Institutionen för systemteknik
Department of Electrical Engineering

Examensarbete

Pedestrian detection using convolutional neural networks

Examensarbete utfört i Bildbehandling
vid Tekniska högskolan vid Linköpings universitet
av

David Molin

LiTH-ISY-EX–15/4855–SE

Linköping 2015

Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Electrical Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings tekniska högskola
Linköpings universitet
581 83 Linköping

Pedestrian detection using convolutional neural networks

Examensarbete utfört i Bildbehandling
vid Tekniska högskolan vid Linköpings universitet
av

David Molin

LiTH-ISY-EX–15/4855–SE

Handledare: **Martin Danelljan**
ISY, Linköpings universitet
David Forslund
Autoliv

Examinator: **Fahad Khan**
ISY, Linköpings universitet

Linköping, 12 juni 2015

Avdelning, Institution
Division, Department

CVL
Department of Electrical Engineering
SE-581 83 Linköping

Datum
Date

2015-06-12

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English
☐ _____

Rapporttyp
Report category

☐ Licentiatavhandling
☒ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ _____

ISBN
__

ISRN
LiTH-ISY-EX–15/4855–SE

Serietitel och serienummer
Title of series, numbering

ISSN
__

URL för elektronisk version
-

Titel
Title

Detektering av fotgängare med hjälp av djupa nät
Pedestrian detection using convolutional neural networks

Författare
Author

David Molin

Sammanfattning
Abstract

Automatisk detektering av fotgängare är ett viktigt forskningsområde med tillämpningar inom aktiva trafiksäkerhetssystem samt autonoma fordon. I och med att aktiva trafiksäkerhetssystem samt autonoma fordon är på väg att bli tekniskt gångbara så har intresset för dessa ökat markant de senaste åren.

Syftet med detta examensarbete har varit att undersöka hur väl convolutional neural networks (CNN) lämpar sig för automatisk detektering av fotgängare. Anledningen till denna studie är att CNN har applicerats på flera problem inom datorseende, med goda resultat. Automatisk detektering av fotgängare har sina huvudsakliga tillämpningar i realtidssystem. Av denna anledning så studerar detta examensarbete även strategier för att reducera beräknings komplexiteten för CNN.

Tillvägagångsättet som har används i detta examensarbete för att detektera fotgängare är att med hjälp av en CNN hitta en sannorlikhetskarta över vart fotgängare finns. Från denna skapas markeringar för fotgängarna i bilden är.

En metod för att hantera skalinvarians för detektionen har även blivit utvecklad för detta examensarbete. Experiment visar att denna metod ger avsevärt bättre resultat för automatisk detektering av fotgängare.

Den prestanda som detta examensarbete har uppnått är likvärdig med vissa andra artiklar som använder CNN.

Nyckelord
Keywords

Convolutional neural networks, pedestrian detection, caltech pedestrian dataset

Sammanfattning

Automatisk detektering av fotgängare är ett viktigt forskningsområde med tillämpningar inom aktiva trafiksäkerhetssystem samt autonoma fordon. I och med att aktiva trafiksäkerhetssystem samt autonoma fordon är på väg att bli tekniskt gångbara så har intresset för dessa ökat markant de senaste åren.

Syftet med detta examensarbete har varit att undersöka hur väl convolutional neural networks (CNN) lämpar sig för automatisk detektering av fotgängare. Anledningen till denna studie är att CNN har applicerats på flera problem inom datorseende, med goda resultat. Automatisk detektering av fotgängare har sina huvudsakliga tillämpningar i realtidssystem. Av denna anledning så studerar detta examensarbete även strategier för att reducera beräknings komplexiteten för CNN.

Tillvägagångsättet som har används i detta examensarbete för att detektera fotgängare är att med hjälp av en CNN hitta en sannorlikhetskarta över vart fotgängare finns. Från denna skapas markeringar för fotgängarna i bilden är.

En metod för att hantera skalinvarians för detektionen har även blivit utvecklad för detta examensarbete. Experiment visar att denna metod ger avsevärt bättre resultat för automatisk detektering av fotgängare.

Den prestanda som detta examensarbete har uppnått är likvärdig med vissa andra artiklar som använder CNN.

