum of L we search for the steepest direction at the current point and follow it done. One method to iteratively update parameters according to their loss is Stochastic Gradient Descent (SGD). Each parameter θ follows the negative direction of the gradient $\Delta\theta = {}^{\delta L}/_{\delta\theta}$ of the loss function:

$$\theta = \theta - \lambda \cdot \Delta\theta \tag{2.3}$$

The factor λ setting the length of each step down the gradient is called the learning rate. It has to been set before training and can strongly influences training results.

A popular extension for Stochastic Gradient Descent (SGD) is momentum [Qia99]. With this method we remember a proportion m of the last update vector v_t and uses this memory to push the SGD faster into a minimum.

$$\begin{aligned} v_t &= m \cdot v_{t-1} + \lambda \cdot \Delta\theta \\ \theta &= \theta - v_t \end{aligned} \tag{2.4}$$

Fig. 2.1.: The effect of a convolution and a following ReLU visualized. The green activations are positive and the purple ones are negative.

2.2.2 Backpropagation

We want to use SGD to optimize the many parameters inside a neural network. The standard approach for this is called **backpropagation** [RHW88]. The aim of backpropagation is to find the partial derivative of the error function with respect to each parameter of the network. Backpropagation repeatedly applies the chain rule to derivatives to all possible paths through our network, starting from the bottom error function. For a composite function $F(x, y) = f(g(x, y))$ we can compute the partial derivative $\frac{\delta f}{\delta x} = \frac{\delta f}{\delta g}\frac{\delta g}{\delta x}$ using the chain rule. In a feedforward network we now have our intermediate node using some operator. While a forward pass they can compute their local gradients independently of any surrounding nodes. While the backward pass all nodes send their gradients to all their inputs starting with the tailing output layer. Each node will receive the gradient from its deeper output and simply multiply it to its already computed local gradient, following the chain rule. With this propagation we can provide gradients for all parameters without having to reason about the composite function on the whole.
With the propagated gradients we can perform SGD optimization on all network parameters and thus can learn the neural network.

2.2.3 Convolution

Convolution is a special kind of linear operation, that is successfully applied in deep neural networks, which we then call a Convolutional Neural Network (CNN). For two one-dimensional continuous signals expressed by the functions f and g their convolution is described as follows:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau \tag{2.5}$$

Here the function g is translated through the functions domain and we integrate over the piecewise multiplication of the shifting g and the stationary f. Because we

work on digital discretized data (*e.g. images*) we need the discretized convolution which in $1d$ looks like this:

$$(I * K)(t) = \sum_{\tau=-\infty}^{\infty} I(\tau)K(t-\tau) \tag{2.6}$$

Again we see how K slides over the domain of I and K, and is multiplicated at each step with I.
We want to use convolutions to examine images. While images could be flatted into a $1d$ representation that would drop $2d$ visual information. As such we want to use convolutions on $2d$ signals:

$$(I * K)(x, y) = \sum_{i} \sum_{j} I(x, y)K(x-i, y-j) \tag{2.7}$$

I is the input, an image, and K is called the **kernel**. The kernel is shifted over both axes of the image and at each position we compute the multiplication between the kernel and the respective part of the image. For example K could be of size 3×3 pixels and for an arbitrarily sized image I, the convolution $I * K$ would return all results of the multiplication between K and each 3×3 pixel sized window of the image, around each pixel $i(x, y)$ inside I. The part of the input image that the kernel is connected to at one computation is called its **receptive field**. The result can be seen as a map, same-sized to the input, describing how much the input responded to the kernel at each point, thus it often is called **response map**.

The first advantage in the use of kernels is that they can remain very small (like 3×3 pixels or 7×7 pixels) but return outputs for the way bigger input and therefore need few parameters to operate. Secondly convolutions are equivariant to translations which means that if we translate the input somehow, the response, at the shifted position in the response map, is still the same as before. This is really important because then the convolutions respond to signals independent of their position in the input image.Lastly convolutions can be described with a value restricted matrix multiplication, in the form of a Toeplitz matrix, and therefore the discretized convolution is a linear operation. Using many optimizations from linear algebra convolutions can be computed immensely fast.
Inside CNNs we use convolutions to map, how much a specific small structure is present at each position inside a image. An example of this is shown in Figure 2.2.

2.2.4 Activation Functions

Conv layers return the response of their inputs to their filters. Next the response level is used to calculate how much the node in the network fires for this position. Conceptually this draws again from the activation process inside neurons in

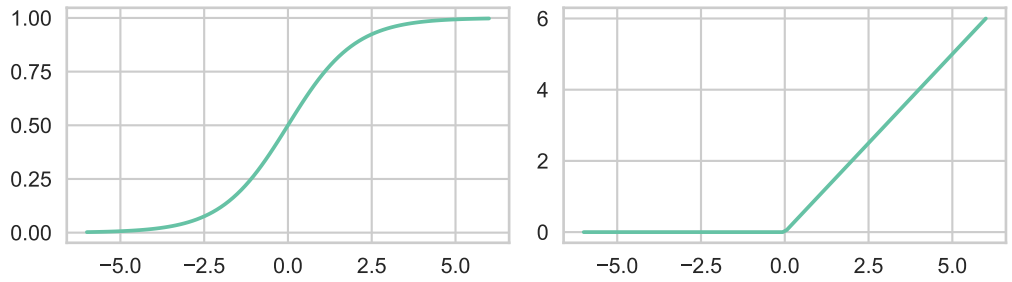


Fig. 2.2.: Activation functions: sigmoid on the left and rectifier from a ReLU on the right.

the brain, which also receive inputs from their dendrites and fire along their axon. Inside an artificial neural network this mapping is done by an activation function $f(x)$. In the most simple cases this can be the identity $f(x) = x$ or the unit step function $f(x) = (\operatorname{sign} x + 1)/2$. With the identity we just use the response values as firing rates not introducing any more complexity and the sign reduces the levels of response to a binary firing switch.

Formerly common to use as an activation function was the sigmoid. The sigmoid $\sigma(x) = 1/(1 + e^{-x})$ (see Fig. 2.2) takes an unbounded input range and squashes it into the range $[0, 1]$. Today sigmoids are mostly avoided as they hold multiple drawbacks. First as the function values saturate in both directions its gradient can approximate zero, which with backpropagation can lead to zeroed gradients or exploding gradients while training. Secondly the gradient computation is slow due to its non-linear form.

One popular function to hold those properties is the rectifier (see Fig. 2.2)

$$\rho(x) = max(0, x) \tag{2.8}$$

employed in ReLUs. While being piecewise linear, and thus still computationally efficient, the rectifier is a non-linear function and can fold the decision space into a non-linear form. Because it does not saturate like the sigmoid it avoids the problems of zeroed or exploding gradients.

2.2.5 Pooling

A typical building block for CNNs is the triple of a Conv layer, followed by a non-linear activation function, followed by a pooling layer. The pooling layer takes its input from the previous layer and compresses it spatially into into a smaller form. Different functions can be applied to the data neighborhood from the input. Average pooling [LeC+90] or max pooling [ZC88] are two of the most prominent ones.

With average pooling we return the average of the values from the neighborhood and with max pooling we return the maximum value from the neighborhood.

It is important to note that average pooling is a linear function while max pooling is not. Pooling can be used to introduce more non-linearity and that is one reason while max pooling is more popular to use. Also as downsampling of the deep filters can be done with strided convolutions alone, pooling is not necessary needed for this and more and more network architectures drop the pooling layers entirely [Spr+14].

2.2.6 Batch Normalization

Batch normalization [IS15] is a method to ease the computation of gradient descent in backpropagation and also to reduce problems of bad weight initialization. Inside training of a deep network emerges the problem of internal covariate shift. Each deeper layer of the network depends on the outputs from its previous layer. The first layer learns on top of the data input, but the second layer learns from the output of the first layers features, and the third layer depends on the second one and so on. In training all these features change through backpropagation and the distribution of all activations change and the deeper the layer is, the longer the chain of evolving distributions in front of it becomes.
If the features in training shift over time the gradient decent becomes harder as each layer has to follow this covariate shift of its successor. BatchNorm helps to avoid this difficulty by normalizing mini batches of inputs. The layer is added after a convolutional layer (or any other linear layer) and collects a batch of output samples $X = \{x_1, x_2, x_3, \ldots, x_n\}$. Then it calculates the mean μ_X and the variance σ_X^2 for this mini batch. Normalizing all the values from that batch with

$$\hat{x_i} = \frac{x_i - \mu_X}{\sqrt{\sigma_X^2 + \epsilon}} \tag{2.9}$$

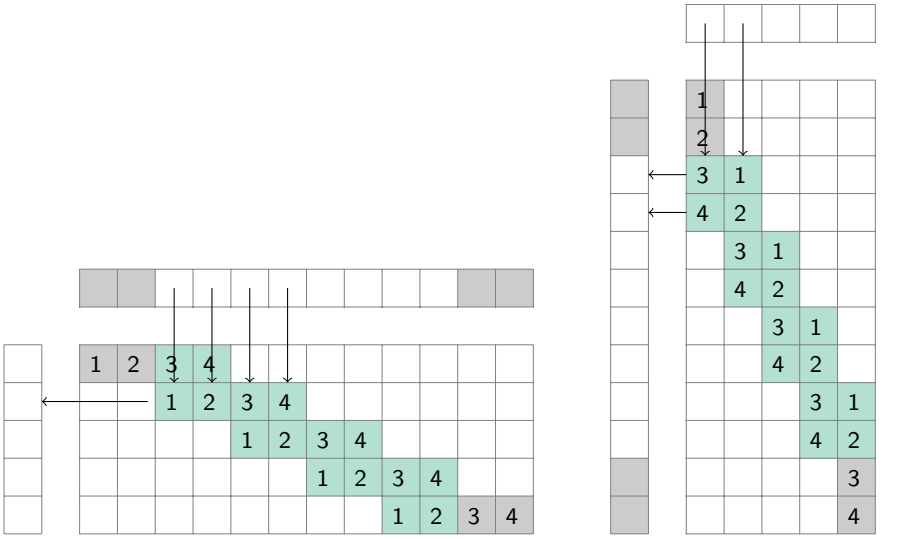
(ϵ is added to avoid divisions by zero) yields the output $\{\hat{x_1}, \ldots, \hat{x_n}\}$.
To use the BatchNorm parameters while testing, we compute a moving average over the means and variances of the batches while training. The forward pass uses these averages to normalize the test inputs.

2.3 Fully Convolutional Networks

Proposed by Long et al. [LSD15], Fully Convolutional Networks (FCNs) are a specific architecture of CNNs. Fully convolutional states that the network consists only of convolutional layers. As these are translation invariant the networks can take arbitrarily sized inputs and produce an dependently sized coarse output.

The original authors use this architecture for image segmentation. They build their network from classic semantic object classifiers like VGG-16 [SZ14]. These take fixed sized images as inputs and produce a single label thereby classifying the whole image. Typically these networks produce high level features through a series of convolutional layers and classify them with a single or a series of FC layers. To label an image pixel-wise with such an architecture we would have to forward the neighborhood of each pixel through the network which would introduce an $\mathcal{O}(w*h)$ time complexity (w and h being the width and the height of the image).

2.3.1 Transposed Convolution



(a) Convolution (b) Transposed Convolution

Fig. 2.3.: Convolution with stride 2 in 1D and transposed convolution with stride 2. Input signals are at the top, output is to the right and the gray boxes are padding. With the arrows we note how input points contribute to the output. The convolution applies a filter of size 4 to an signal of size 8 and with the padding produces an output of size 5. The transposed convolution in return takes a input of 5 and returns a signal of size 10. Figure is based on [Shi+16].

Transposed convolution or sometimes mistakenly called deconvolution, first used by Zeiler et al. [Zei+10], introduces a new operation for neural networks. Simply spoken a transposed convolution takes the receptive field and produces a larger

window as output. Figure 2.3 visualizes the differences between convolution and transposed convolution in $1d$. Both use a sliding kernel but the transposed convolution inverts the forward and backward passes. As such a transposed convolution is actually the gradient of some convolution [DV16].

2.3.2 Conversion of FC layers

FC layers can directly be converted into Conv layers, which makes FCNs immensely efficient. To understand this process let us examine the VGG-16 network. After the 13 Conv layers and 5 corresponding max-pooling layers we get an output from `pool5` with $512 \times 7 \times 7$ pixels. Through pooling the original images (224×224 pixels) got reduced to 7×7 pixels images over 512 channels. These go though 3 FC layers. The first FC layer returns 4096 values. We replace it with a Conv layer with kernel size 7 and 4096 channels thus also resulting in an $1 \times 1 \times 4096$ pixels output. The weights from the FC layer can be reshaped into the size of the Conv layer. The other two FC layers with 4096 and 1000 outputs, respectively can directly be put into convolutions with kernel size 1.

For the former images of 224×224 pixels nothing changed as the output of the first FC layer is still 4096 values but if we now increase the data size to e.g. the double 448×448 pixels, the advantage becomes apparent. The `pool5` will return outputs of size 14×14 pixels and the new Conv layer will slide over the bigger images returning outputs of size 8×8 pixels. Lastly the other 1×1 pixel-sized convolutions do not change the size so that we get 64, spatially located, predictions over the 1000 classes instead of one. These are the same as if we would have slided a window over the original image with an 8×8 grid but because the computations on the GPU are shared for the convolutional kernels we heavily increased performance compared to the naïve solution.

2.3.3 Upsampling through Transposed Convolutions

A FCN as in [LSD15] outputs a small sparse score map. For an input image of 200×200 pixels, that could be something around 10×10 values. Naturally we want an output map with the same size as the input. Firstly we can use feature maps from earlier layers with their own classifiers to reduce the stride of the resulting score map but the score map from deeper layers has still to be upsampled to the input size. Upsampling can be done directly within the network using transposed convolutions. It is possible to use transposed convolutions to upsample the inputs and filter them at the same time but this often introduces artifacts [ODO16]. Alternatively refining e.g. bilinear kernels in training does not help much in performance

[SLD16]. It seems successful to upsample the score maps with a normal interpolation followed by normal convolutions at the bigger size [Don+16].
While bringing upsampling inside the network proves to be computationally efficient it, depending on the library used, can introduce significant memory costs on the GPU.

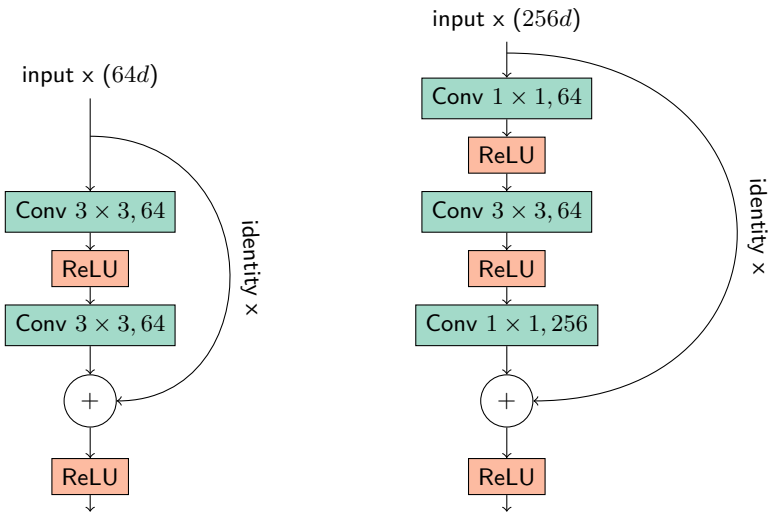
2.4 Residual Networks

After the success of AlexNet [KSH12] and with the possibilities of fast GPU computing, neural networks got deeper and bigger. The VGG-16 by Simonyan and Zisserman [SZ14] has 442.5 million trainable parameters for its 16 layers. The more parameters a network has the harder the training becomes. For the last years multiple new network architectures have been developed that achieve higher accuracy needing less parameters and therefore shorter inference time [CPC16]. One of those architectures are Residual Networks (ResNets) by He et al. [He+16] with which they won the COCO Lin et al. [Lin+14] and the ImageNet Russakovsky et al. [Rus+15] challenges in 2015. A Conv layer in a standard feed forward network learns a mapping $\mathcal{H}(x)$ from its input x. When training the network we reach the point at which the accuracy saturates. At this point the layers which are still being trained should approximate an identity mapping of their inputs as the optimum has already been reached. In practice with deep networks, new layers, at some point in training, rapidly move away from the identity they should learn and degrade the training error after it already saturated. Thus making these networks deeper will worsen their results [HS15].

The main idea of ResNets is pretty simple. They are built from many small identical blocks (see Fig. 2.4). Instead of blocks that learn the mapping $\mathcal{H}(x)$ we introduce blocks that learn the *residual* $\mathcal{F}(x) = \mathcal{H}(x) - x$ of this mapping. For that we add skip connections between the top and the bottom of these blocks, so that the input can be added to the residual, that is the output of the Conv layers. Learning the residual instead of the actually mapping has the advantage, that if new Conv layers are added and stay unlearned the whole residual block still yields the identity mapping which will not lead to the degradation problem we see with other deep networks architectures.

2.4.1 Network design

Using the basic concept of residual blocks we build a successful architecture. First lets look at what the blocks can contain. The first example in Figure 2.4 uses just



(a) A simple non-linear residual block.

(b) A bottleneck block as employed in the deeper ResNets.

Fig. 2.4.: Two examples of residual blocks. The input of the block is fused into the ouptut of the last Conv layer and added element-wise channel-per-channel onto it. A ReLU is trailing the sum again.

two Conv layers and a ReLU to learn the residual. Therefore the block is described by:

$$\mathcal{F}(x) = W_2\rho(W_1x + b_1) + b_2 \tag{2.10}$$

W_i and b_i denote the weights and biases for the two layers, respectively and the ρ denotes the ReLU thus making the block non-linear. Actually any chain of layers could be used to model the residual mapping. One could use this technique with just one Conv layer but as the original authors [He+16] depict, this will result in a normal linear projection which does not improve performance compared to a normal network.

Deeper networks will become more complex. We want to limit the increase in complexity. This can be done with bottleneck blocks, as the second example in Figure 2.4. Here we use 1×1 Conv layers to first reduce the dimensionality of the input and to increase it again after the convolution. The actual mapping convolution now has a smaller input dimension. For this work we use the 50-layered ResNet as described in the original paper. Its architecture is shown in Figure 2.5. At first stand a Conv layer, with a big kernel size and stride 2, followed by a max pool, thus directly reducing the spatial size to a quarter of the input size. Then we build four chains of residual blocks, each at a constant input size. Between each chain we downscale by using a once using stride of 2 in a Conv layer and in return double the channels. The last chain returns responses of size 7×7 and with the average pooling layer we get to a singular score over all channels which are classified using the the final FC

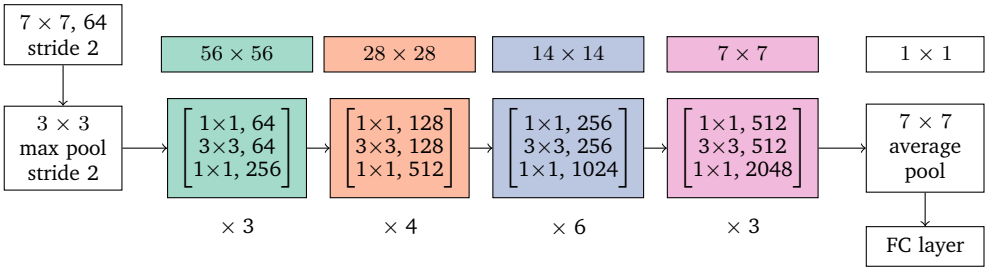


Fig. 2.5.: ResNet50: Each bottleneck block has three Conv layers with their kernel sizes and channel count denoted in the graph. Beneath each block is the recurrence written and above are the output sizes for these blocks.

layer.

This ResNet does not contain any dropout layers or different separate regularization elements. The original authors argue that the thin deep structure of this design does inherently regularize.

2.5 Evaluation

The result of this work is a binary object detector. Therefore a test run of the detector will result in a set of n boxes B_p, that the method predicts as positive (object of the class the classifier was trained on) with probabilities $p_p \subset \mathbb{R}^n$. For the test set of images we know the actual ground truth boxes B_g of this class. With some defined condition we mark the predicted boxes B_p as true positive or false negative. The boxes from B_g, that were not predicted as positives by the detector, are false negatives.

Precision and Recall

One way to evaluate a binary detector is to plot the precision-recall curves. Precision is defined as the proportion of the predicted positives that are actual positives:

$$\textit{precision} = \frac{|B_p \cap B_g|}{|B_p|} \tag{2.11}$$

Therefore a detector which does not predict any positives would have perfect precision even though it is obviously bad. The counteracting measure to use is the recall. Recall or also called True positive rate (TPR) is defined as the proportion of all ground truth positives that we predicted as positives:

$$\textit{recall} = \textit{tpr} = \frac{|B_p \cap B_g|}{|B_g|} \tag{2.12}$$

Here a detector that predicts only positives will receive full recall and therefore we want a model with high precision **and** high recall.

Our detector returns probability scores for each positive predicted box. To reduce those to a binary decision of *does belong* and *does not belong* we have to select an cut-off value for accepting positives. We can look at how precision and recall change while we change this threshold. This transition is plotted in the precision-recall curve in which we plot precision against recall over the changing discrimination value.

Receiver operating characteristic (ROC)

Another often employed evaluation method are ROC curves and the area under the curve (AUC) of those. For the ROC we pose the TPR which tells how many of the positives we detected as positives against the False negative rate (FPR) which tells how many of the negatives were predicted as false positives. TPR tells the probability that we hit a positive and FPR tells the probability of an false alarm.

Plotting TPR against FPR over the changing discrimination threshold the same way as the precision-recall curve returns the ROC curve. The perfect detector would have a TPR of 1 and a FPR of 0, therefore we want a detector approaching this point of perfect classification. The ROC space reaches from 0 to 1 over both axes. A randomly guessing detector maps to the diagonal in this plot which is then discriminator between good and bad models.

To even summarize this metric more we calculate and report the AUC values for the different models. As the name states the AUC is the area under the ROC curve with values in $[0, 1]$. Because the random classifier returns the diagonal inside the ROC space, the AUC value can be interpreted as the probability that our detector will score a positive ground truth sample higher than a negative one. With AUC of 1 it would be perfect with 0.5 it would be random. We calculate the AUC with the trapezoidal rule applied to the ROC curve.