Abstract

Pedestrian detection is an important field with applications in active safety systems for cars as well as autonomous driving. Since autonomous driving and active safety are becoming technically feasible now the interest for these applications has dramatically increased.

The aim of this thesis is to investigate convolutional neural networks (CNN) for pedestrian detection. The reason for this is that CNN have recently been successfully applied to several different computer vision problems. The main applications of pedestrian detection are in real time systems. For this reason, this thesis investigates strategies for reducing the computational complexity of forward propagation for CNN.

The approach used in this thesis for extracting pedestrians is to use a CNN to find a probability map of where pedestrians are located. From this probability map bounding boxes for pedestrians are generated.

A method for handling scale invariance for the objects of interest has also been developed in this thesis. Experiments show that using this method gives significantly better results for the problem of pedestrian detection.

The accuracy which this thesis has managed to achieve is similar to the accuracy for some other works which use CNN.

Acknowledgments

I would like to thank all contributors to the Caffe library, by creating this library you have made it possible for people to use CNN in research and other projects without having to write plenty of code just to implement the basic layers. I would also like to thank my supervisors and examiner for their help with this thesis.

Linköping, June 2015
David Molin

Contents

1 Introduction

Pedestrian detection is the problem of detecting and locating pedestrians in an image or video sequence. This field of research is important since autonomous cars are under development and are deployed on an experimental level. Developing pedestrian detectors with a high accuracy is essential for autonomous cars since poor performance in this field could lead to injuries or casualties in traffic. The field of pedestrian detection is also similar to many other fields in computer vision such as object detection, object classification and object localization. Therefore advancements done in pedestrian detection might generalize to these other fields.

1.1 Prior work

This section gives a short description of the theory behind convolutional neural networks (CNN). This section also gives a brief overview of the methods used for pedestrian detection and some background on other methods which have used CNN for pedestrian detection.

1.1.1 Pedestrian detection

It is possible to approach the problem of pedestrian detection in multiple ways. Methods usually fall under the umbrella term machine learning. Many methods use hand crafted features such as HOG features [1] or randomly generated low level features such as Haar features [2]. These features are then used to train the model which does the classification. Other approaches include using several simple classifiers to make a strong classifier, for example Ada boost [3]. Another common approach is to have the classifier learn the features [4]. CNN belong to this type of classifier.

Recently the interest for CNN has increased since this method has managed to achieve high performance in several different fields of computer vision. The most notable results are for general classifications tasks such as ILSVRC [5]. As a result some works have tried apply CNN for pedestrian detection, with mixed results. Sermanet et al. [6] got inferior performance compared to state of art for pedestrian detection on Caltech pedestrian dataset. New methods such as Tian et al. [7] got better results than previous state of art. However, Tian et al. used additional datasets for training as well as using additional lables for the pedestrians in the Caltech dataset. For this reason it is hard to compare their results with other methods.

1.1.2 CNN

CNN is a variant of neural network with the constraint that weights are shared for shifted versions of the hidden layer. One interpretation is that the activations for a hidden layer are convolved with a filter in order to produce the activations for the next layer. Non linear activation functions are placed between the convolution layers in order to introduce non-linearities in the classification. Other layers are used as well such as max pooling, in order to make the results less affected by small translations.[1]

1.1.2.1 History

CNN date back to the 1980 when Neocognition was published [8]. During the 90's it became popular due to being able to perform optical character recognition, among other things, very well [9]. Recently interest in CNN have increased since it has managed to outperform other methods in several computer vision applications. The reason is likely that more computational power as well as more training data is available. The discovery of the rectified linear unit (ReLU) activation function also helped to increase the performance of CNN [10].

1.1.2.2 Forward & backward propagation

Supervised training of CNN is done by forward and backward propagation. During the forward propagation phase a training sample is sent through the CNN to compute an output value. In the backward propagation step this value is then compared to a ground truth. The error is then propagated backwards through the net to update the parameters of the network.