3 Related Work

In this chapter we quickly want to revise other works touching the tasks of this thesis. In Section 3.1 we discuss multiple successful object detection methods using CNNs and in Section 3.2 we list previous image retrieval systems also focusing on those employing CNNs.

3.1 Object detection

The first work to approach object detection with a CNN were Girshick et al. [Gir+14], with their **Regions with CNNs (R-CNN)**. For each image they use Selective Search to generate a set of region proposals. Selective Search is a technique to identify object-like, but type independent, regions probing texture and color of the image at multiple scales. All these boxes are reshaped to a uniform size and forwarded through a normal classification network (in their case AlexNet [KSH12]) to yield feature vectors from the FC layers. Lastly a multi-class Support Vector Machine (SVM) classifies these proposals. Additionally the R-CNN is trailed by a linear regression model which uses the classified boxes to produce tighter box coordinates. Although the results proved that CNNs can greatly improve detection results, the method is hard to train because the classifier network, the SVM and the regressor have to be trained independently. Also at test time classifying each region box independently is extremely slow.

A year later again Girshick [Gir15] released **Fast R-CNN** which improves computation performance and training complexity. Instead of computing features for each proposed region, Fast R-CNN computes the features for the whole image at once. Then the regions in the image are projected onto the feature map of the network, thus yielding the patch-wise features. Those patch-wise feature maps are pooled, forwarded through FC layers and finally into a classifier and the box regressor. The SVM is replaced with a SoftMax and the linear regression model with a additional FC layer. As such the three models from R-CNN become one unified network which is faster and easier to train.

In the same year Ren et al. [Ren+15] tackled the big bottleneck of this approach, which is the region proposal system, with their **Faster R-CNN**. The previously used Selective Search is slow and still independent of the model. Using also the features generated by the Conv layers, they add a new network branch, a Region Proposal Network (RPN), to generate box proposals. These are used for the same Region of Interest (RoI) pooling as in the Fast R-CNN. The RPN is a FCN that takes the feature map and at each positions tests for a predefined set of k anchor boxes whether those could be objects. So for each window the RPN outputs a score of *objectness* and coordinates of the bounding box. Those are then the inputs for the Fast R-CNN method.

This year He et al. [He+17] extended this approach with **Mask R-CNN** to pixel-wise segmentation of images. In the previous versions the regions from the original image were projected onto the feature maps by just down-scaling their shapes with the respective factor and discretizing them on the feature grid. This rounding introduces errors in the proposed bounding boxes and leads to misalignment. Instead they use bilinear interpolation inside each region to find more precise feature representations. This process is done by a new branch inside the Faster R-CNN and outputs binary pixel-wise masks for each RoI. The regions are classified as in the Faster R-CNN and as such they are able to return semantic image segmentations.

Dai et al. [Dai+16] incorporates the RoI-wise processing into the main network branch of their **R-FCN** through a series of Conv layers. The Conv layers encode scores at relative positions like top-left or bottom-center. Next the regions from the RPN are divided into the same relative sub-regions and each sub-region is used to pool from the corresponding layer and then to vote for the whole RoI.

As our work also has to focus on time performance, it is related to other fast object detectors. Redmon et al. [Red+16] use a freshly designed network architecture **YOLO** to achieve object detection at real-time. The input image is divided into a regular grid. The last FC layer from their network predicts for each cell of the grid the class probabilities and the coordinates and objectness confidence of two bounding boxes. Detections get selected by the multiplication of their conventional class probabilities and their individual confidence predictions. While the method is extremely fast it is limited by the shape of bounding boxes it will predict.

3.2 Image retrieval

Retrieving relevant images from large databases by searching for similarities to an exemplar image, is a long used idea. The first methods employed hand-crafted features like histograms, SIFT or HOG to compare different images in a low dimensional space. Problems in such approaches arise with semantically close groups of objects with wide visual differences.
One of the more classical but successful approaches is **OASIS** [Che+10]. Here the authors solve a bilinear similarity function for whole images using sparse local pattern descriptors. While the approach is successful under its training settings it is heavily restricted in that it has to be trained on all needed image classes.

Wu et al. [Wu+13] learns image similarity using stacked neural networks. The inputs for those are low image features and they learn representations of modalities of these features giving a optimized multi-model combination. With a bag of five features they achieve better results, retrieving similar images, than OASIS but also lack wider applicability through the complexity of training.

The step forward to just learning similarity on top of raw pixels is done by Babenko et al. [Bab+14], as they show that using just the distance between the outputs of FC layers, it is possible to retrieve visually similar images. More sophisticated is **DeepRanking** [Wan+14] use the deep feature of an AlexNet, together with visual features extracted by convolutions, in a linear embedding as a image descriptor. With the distance between the embeddings as measure, they rank the similarity of images to a query image. The visual features and the linear embedding are learned by feeding the network triplets of query, positive, and negative images. They show that the learned features outperform hand-crafted ones.

Neural networks shown success in being used in reverse image search systems. In [KV15] they train different network architectures on images of shoes and use a distance measure between the descriptors from the trailing FC layer to retrieve alike shoes to a query shoe. Similarly the image descriptions from a task-specifically fine-tuned network are used in [CL15] to select similar styles inside clothing catalogs. Using the idea of [Wan+14] Bell and Bala [BB15] learn a embedding of image descriptors. They use a siamese network architecture to learn from tuples of similar or dissimilar pairs. With the learned embedding space they retrieve similar interior designs given a query product by picking close training samples in the embedding space.

4 Our method

In this chapter we explain how we build our network from ResNet and under which conditions we train it (see 4.1). Next we show how we generate bounding box object predictions on top of the pixel-wise class probabilities (see 4.2).

4.1 Training

We use the ResNet-50 (see 2.4.1) as the basis for our detector and take the pre-trained model[1] that is provided by the original authors. They trained the network on the 1000 classes of the ImageNet 2012 dataset [Rus+15] and therefore we can assume that the learned features are good class-independent image descriptors. For our single-class detection we replace the original FC layer with a new one, now only returning two values (background *vs* foreground).
As we aim for fast *approximate* training on a single class, it is not feasible to retrain or refine any of the Conv layers and instead we only train the weights of the single FC layer. Additionally refining the last Conv layers in the same restricted number of iterations did not yield considerable better results.
We want to use an FCN to provide localized object predictions. This is done by the classifier to FCN conversion described in Section 2.3. The classifying ResNet is transformed by converting the FC layer into a Conv layer after training and appending an upsampling operation of the sparse prediction map. Following the results of [SLD16], we do not train the upsampling and use fixed bilinear weights. Again the new Conv layer could be refined on the pixel level using segmentation maps but under our self-imposed restrictions the further training did not result in considerable success.

For optimizing we use SGD with momentum of 0.9. The weight decay is set at 0.0001 and the learning rate fixed at 1×10^{-6}. Employing a learning rate reduction method, like reducing after fixed steps, did not influence training results and is thus omitted. Because we want time limited training we abort training independent from any validation measure after 500 iterations. Thereby we guarantee constant time complexity in training. The fixed number of iterations was chosen after multiple test with shorter and longer training durations.

[1] `https://github.com/KaimingHe/deep-residual-networks#models`

4.1.1 Preprocessing and augmentation

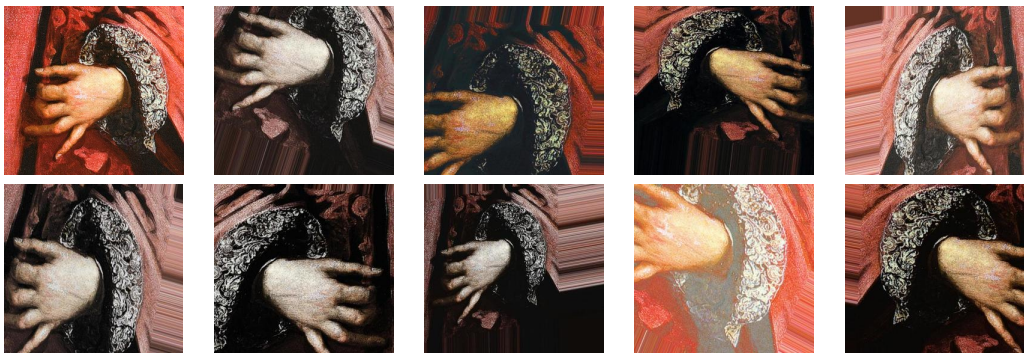


Fig. 4.1.: Top-left is the original patch and the other ones are transformed samples. Painting sample is from the *Portrait of the Procurator Giolamo Querini* by Bombelli, Sebastino [Bom69].

It has long been shown that preprocessing the training data and expanding it by modifying the data improves training time and test results [Dos+14].
We preprocess the training samples I_k with only one step, by subtracting the mean of all images of the dataset from each sample [KSH12]. Through this step the input data becomes zero-centered.
Moreover as we will increasingly reduce the number k of samples to train from, data augmentation is immensely important. We employ multiple image transformations to artificially enlarge the training set. For this we use the preprocessing methods provided by the Keras library [Cho+15] and add additional transformations. Generating new samples is done with a set of bounded transformations. The composite transformation T_p is chosen with the parameter vector $p = \{\alpha, \beta, s, t_x, t_y, m, t_s, t_v, f_s, f_v, v_s, v_v\}$ defining the following transformations which are applied in the given order:

- **Rotation** with $\alpha \in [-20°, +20°]$
- **Translation** with $t_x, t_y \in [-5\,\%, 5\,\%]$ of the patches' size in either directions on both axes.
- **Shearing** with $\beta \in [-11.50°, +11.50°]$
- **Zoom** with $s \in [70\,\%, 130\,\%]$
- **Flipping** horizontally $m \in \{0, 1\}$.
- **Contrast**. Patches are converted to the HSV color space. We raise saturation and value to a power $t_s, t_v \in [0.25, 4]$, then multiply by factors $f_s, f_v \in [0.7, 1.4]$, and add these to some values $v_s, v_v \in [-0.1, 0.1]$ [Dos+14]. After the contrast transformation patches are again converted back to RGB color space.

Some examples of transformed images under those boundaries are seen in Figure 4.1. The parameter boundaries were chosen similar to those in Dosovitskiy et al. [Dos+14]. While training, the image generator picks random images from the training set, processes each image using a random parameter vector and returns the transformed images online. Finally they are resized to a size of 224×224 pixels and fed into the network.

Because the generation of new augmented samples severely slows down training if done on demand we generate new samples in parallel to the network training on the CPU.

For good training performance we need a balanced set of positive samples and negative background samples. Therefore we want to feed the image augmenter a balanced set of seed patches. We enlarge the set of positive seed samples by initial randomly translated patches around each $i \in I_k$ with $t_{x'}, t_{y'} \in [-25\,\%, 25\,\%]$. This also has the advantage, that the image augmenter additionally receives the surroundings of the original query box. Depending on the set size k we get ppI number of seed samples from each image i and take the same number of negative seed samples. Thereby we have at last $n = 2 \cdot k \cdot ppI$ input samples for the generator.

4.2 Generating detections

Our desired output of the algorithm is one or are multiple boxes $B = \{b_0, \ldots, b_n\}$ enclosing each a section of the image I, which are predicted to contain the object that was searched for. The FCN returns a score map M as output. Each score $m_{i,j} \in M$ predicts the probability of this pixel (i, j) to belong to the class c we trained for. Therefore we need a method $g(M) = B$ to return high scoring bounding boxes from a probability map. The way of doing this used here consists of multiple steps:

1. We penalize (near-) empty regions inside the probability map with a negative distance transform
2. We generate a set of boxes on a regular grid over the image shape
3. For all those boxes we calculate their density of scores
4. We apply a low threshold to filter out unnecessary boxes
5. We apply Non maximum suppression (NMS) to get the top scoring boxes

In the following sections we explain each step in depth.

4.2.1 Negative Distance Transform

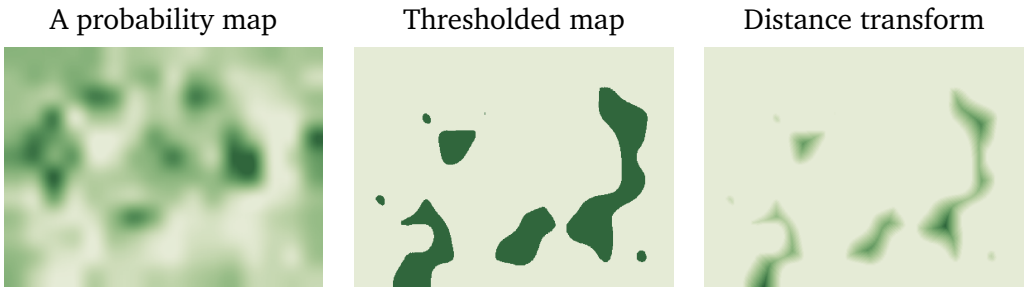


Fig. 4.2.: Computing the negative distance transform from a probability map

Many images produce probability maps M, that are sparsely filled. Positive regions receive a positive score from the network, while non-positive regions are at-most zero. Thus the maps contain flat regions of constant value. We want to score the inner lying values of flat regions negatively, compared to boundary values. Here we introduce a distance transform to, sort of, impress those regions (see Fig. 4.2).
First we threshold the score map with a near-zero value to set all values, that can safely assumed to be negatives, to zero.
Second we apply a distance transform to the mask M' that resulted from the thresholding. A distance transform fills each zero pixel of the mask with its distance to the nearest one valued pixel. The distances are calculated with some metric, like the Euclidean distance or the L_1 distance. As the specific metric is not important for our vague penalizer and as we aim for good time performance we use the L_∞ distance, also known as the chessboard distance. For our 2-dimensional image space the metric defines the distance between two points (x_1, y_1) and (x_2, y_2) as follows:

$$L_\infty = \max(|x_2 - x_1|, |y_2 - y_1|) \tag{4.1}$$

The transformed image is normalized and substracted from the original probability map. This additional step increases mean AUC on the PASCAL-Part by 2% for sets with < 25 images.
Nevertheless does this additional computation slow down the image processing and with only a marginal improvement this step probably can be omitted in live applications.

4.2.2 Calculating the score densities

For each processed image we generate a set of boxes to calculate the score densities for. At three scales we take boxes $B = \{b_1, \dots, b_i = (x_0, y_0, x_1, y_1)_i, \dots, b_k\}$ with three different aspect ratios for $k = 9$ [Ren+15] proposals at each position. We slide these anchor boxes with overlapping stride over the image and collect the resulting slices. Also we save the areas $\{A_1, \dots, A_i, \dots\}$ for all rectangles.
Density is calculated by summing up all the probabilities from the score map inside each window and dividing those by the area of the box. To achieve that as efficiently as possible we use the integral image M_Σ of our score map $M \in \mathbb{R}^{w \times h}$. A value $M_\Sigma(x, y)$ in the integral image is the sum of all values in the source image lying left and above the anchor coordinates (x, y):

$$M_\Sigma(x, y) = \sum_{\substack{x' \leq x \\ y' \leq y}} M(x', y') \tag{4.2}$$

Computing M_Σ has only $\mathcal{O}(w \cdot h)$ complexity and with the integral image the sum inside any rectangle of M can then be yielded efficiently with:

$$\sum_{\substack{x_0 < x \leq x_1 \\ y_0 < y \leq y_1}} M(x, y) = M_\Sigma(x_1, y_1) + M_\Sigma(x_0, y_0) - M_\Sigma(x_1, y_0) - M_\Sigma(x_0, y_1) \tag{4.3}$$

Next the probability density of each box is computed through:

$$\rho_i = \frac{\sum_{b_i} M(x, y)}{A_i} \tag{4.4}$$

Using the density to rank the boxes B the ranking will favor flat regions of high scores. While this will probably be a region of a positive detection we can expect the top scoring regions to be inside the object because including the boundaries would lower their densities. In Section 5.3 we will indeed recognize this effect.

4.2.3 Non maximum suppression

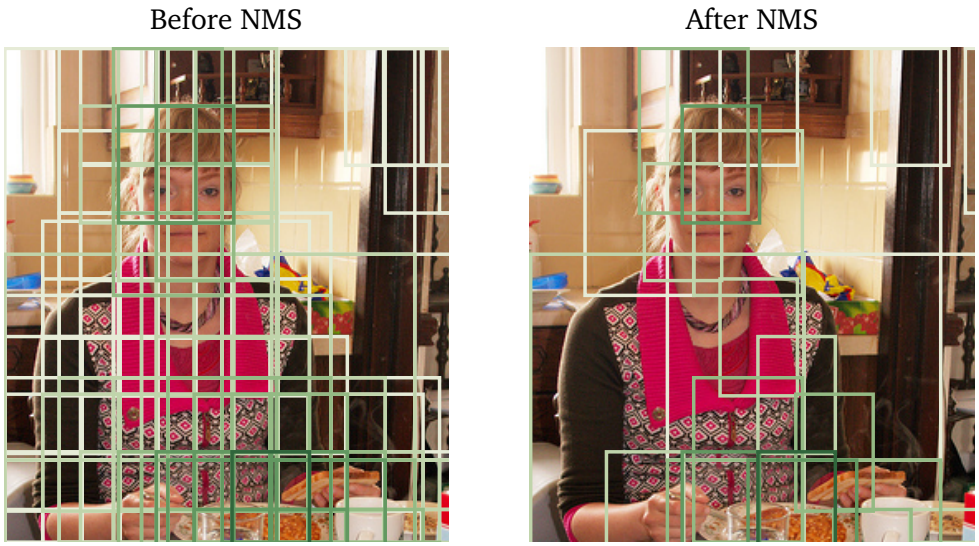


Fig. 4.3.: The effect of Non maximum suppression (NMS).

Depending on the number of scales, we generate hundreds or even thousands of boxes per image. We want to eliminate boxes that are overlapping with another box, which has a higher density. Firstly we drop the worst boxes $\{b_i : \rho_i < \rho_{min}\}$ with a low set threshold. While this is not necessary, it improves time performance and does not affect testing performance.

To the remaining boxes we apply NMS, as described by Felzenszwalb et al. [FMR08]. In the NMS we first sort all boxes by their scores ρ_i in descending order, which in our case are the probability densities. We mark the strongest box as picked. Then we calculate the overlap in area between the new picked box and all remaining boxes. All boxes of which the area overlaps the picked box with 50% or more are dropped. We pick the next strongest score of the remaining set and continue this process until all boxes are either dropped or picked. The strongest not-overlapping boxes remain picked and are returned. Figure 4.3 shows the condensing effect of NMS.

For minimal computational cost of this step we use the improved implementation from Malisiewicz et al. [MGE11].

5 Test results

Testing of the method is done on the PASCAL-Part dataset [Che+14], which provides part-based segmentation annotations for images of the PASCAL-VOC 2010 challenge dataset [Eve+10]. For the 20 object classes PASCAL-Part contains binary segmentation maps for a selected number of parts of all objects classes.
For testing the detection method we train a model with different number of images. From PASCAL-Part we pick sets of classes $c \subset C = \{\texttt{bird}, \texttt{person}, \texttt{train}, \dots\}$ and parts $p \subset P = \{\texttt{head}, \texttt{beak}, \texttt{lhand}, \dots\}$. We gather all images from PASCAL-VOC 2010 which contain parts p from the classes c. Then we extract all patches of these images around the selected parts. We call this set of image patches $I_{c,p}$, e.g. all images of beaks of birds. Then we pick a random subset I_k of k images from $I_{c,p}$ and train the network with this subset. We want to investigate the influence of k on detection results as we assume that the networks' success is neatly correlated to number of examples.
The remaining set of images $I_{c,p} \setminus I_k$ will be the test set for this single network. For each set (c, p) we take random patches out of the corresponding PASCAL-VOC images guaranteeing that they do not overlap with images in $I_{c,p}$. These patches are our negative samples for (c, p) and will be fed into the same data augmentation as the positive samples.

Following in the next chapter we first rationalize our reduced and unequal number of tests (see 5.1). Next in Section 5.2 we quickly explain the baseline method, which was used as comparison for our results. Finally in Section 5.3 we report our test results in detection success and time performance (see 5.4).

5.1 Repeated training and testing

Obviously we also have to consider the quality of the subset, not only its size. As we know that a part of the generated and augmented samples is way to small or contains no significant structure to learn, we have to assume that some image patches are not useful for training. Some patches are smaller than 10×10 pixels, and others are nearly monochrome. To avoid the influence of bad images, we repeat the training with one set size k multiple times and calculate the mean results of the multiple runs.

samples	1 round	2 rounds	4 rounds	10 rounds
1	0.7	0.91	0.9163	0.9527
10	0.8497	0.9774	0.9880	0.9986
25	0.9022	0.9904	0.9965	0.9999
50	0.9522	0.9977	0.9996	0.9999
100	0.9875	0.9998	0.9999	1.

Tab. 5.1.: Probabilities that at most half of the rounds fail for different number of samples and different numbers of repetitions.

Lets assume 30% of all image patches are useless for training. For how many times do we have to repeat the training with k images, so that we can neglect the influence of these bad samples? To get a rough estimate we reduce this problem to a binary one. First we compute how probable it is that a training will fail for one k.
The probability p_b that one image is bad is 0.3. Lets further assume, for simplicity, that a training will still sufficiently succeed if up to $v = 40\%$ of the samples are bad. We get the probability that the training will succeed from the Cumulative Density Function (CDF) of the binomial distribution:

$$p_s(k) = B_{k,p_b,vk} = \sum_{i=0}^{vk} \binom{k}{i} \cdot p_b^i \cdot (1-p_b)^{k-i} \tag{5.1}$$

For $k \in \{1, 10, 25, 50, 100\}$ this returns $p_s = \{0.70, 0.85, 0.90, 0.95, 0.99\}$, respectively. These are the probabilities that we will not fail one specific training. The probability that a training fails is $p_f = 1 - p_s = \{0.30, 0.15, 0.10, 0.05, 0.01\}$. Now we want to look at the probability, for an n times repeated run of k images, that half or less of these rounds fail. Again this can be computed with the CDF:

$$p_{repeat}(n, p_f) = B_{n,p_f,n/2} = \sum_{i=0}^{n/2} \binom{n}{i} \cdot p_f^i \cdot (1-p_f)^{n-i} \tag{5.2}$$

From the estimations (see Tab. 5.1) we infer that it is sufficient to test the high sampled models way less, than the low sampled ones. Each test, if not noted different, will be the mean results of $\{20, 20, 8, 4, 2\}$ rounds for $\{1, 10, 25, 50, 100\}$ samples, respectively. Through this cascade scheme we reduced the needed test rounds from 100 to only 54, without losing accuracy in test results.