1.1.2.3 Initialization of weights

Random initialization

When initializing the weights for a neural networks, random weights are sometimes used. The reason behind using random weights as opposed to initializing

[1] For a more detailed introduction to CNN see Appendix A.

all weights by some constant is that using random weights breaks the symmetry of the network. If constant weights are used then all weights to each hidden node will be updated in the same way, therefore the resulting network would be equivalent of using only one hidden node for each layer of the net. In this thesis random weights will be used as initialization for all layers where there are no weights available from another trained network. These layers correspond to the layers described in Section 2.2.1

Unsupervised learning for initializaton

Unsupervised learning is sometimes used to initialize the weights for the network. This means that the weights are initialized in such a way that they capture some of the underlying data structure instead of being purely random. Using unsupervised learning has shown to boost the performance [11].

Using a model trained for an other task

Another approach for initializing a CNN is to take the architecture and weights from an network which is already trained for another task. This is used for multiple different tasks with good results [12]. This method will be used for initializing many of the weights in this thesis.

1.1.2.4 Results for CNN for other tasks

The performance of CNN architectures are usually compared by studying the performance on standardized classifications tasks such as ILSVRC. This data set contain 1000 of object classes.

1.1.2.5 Output of CNN

The CNN are commonly used by applying them on an image patch of a fixed size. The CNN will then map this image patch to one label. This is done by first sending the input through several layers and then sending this result through a fully connected network in the end. Other approaches can also be used for this last step, one such method is to use support vector machines on the activations for the last layer. Using fully connected networks in the end is for example done in Szegedy et al. [13]. Support vector machines is for example used in Razavian et al. [12]. Another way is to use a sliding window approach where a larger image is sent through the CNN and the output is a label map instead of just one label. For example Sermanet et al. and Oquab et al. use this approach for evaluation [14] [15].

If one takes a subimage of a given size $W_{in} \times H_{in}$, then the input to the fully connected layer will get an input of size $W_{fc} \times H_{fc}$. Then the first fully connected layer is equivalent to a convolution layer with kernel size $W_{fc} \times H_{fc}$ and the rest of the layers are of size 1×1. Therefore these two methods are closely related.

1.1.2.6 Weight updating policy

The problem of optimizing the weights of a neural network is a non-convex problem. For this reason stochastic gradient descent (SGD) or a similar algorithm is used. There exist several variations such as momentum methods and Nesterov's accelerated gradient (NAG). [16] shows the relation in between classical momentum methods and NAG.

1.1.3 Caltech pedestrian dataset

A popular test datset for pedestrian detection is the Caltech pedestrian dataset [17]. This dataset contains approximately 10 hours of labled video. Half of the data is reserved for evaluating performance and the other half is for training a model. This dataset also has a standardized method for evaluation. For these reasons this dataset will be used for this thesis. Another notable dataset which also is quite popular is the ETH dataset [18].

1.1.4 Caffe

For this thesis a modified version of the library Caffe [19] has been used for training the CNN and evaluating the performance. Caffe is an open source library for using CNN created by Yangqing Jia et al. The reason for using caffe for this thesis is that most standard and some non-standard layers are already implemented. Also most of the layers have a GPU implementation which allows for a fast execution. All of the code for compiling caffe is also available which makes it easy to modify.

1.2 Problem formulation

The aim of this thesis is to design an approach based on CNN which can do pedestrian detection in close to real time. This thesis will investigate the problem of training and evaluating a CNN by feeding it entire frames. The network will be trained to output label maps corresponding to the objects present for several different scales. From the output of the CNN it will be possible to extract bounding boxes directly. This framework will be applied to the problem of pedestrian detection. The Caltech pedestrian dataset is used for evaluation as well as training data.

1.3 Contributions

This thesis presents a CNN framework for detecting pedestrians by using whole video frames as input to the CNN. In this thesis several novel approaches for this problem are investigated.

Several works use loss layers at multiple depths of the network [13] [20]. These methods use several output layer, but all of these layers try to perform

the same task. This thesis proposes a novel approach where several output layers at multiple depths are used for detecting objects for different scales. This thesis also shows that doing this improves the results for the problem of pedestrian detection significantly. Methods such such as Sermanet et al. [14] use a sliding window approach in order extract an output from the network. This thesis investigates the use of sliding window training for CNN based object detection.