5.2 Baseline

To compare our results we additionally solve the task with a baseline method. We build a simple object detector inspired by Felzenszwalb et al. [Fel+10] using Histogram of oriented Gradients (HOG) [DT05].

5.2.1 HOG

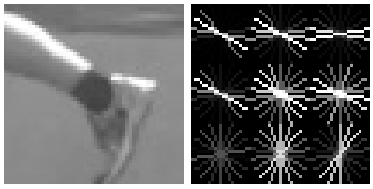


Fig. 5.1.: A example of a patch and the resulting HOG cells.

HOG features encode the shape of an object assuming that its shape can be described by the distribution of edge directions. As the name already states we want to look at the gradients of the image at different orientations. We compute the horizontal and vertical gradients from the relative luminance of the original image, using the standard weighted channel contributions. To get the spatial localization we split the image into small regions called cells. For each cell we compute the histogram, which is just an $1d$ vector containing the gradients over a fixed set of angles in that cell. Next we group these cells into larger overlapping blocks. The blocks are used to normalize the contained histograms. This introduces better invariance to illumination and edge contrast. As the blocks are overlapping, each cell will appear multiple times in the output but under different normalization. For normalization we use the L2-norm followed by clipping [Low04].

5.2.2 Training

We collect 1000 samples of augmented samples, using the same method as described in Section 4.1.1. For all samples we compute the HOG features using the settings as in [DT05]. In Felzenszwalb et al. [Fel+10] their model consists of multiple parts each described by its own HOG features. Because we will only train for small parts we describe each sample by only one HOG descriptor at one scale. Using the set of balanced positive and negative samples we train a linear SVM.

5.2.3 Testing

The testing pipeline should be as similar as possible to the novel method. To use the same box generation, density calculation and NMS as in Section 4.2, we need a map of class probability predictions as input. Sliding a window over the images and computing the HOG features for all positions in the image is rather inefficient. Instead we compute the HOG descriptors for the whole image and classify a window that is slided over the feature map. The SVM returns the predicted class (background *vs* foreground), and the scores are aggregated for all windows to build the needed probability map. Although it is not useful to train a multi-scale classifier for our small training samples, we have to take differently scaled test objects into account. Thus we test on multiple scales. Because the SVM needs a fixed shape, the same number of HOG blocks, as input to predict their class, we change the HOG parameters for the different scales. For windows half the size we also halve the width of each HOG cell, thus resulting in the same number of blocks per window.

5.3 Detection results

Fig. 5.2.: Examples from `person_hair`: Top detections are from a run with 1 sample, bottom detections are from a run with 50 samples. Note how the 50 sampled network detects more variations of hair while upper one fails at most. Also note how both networks miss-detect the cats head as human hair.

We test the pipeline for the class-parts combinations (c, p) listed in appendix A.1, using the cascade scheme described in Section 5.1. To evaluate the detection we collect all predicted bounding boxes for all images and their scores. Boxes which areas overlap 70% or more with a ground truth patch are counted as positive, all others as negative. All evaluations can be found in appendix A.2. Refer to Figure 5.4 for summary of the test results. As the first takeaway of our results, Figure 5.2 exemplary shows that our region scoring approach prefers smaller boxes inside the RoI we actually want to enclose. Because of our overlap condition, which does not penalize this, the under-scaled boxes are not reflected as misses in the evaluation results.

For a subset of the evaluations we report the evaluation of the baseline. The detector using HOG features returns poor detections independently of training set size. To simplify the comparison to our approach we only report the best results of the baseline, achieved using the biggest set size ($k = 100$). While the baseline provides evidence for the success of our method, we can not use the overperformance to disqualify any classical detector as we employ highly simplified model.

We clearly show the correlation between set size and the success in testing (see Fig. 5.2). Training with only one sample returns the worst results, by distance, with a mean AUC over all parts and all runs of 0.72. With increasing set size the test results saturate quickly. The mean AUC for all larger set sizes lie inside a range of only 0.03. That may be explainable with our training settings, which are set for fast training and do not let time to incorporate a high number of examples into the representation learning.

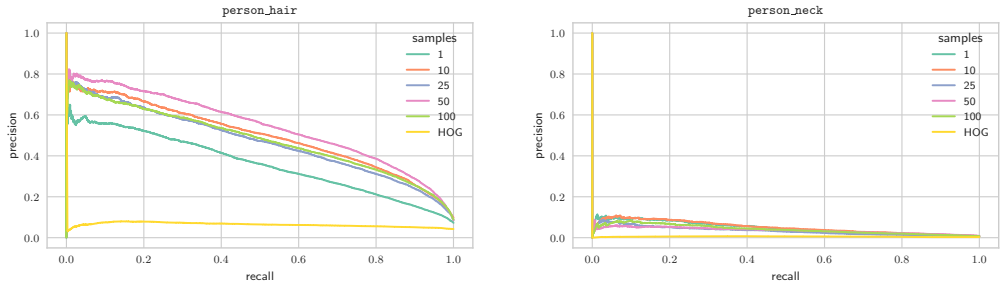


Fig. 5.3.: Precision-Recall curves for two parts. `person_hair` on the left as an successful example and `person_neck` as an example for an utterly failed training. Certainly though we see in both plots the clear separation between different set sizes.

Besides the anticipated correlation between set size and test success, we also show the correlation between different object parts and the success in quickly learning their representation. An example for this difference is given in Figure 5.3. Training on human hair was one of the more successful experiments while training on human necks does not return a useful classifier. The difference can be explained by the amount of discriminating structure a part has to offer. While necks most of the time are very small mostly contrast-less and structure-less patches, hair has a distinguishable surface and therefore more recognizable visual features.

Another potential explanation stems from our training method. We train our classifier not on the pixel-precise masks, that PASCAL-Part provide, but on rectangular patches retrieved from the original images around each part. As such a patch of human hair will, most of the time, contain a section of the head underneath. One therefore can expect that the model not only learns the representation of the part itself but also of the surrounding object segments. We see this relation in the quite similar results of `person_hair` and `person_head`. Firstly such spatial relations are only relevant in some cases, and secondly while this may frustrate part-based training in some cases, most of the time spatial relations to semantically related parts are an actual indicator for the searched part. If we search for human hair and find human heads, we also found many instances of human hair.

Furthermore we want to check that the cascade testing scheme was intensive enough to report sufficiently precise results. We calculate the variance of AUC scores for the different parts and training sizes to verify their precision (see Fig. 5.5). As we expected the variance drastically decreases with increasing set size, and as such we conclude our testing scheme to be adequate for this random sampling. Even for the 20 times repeated 1 sampled training runs, the variance of AUC scores remains under 0.02 most of the times.

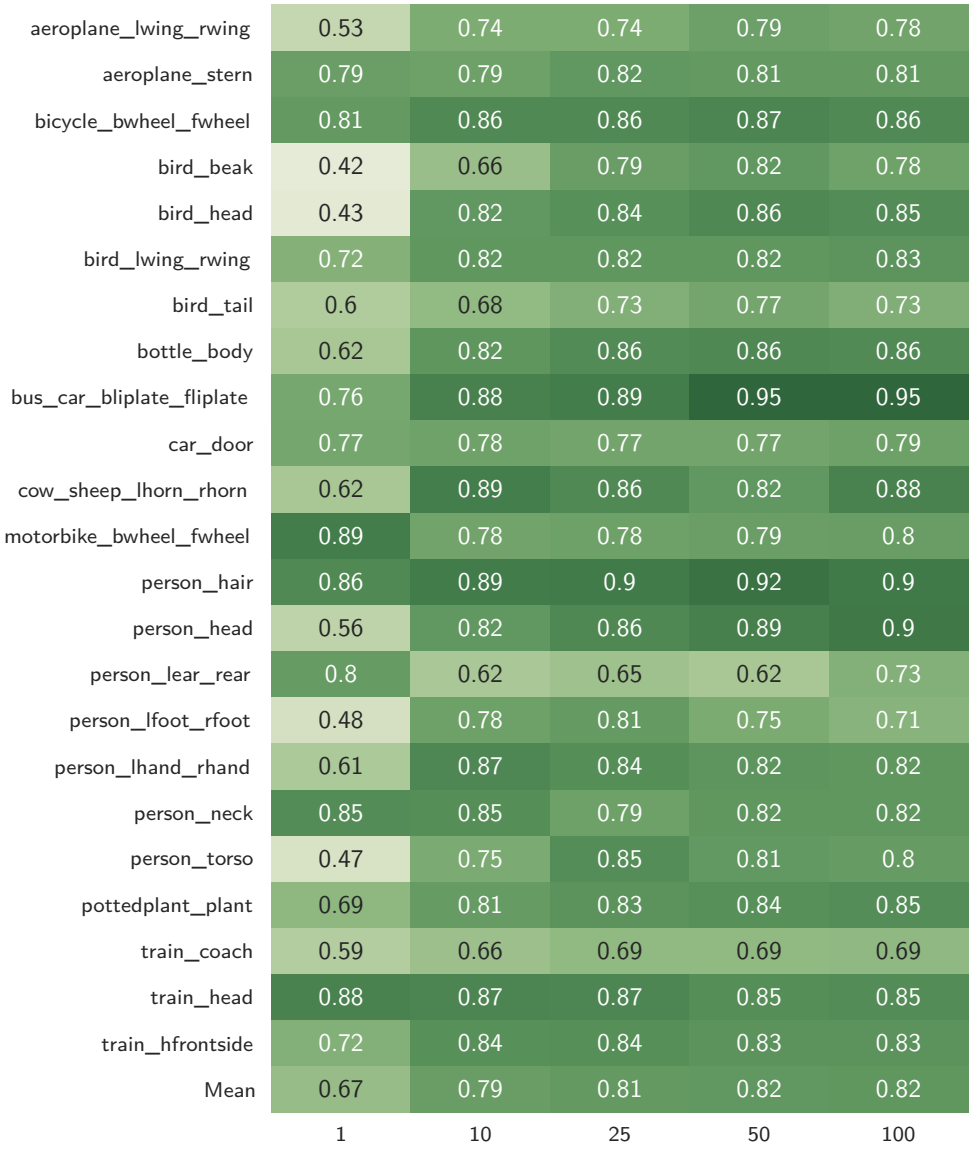


Fig. 5.4.: AUC of the ROC curves at different classes and training set sizes.

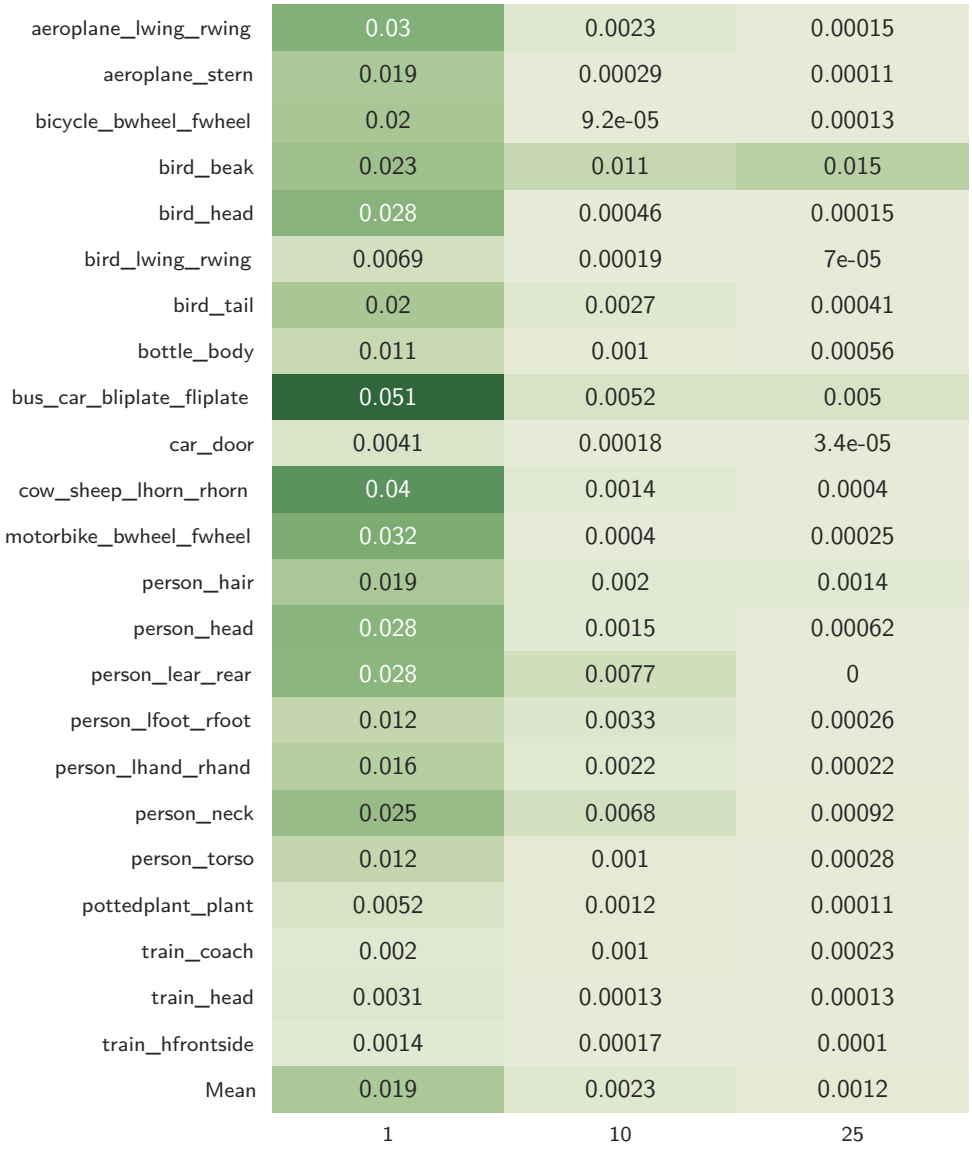


Fig. 5.5.: Variance of the AUC at different classes and training set sizes.

5.4 Time performance evaluation

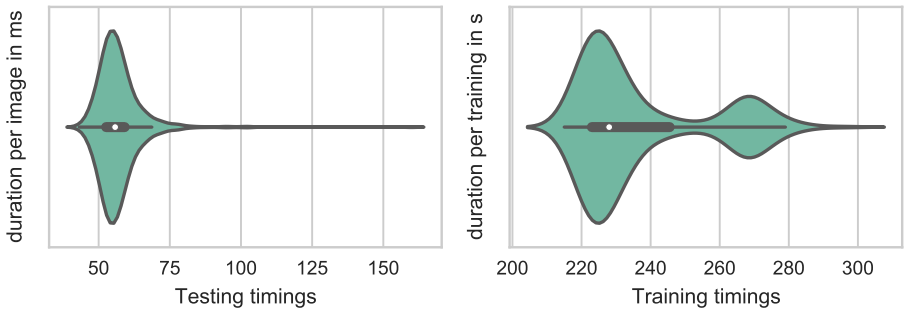


Fig. 5.6.: Distribution of durations in testing and training. Time in training measured in seconds for training over 500 iterations. Time in testing is measured in ms per image completing forwarding and all postprocessing.

For the application of this method in a live production setting, speed is from huge importance. Therefore we measure and report the duration for training and deployment of the network (see Fig. 5.6). Training takes around 3.5 minutes on a single Titan X and testing runs at around 55ms per image. With therefore approximately 18 images / second we are reaching real-time search in image databases.
Although this is a short time for a train and test run of a neural network it nevertheless does reduce the applicability of this approach. If we assume an art image database of 10.000 images, a query and complete search through the database would take $t \approx 210\,\mathrm{s} + \frac{10000}{18} s = 12.80\,\mathrm{min}$.

There might be options to improve the time performance. One idea is to precompute the last feature maps for all images, so that, while searching through the images, we only have to compute a single convolution and the bounding box retrieval. This might drastically improve retrieval speed but training the network still causes a practical barrier.

6 Application

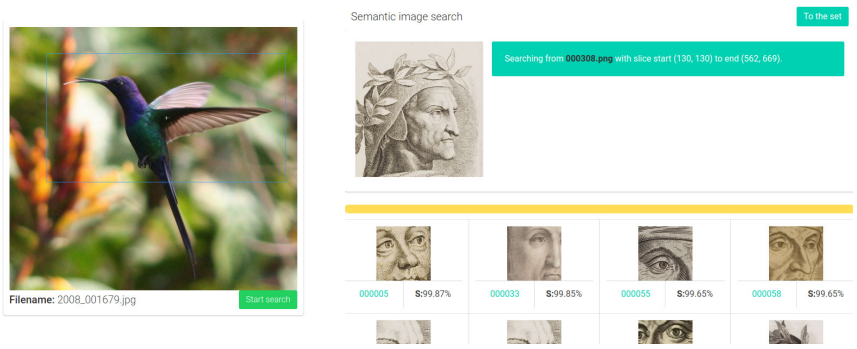


Fig. 6.1.: Two screenshots of the provided online app using our method. Left shows the selection screen previous to a new search. Right shows the result screen displaying live results.

We bring the described method to application as a semantic image search machine. For this we build a small exemplar web application using the Flask microframework. After configuring the program for a new image database, the web application enables users to search for similar objects given a query image patch. In the main view the user can sift through the image dataset to pick an image which contains the object of interest. For a single image the Graphical User Interface (GUI)(see Fig. 6.1) provides a intuitive way to select a rectangular part of an image. Alternatively the user may upload an image to the web page, to search the database with an external image. Next the user can start the search with the provided patch. The backend quickly fine-tunes the ResNet under the same conditions as described in Section 5. Because we do not have any semantic information about the images or the objects seen in them we can not yield guaranteed negative samples given a query patch. Thereby we resort to random patches that we randomly sample at different sizes from the whole dataset. In training we pick 500 seed patches from the query image so that we can set those against a equal number of random chosen negatives. We assume that even with some actual positive under the negatives, the high number of samples lowers their influence in training.
Following the training, the network is instantly reshaped into the FCN and set up for normal image forwarding. The images from the database are all forwarded through the detection pipeline. While the network is still running the program collects the best results and presents the patches with the highest score to the user, in descending order by their scores.

6.1 Retrieval examples

Fig. 6.2.: The first patch on the left is the query image. Right are the five highest scoring retrievals cutted to show the context. The actual proposed patches are drawn sharp on the blurred source images.

Returning to the objective domain of this thesis we set up our application with two dataset of art historical images. The first is a collection of drawn or printed portrait illustrations containing 923 images. The second consists of 567 scans of pages from the *Sachsenspiegel* [vonnd], a German law book from the 13th century.
Figure 6.2 displays some qualitative examples from those sets. With the retrieved patches the presence of too small bounding boxes (see 5.3) again becomes apparent. The first two rows are from search runs on the Portrait dataset. Observe in the first row how from the query portrait of *Dante*, three other portraits of *Dante* were retrieved (the second retrieval is from a different image depicting the same portrait). The two rows in the middle show examples from the *Sachsenspiegel* set. Generally most tests with this dataset proved to be unsuccessfully with the second example at least retrieving some other patches of horses or similar animals. Without further investigation we suspect that might derive from the overall similarity of the depicted figures and the fact that most images are substantially filled with text which therefore make up considerable number of the negative training samples.

At last we show an example of a cat from the PASCAL-VOC dataset which exclusivity retrieves cats for the at least top-hundred matches.

7 Conclusion

In this thesis we present an zero-knowledge image retrieval system employing a Fully Convolutional Network for efficient object detection. With a set of transformed samples derived from a query image we fine-tune the underlying ResNet. Training is done without a training progress dependent abort condition. The linear classifier is converted to a convolution to predict sparse class probabilities on images of varying size. Using these probability maps we predict the existence of alike objects in images. We yield these patches and thereby we retrieve object instances from image set related to the training samples.

With our results on PASCAL-Part and the application on art historical images we provide evidence that this approach can carry out image retrieval in unlabeled, context-less images using a small number of examples, ultimately only a single one. On the one hand, the tests on PASCAL-Part verify the supposition, that providing the network with more positives image samples drastically improves its detection results, on the other hand we showed that under our test settings, results quickly converged over an increasing number of examples.
Although we prove the methods effectiveness we recognize multiple drawbacks. Firstly the time complexity (see 5.4) restricts its use on small datasets of images and makes instance retrieval somewhat cumbersome. Secondly our region scoring method throughout under-scales object boxes thus serving more as a translation prediction.
The approach could be continued and improved on multiple different ways. Of most importance is the unanswered question, how to use the image representation provided by the deep filter of a CNN to bootstrap an approximate object detector without having to actually iteratively train the network. We could not test any approaches in this domain and as such can not predict a possible solution.
To fasten the approach it may be possible to formulate the complete pipeline inside the neural network, reducing time costs. Taking densities of regions is nothing else than average pooling and thus could be done within the network after convolutional upsampling.

All code is made publicly available under `https://github.com/mrtukkin/bachelor-thesis`.

Glossary

AUC area under the curve. 9, 10, 21–24

CDF Cumulative Density Function. 19

CNN Convolutional Neural Network. 2–4, 11

Conv Convolution. 3, 6–8, 12

FC Fully Connected. vii, 5, 6, 8, 11, 12

FCN Fully Convolutional Network. 5, 6, 12, 15, 26

FPR False negative rate. 9, 10

GUI Graphical User Interface. 26

HOG Histogram of oriented Gradients. viii, 19–21

NMS Non maximum suppression. 15, 17, 20

R-CNN Regions with CNNs. 11, 12

ReLU Rectified Linear Unit. 3, 7, 8

ResNet Residual Network. 7, 8, 26

ROC Receiver operating characteristic. 9, 10, 23

RoI Region of Interest. 12

RPN Region Proposal Network. 12

SVM Support Vector Machine. 11, 20

TPR True positive rate. 9, 10

Bibliography

[Bab+14] Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. „Neural Codes for Image Retrieval“. In: *European Conference on Computer Vision*. 2014, pp. 584–599 (cit. on p. 19).

[BB15] Sean Bell and Kavita Bala. „Learning Visual Similarity for Product Design with Convolutional Neural Networks“. In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), p. 98 (cit. on p. 19).