2

Method

In this chapter, the method used to detect pedestrians is described. This method will use CNN. The objective of this CNN is to output a probability map of where there are pedestrians present, as well as the offsets for bounding box candidates. The training of CNN is described in Section 2.4. Section 2.3 describes how the ground truth used in this training phase is generated. For the evaluation phase input images will be sent through the network. Section 2.5 describes how bounding box candidates are computed from the output of the network. The architechture of the network is described in Section 2.2. Section 2.1 shows how the approach used in this thesis relates to classifying region proposals.

2.1 Sliding window evaluation

A common approach for object detection is to extract region proposals and then resizing the contents of those region proposals to a fixed size [13] [6]. This image is then sent through a classifier to give a predicted class for that region.

Evaluation of two adjacent regions for CNN share a lot of computations. For this reason this thesis will explore the possibility using a sliding window approach when classifying images. In other words large images are sent through a CNN in order to get predictions of several adjacent regions. This approach has been explored to some extent for the evaluation phase in [14]. During the training phase backward and forward propagation of two adjacent datapoints will share a lot of computations too so it should be possible to speed up the training as well by training the layers in this way. Figure 2.1 illustrates the difference between sliding window evaluation and classifying region proposals.

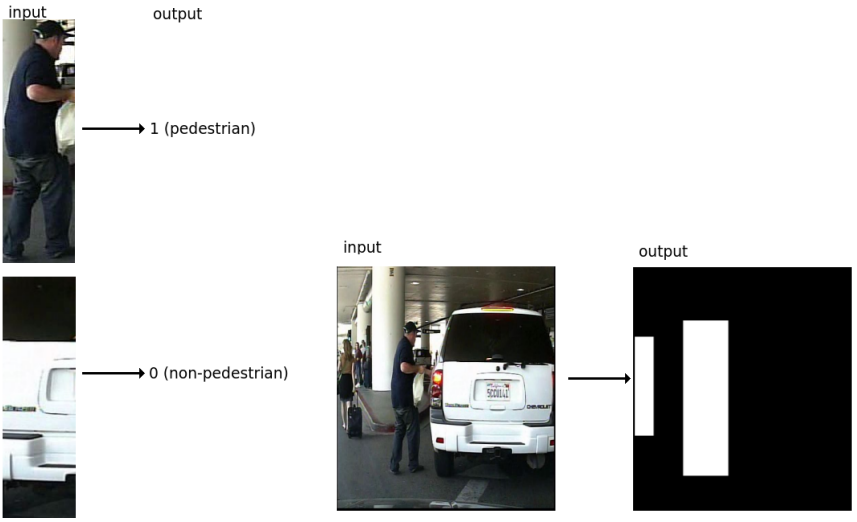


Figure 2.1: *This figure shows the difference between mapping a subimage to a single label (Figure 2.1a) and using the entire image to create a label map (Figure 2.1b). When a subimage is mapped to a single label the output of the network will be the estimated probability that the subimage contains an pedestrian. For the case where the output is a label map the desired output will be an grayscale image with pixel values between 0 and 1. The value of each pixel correspond to the probability that there is an pedestrian present at that location. In Figure (2.1b) the desired output is shown. At locations where there is a pedestrian present the desired output is 1 (white). At locations where there are not any pedestrians the desired output is 0 (black)*

2.2 Network architecture

Since the architecture of googLeNet [13] was very successful in ILSVRC2014 [5] much of the architecture of the CNN used in this thesis will be an adaption of that architecture. The only modification of the overall architecture is that layers for extracting the desired output has been added at the end of the network as described in Section 2.2.2, or at multiple depths as described in Section 2.2.3.

2.2.1 Output extraction

This section will describe the layers added at a given depth of the network in order to give an output similar to the ground truth described in Section 2.3.2 These layers will extract both the presence of a pedestrian as well as the offsets for the bounding box. A visualization of these layers can be found in Figure 2.2

2.2.1.1 Extracting presence of pedestrian

In order to extract the presence of a pedestrian the activations at a given depth of the network is convolved with multiple 5×5 kernels producing 64 channels. A ReLU is applied to the result, then this result is convolved with another set of 5×5 kernels producing 1 channel as output. At the end, a sigmoid layer is used to map this to a result to [0,1]. During the training the output will be compared to the ground truth label in a weighted cross entropy layer, using the weights described in Section 2.3.2