[Bom69] Sebastino Bombelli. *Retrato Del Procurador Girolamo Querini*. 1669 (cit. on p. 21).

[Che+10] Gal Chechik, Varun Sharma, Uri Shalit, and Samy Bengio. „Large Scale Online Learning of Image Similarity through Ranking“. In: *Journal of Machine Learning Research* 11.Mar (2010), pp. 1109–1135 (cit. on p. 19).

[Che+14] Xianjie Chen, Roozbeh Mottaghi, Xiaobai Liu, et al. „Detect What You Can: Detecting and Representing Objects Using Holistic Models and Body Parts“. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014 (cit. on p. 26).

[Cho+15] François Chollet et al. *Keras: Deep Learning Library for Theano and Tensorflow*. 2015 (cit. on p. 21).

[CL15] Ju-Chin Chen and Chao-Feng Liu. „Visual-Based Deep Learning for Clothing from Large Database“. In: *Proceedings of the ASE BigData & SocialInformatics 2015*. 2015, p. 42 (cit. on p. 19).

[CPC16] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. „An Analysis of Deep Neural Network Models for Practical Applications“. In: *arXiv:1605.07678 [cs]* (2016) (cit. on p. 12).

[Dai+16] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. „R-FCN: Object Detection via Region-Based Fully Convolutional Networks“. In: *Advances in Neural Information Processing Systems (NIPS)*. 2016, pp. 379–387 (cit. on p. 18).

[Don+16] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. „Image Super-Resolution Using Deep Convolutional Networks“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.2 (2016), pp. 295–307 (cit. on p. 12).

[Dos+14] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. „Discriminative Unsupervised Feature Learning with Convolutional Neural Networks“. In: *Advances in Neural Information Processing Systems (NIPS)*. 2014, pp. 766–774 (cit. on pp. 21, 22).

[DT05] Navneet Dalal and Bill Triggs. „Histograms of Oriented Gradients for Human Detection“. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 1. 2005, pp. 886–893 (cit. on p. 28).

[DV16] Vincent Dumoulin and Francesco Visin. „A Guide to Convolution Arithmetic for Deep Learning“. In: *arXiv:1603.07285 [cs, stat]* (2016) (cit. on p. 11).

[Eve+10] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. „The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results“. In: (2010) (cit. on p. 26).

[Fel+10] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. „Object Detection with Discriminatively Trained Part-Based Models“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.9 (2010), pp. 1627–1645 (cit. on p. 28).

[FMR08] Pedro F. Felzenszwalb, David A. McAllester, and Deva Ramanan. „A Discriminatively Trained, Multiscale, Deformable Part Model“. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2008 (cit. on p. 25).

[Gir+14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. „Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation“. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 580–587 (cit. on p. 17).

[Gir15] Ross Girshick. „Fast R-CNN“. In: *International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448 (cit. on p. 17).

[He+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. „Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification“. In: *International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034 (cit. on p. 3).

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. „Deep Residual Learning for Image Recognition“. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778 (cit. on pp. 12, 13).

[He+17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. „Mask R-CNN“. In: *arXiv preprint arXiv:1703.06870* (2017) (cit. on p. 18).

[HS15] Kaiming He and Jian Sun. „Convolutional Neural Networks at Constrained Time Cost“. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 5353–5360 (cit. on p. 12).

[IS15] Sergey Ioffe and Christian Szegedy. „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift“. In: *arXiv:1502.03167 [cs]* (2015) (cit. on p. 9).

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. „ImageNet Classification with Deep Convolutional Neural Networks“. In: *Advances in Neural Information Processing Systems (NIPS)*. 2012, pp. 1097–1105 (cit. on pp. 12, 17, 21).

[KV15] Neal Khosla and Vignesh Venkataraman. „Building Image-Based Shoe Search Using Convolutional Neural Networks“. In: *CS231n course project reports* (2015) (cit. on p. 19).

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. „Deep Learning“. In: *Nature* 521.7553 (2015), pp. 436–444 (cit. on p. 4).

[LeC+90] Yann LeCun, Bernhard E. Boser, John S. Denker, et al. „Handwritten Digit Recognition with a Back-Propagation Network“. In: *Advances in Neural Information Processing Systems (NIPS)*. 1990, pp. 396–404 (cit. on p. 8).

[Lin+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, et al. „Microsoft COCO: Common Objects in Context“. en. In: *European Conference on Computer Vision (ECCV)*. 2014, pp. 740–755 (cit. on p. 12).

[Low04] David G. Lowe. „Distinctive Image Features from Scale-Invariant Keypoints“. In: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110 (cit. on p. 28).

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. „Fully Convolutional Networks for Semantic Segmentation“. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3431–3440 (cit. on pp. 10, 11).

[MGE11] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A. Efros. „Ensemble of Exemplar-Svms for Object Detection and Beyond“. In: *International Conference on Computer Vision (ICCV)*. 2011, pp. 89–96 (cit. on p. 25).

[ODO16] Augustus Odena, Vincent Dumoulin, and Chris Olah. „Deconvolution and Checkerboard Artifacts“. In: *Distill* (2016) (cit. on p. 11).

[Qia99] Ning Qian. „On the Momentum Term in Gradient Descent Learning Algorithms“. In: *Neural networks* 12.1 (1999), pp. 145–151 (cit. on p. 5).

[Red+16] Joseph Redmon, Santosh Divvala, Ross Girshick, et al. „You Only Look Once: Unified, Real-Time Object Detection“. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788 (cit. on p. 18).

[Ren+15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. „Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks“. In: *Advances in Neural Information Processing Systems (NIPS)*. 2015, pp. 91–99 (cit. on pp. 18, 24).

[RHW88] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. „Learning Representations by Back-Propagating Errors“. In: *Cognitive modeling* 5.3 (1988), p. 1 (cit. on p. 6).

[Rus+15] Olga Russakovsky, Jia Deng, Hao Su, et al. „ImageNet Large Scale Visual Recognition Challenge“. en. In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252 (cit. on pp. 12, 20).

[Shi+16] Wenzhe Shi, Jose Caballero, Lucas Theis, et al. „Is the Deconvolution Layer the Same as a Convolutional Layer?“ In: *arXiv:1609.07009 [cs]* (2016) (cit. on p. 10).

[SLD16] Evan Shelhamer, Jonathan Long, and Trevor Darrell. „Fully Convolutional Networks for Semantic Segmentation“. In: *arXiv:1605.06211 [cs]* (2016) (cit. on pp. 12, 20).

[Spr+14] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. „Striving for Simplicity: The All Convolutional Net“. In: *arXiv:1412.6806 [cs]* (2014) (cit. on p. 9).

[SZ14] Karen Simonyan and Andrew Zisserman. „Very Deep Convolutional Networks for Large-Scale Image Recognition“. In: *arXiv:1409.1556 [cs]* (2014) (cit. on pp. 10, 12).

[vonnd] Eike von Repgow. *Heidelberger Bilderhandschrift Des Sachsenspiegels - Universitätsbibliothek Heidelberg, Cod. Pal. Germ. 164*. [Begin 14th. c. (Date according to written find)] (cit. on p. 36).

[Wan+14] Jiang Wang, Yang Song, Thomas Leung, et al. „Learning Fine-Grained Image Similarity with Deep Ranking“. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 1386–1393 (cit. on p. 19).

[Wu+13] Pengcheng Wu, Steven CH Hoi, Hao Xia, et al. „Online Multimodal Deep Similarity Learning with Application to Image Retrieval“. In: *Proceedings of the 21st ACM International Conference on Multimedia*. 2013, pp. 153–162 (cit. on p. 19).

[ZC88] Y. T. Zhou and R. Chellappa. „Computation of Optical Flow Using a Neural Network“. In: *IEEE International Conference on Neural Networks*. Vol. 1998. 1988, pp. 71–78 (cit. on p. 8).

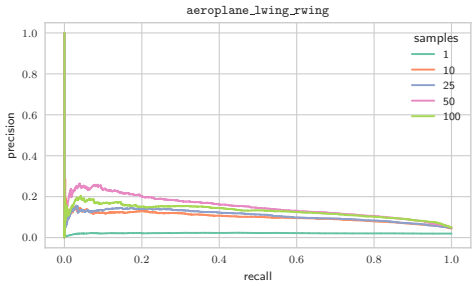
[Zei+10] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Robert Fergus. „Deconvolutional Networks“. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2010, pp. 2528–2535 (cit. on p. 10).

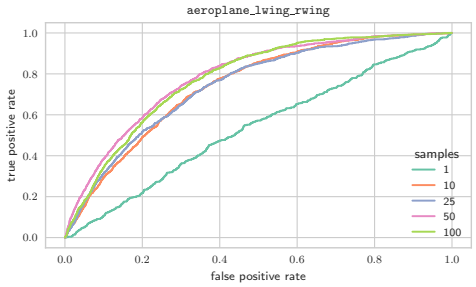
Appendix A

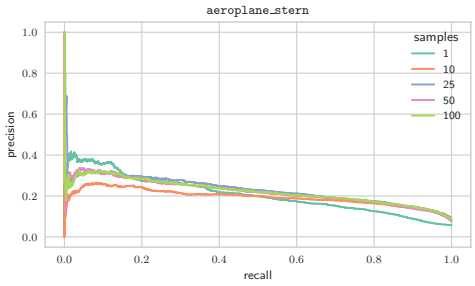
A.1 PASCAL-Part class-part combinations

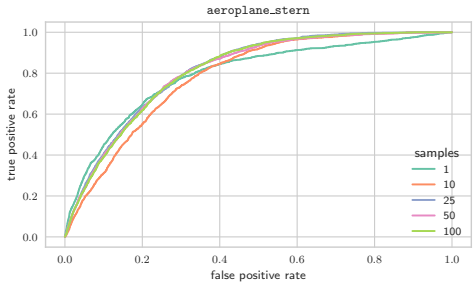
Tag	Classes	Parts
aeroplane_lwing_rwing	Plane	Left wing, right wing
aeroplane_stern	Plane	Stern (Rudder)
bicycle_bwheel_fwheel	Bicycle	Back wheel, front wheel
bird_beak	Bird	Beak
bird_head	Bird	Head
bird_lwing_rwing	Bird	Left wing, right wing
bird_tail	Bird	Tail
bottle_body	Bottle	Body
bus_car_bliplate_fliplate	Bus, Car	Front and back license plate
car_door	Car	Door
cow_sheep_lhorn_rhorn	Cow, Sheep	Left horn, right horn
motorbike_bwheel_fwheel	Motorbike	Front wheel, back wheel
person_hair	Person	Hair
person_head	Person	Head
person_lear_rear	Person	Left ear, right ear
person_lfoot_rfoot	Person	Left foot, right foot
person_lhand_rhand	Person	Left hand, right hand
person_neck	Person	Neck
person_torso	Person	Torso
pottedplant_plant	Potted plant	Plant
train_coach	Train	Coach (normal carriage)
train_head	Train	Head (motor coach)
train_hfrontside	Train	Head front side

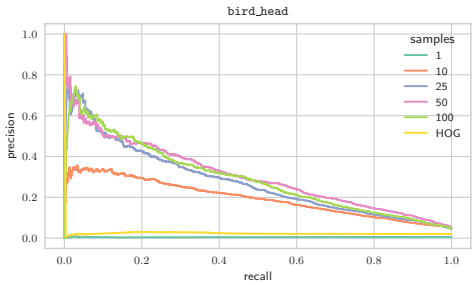
A.2 PASCAL-Part test results

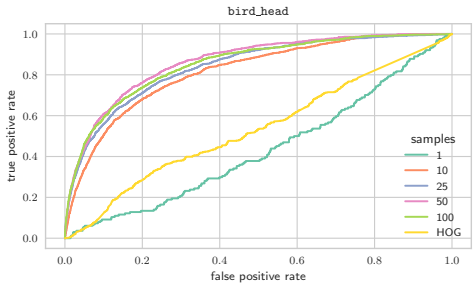


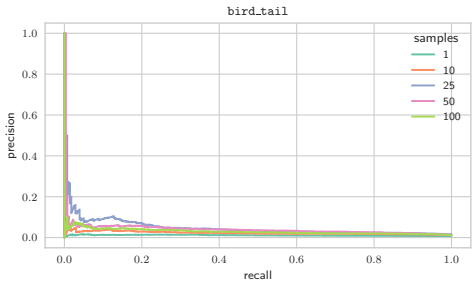


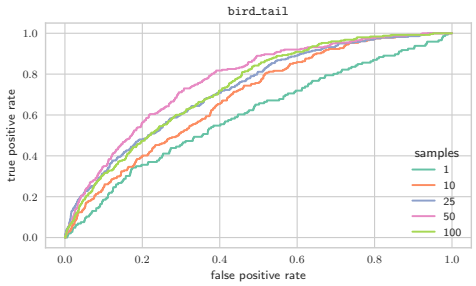


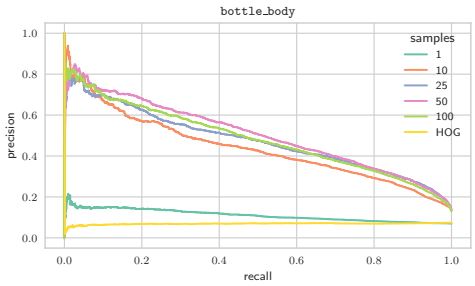


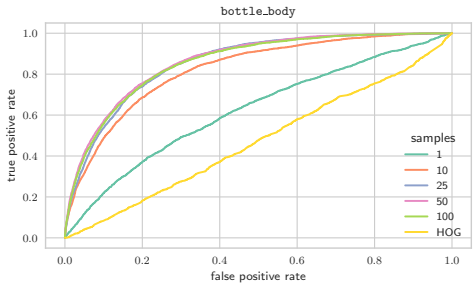


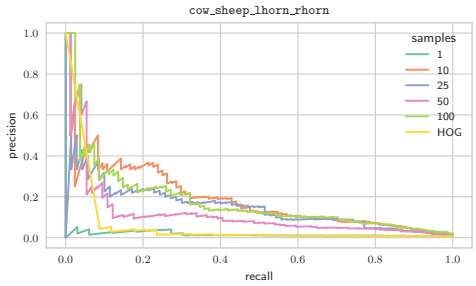


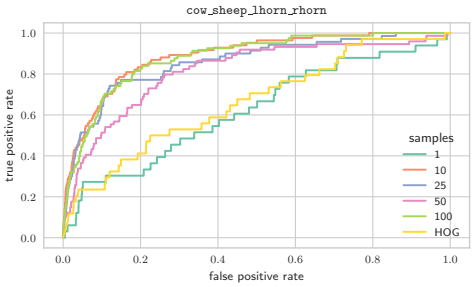


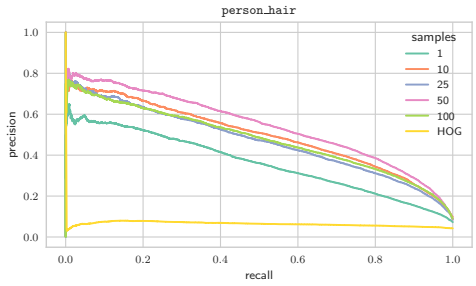


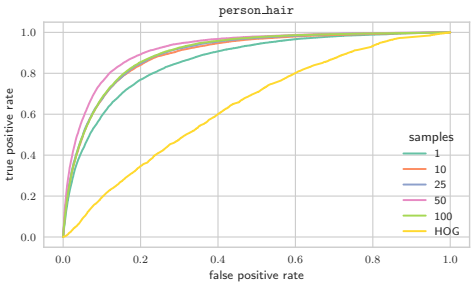


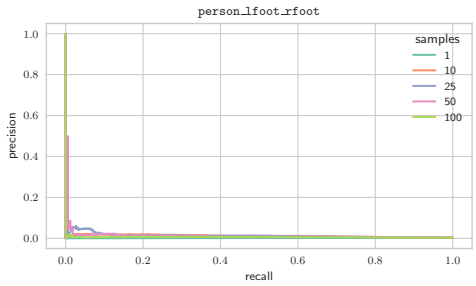


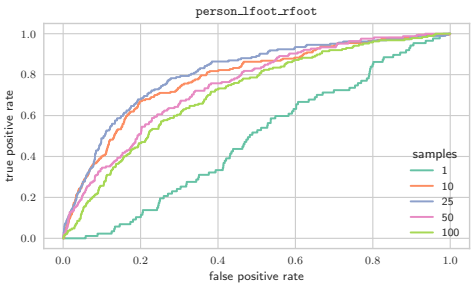

cow_sheep_lhorn_rhorn
precision
recall
samples
1
10
25
50
100
HOG

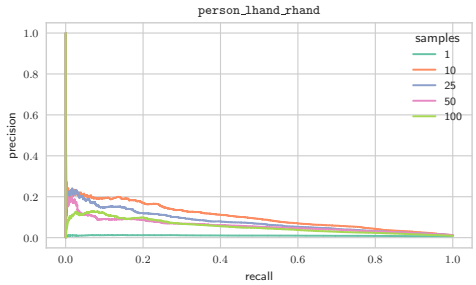

cow_sheep_lhorn_rhorn
true positive rate
false positive rate
samples
1
10
25
50
100
HOG

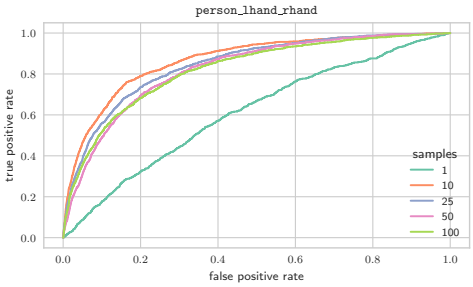

person_hair
precision
recall
samples
1
10
25
50
100
HOG

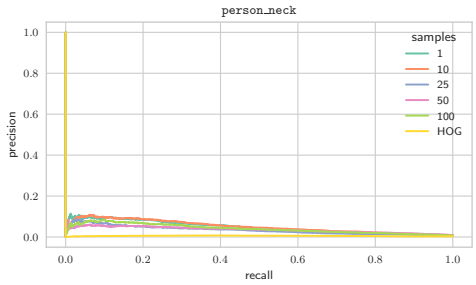

person_hair
true positive rate
false positive rate
samples
1
10
25
50
100
HOG

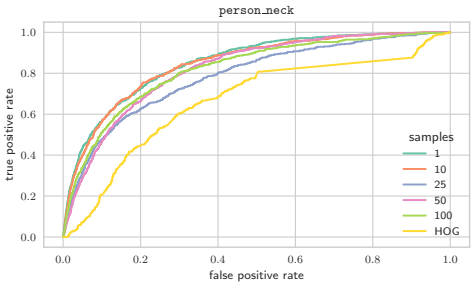

person_lfoot_rfoot
precision
recall
samples
1
10
25
50
100

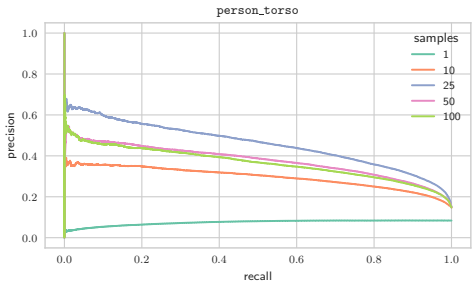

person_lfoot_rfoot
true positive rate
false positive rate
samples
1
10
25
50
100

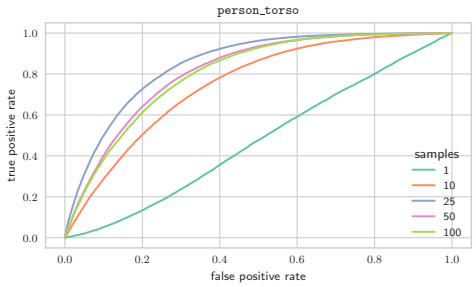

person_lhand_rhand
precision
recall
samples
1
10
25
50
100

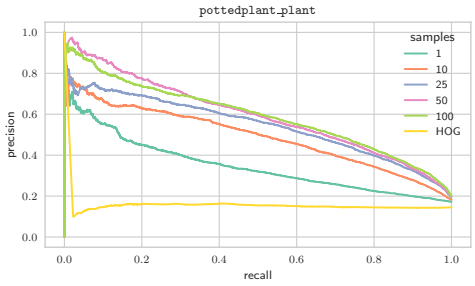

person_lhand_rhand
true positive rate
false positive rate
samples
1
10
25
50
100

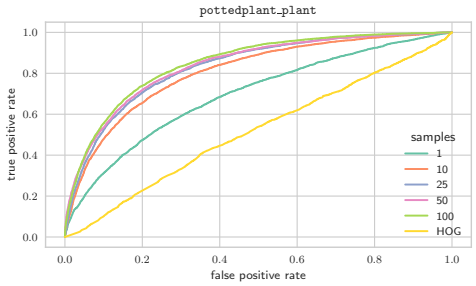

person_neck
precision
recall
samples
1
10
25
50
100
HOG

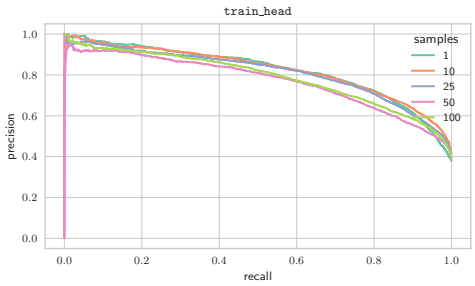

person_neck
true positive rate
false positive rate
samples
1
10
25
50
100
HOG

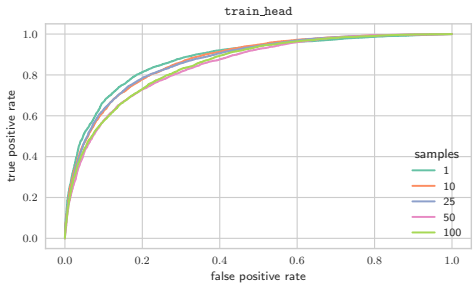

person_torso
precision
recall
samples
1
10
25
50
100


person_torso
true positive rate
false positive rate
samples
1
10
25
50
100


pottedplant_plant
precision
recall
samples
1
10
25
50
100
HOG


pottedplant_plant
true positive rate
false positive rate
samples
1
10
25
50
100
HOG


train_head
precision
recall
samples
1
10
25
50
100


train_head
true positive rate
false positive rate
samples
1
10
25
50
100