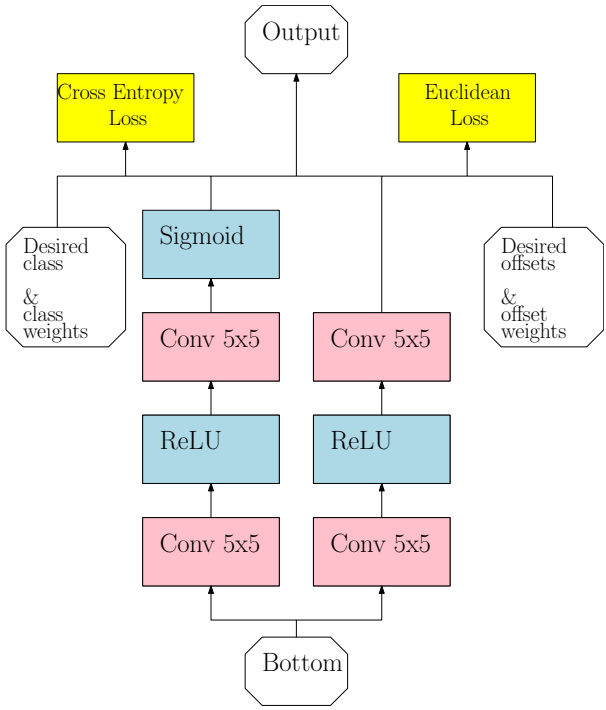


Figure 2.2: *A visualization of the layers added at a given depth to extract an output. The boxes labled Conv correspond to convolution layers. These are described in more detail in Appendix A. The numbers* 5×5 *means that* 5×5 *kernels are used for these layers. Note that the loss layers are only used during the training phase and reading the output directly is only used in the evaluation phase.*

2.2.1.2 Extracting bounding box offsets

The reason why it is desirable to get an output with bounding box offsets is that this can capture some of the scale variations for detected pedestrians and generate a bounding box of a suitable size. Another reason is that the probability output generally is the largest at some location which is not the center of the

pedestrian. These offsets can therefore be used to place the center of the output bounding box at a more suitable location. The offsets to the edges of the bounding box is found by convolving the same activations as used as inputs to the layers in Section 2.2.1.1 with multiple 5x5 kernel producing 64 channels. A ReLU is applied to the result. This result is then convolved with another 5x5 kernel producing a four channel output. During the training the output will be compared to the ground truth bounding box regression values in a weighted euclidean loss layer, using the weights described in Section 2.3.2

2.2.2 Using one output level

One approach tested in this thesis is to have one output levels at the end of the network and have these layers output bounding boxes for all scales, without using the splitting of sizes described in Section 2.3.2.3. A visualization of this network can be seen in Figure 2.3

2.2.3 Using multiple output levels at various depths

Another approach which have been tested is to have multiple output levels at different depths of the network. For a justification why this is used see Section 2.3.2.3 The outputs extracted early in the net correspond to pedestrians far away and the outputs extracted at the top correspond to pedestrians close to the vehicle. The ground truth used for training this network is generated by using the splitting of sizes described in Section 2.3.2.3. A visualization of this network can be seen in Figure 2.4

2.3 Training data

This section will describe in more detail how the training data is generated in order to be able to train a network which does the desired task as described in the introduction to this chapter.

2.3.1 Caltech pedestrian dataset

For training as well as testing the Caltech dataset will be used. This dataset is divided into a training set and an evaluation set. There is no overlap in between these sets, since this would give a positive bias to the results.

The data set contains frames from a vehicle mounted camera. The dataset also contains bounding boxes for different types of pedestrians in these frames.

There are three different classes of bounding boxes: "pedestrian", "pedestrian?" and "people". The label "pedestrian" is used for pedestrians which are obviously of this class. The label "people" correspond to groups of people who are so close to one another that is hard to label them individually. The label "pedestrian?" correspond to objects which are very hard to determine whether they are pedestrians or not. In Figure 2.5 an example of what an input frame looks like and how bounding boxes are placed can be seen.

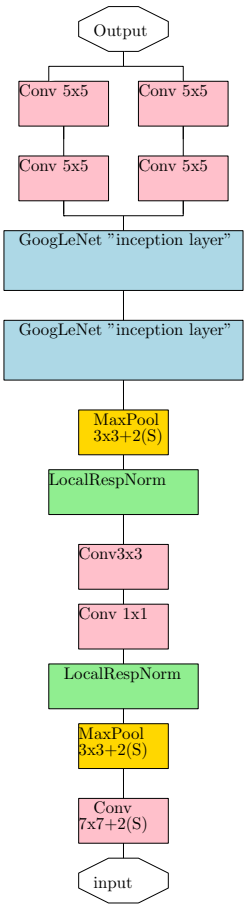


Figure 2.3: *Net architecture for the case where only one output level is used. Each box correspond to a different type of layer. The $m \times m$ correspond to the size of the kernel for this layer. +2(S) correspond to using a downsampling by a factor of 2 after this layer. The boxes labled conv correspond to convolution layers. The boxes labled maxPool are max pooling layers. The boxes labled LocalRespNorm are local response normalization layers. What these layers do is described in more detail in Appendix A. The inception layer is just a group of normal convolution layers and pooling layers in a configuration described in [13]. Note that all ReLU layers are not shown here, but they are placed after almost all convolution layers. Each layer uses the output from the layer below them as input.*

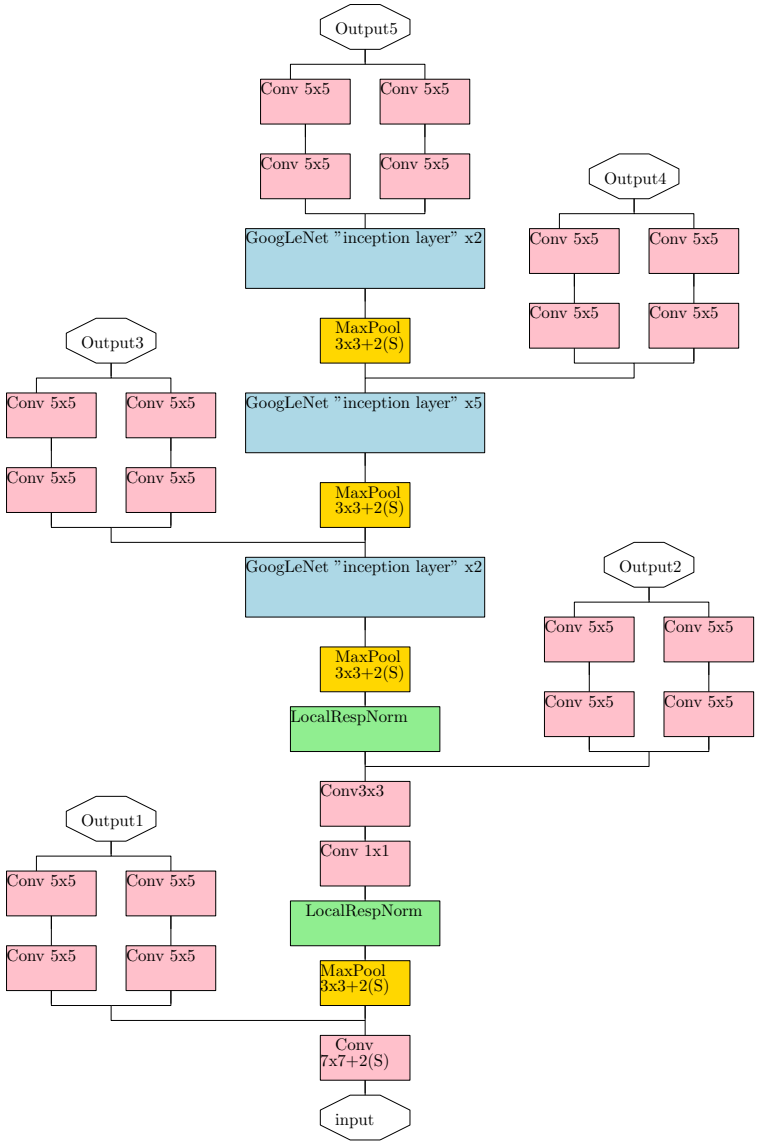


Figure 2.4: *Net architecture for the case where multiple output level is used. Each box correspond to a different type of layer. The $m \times m$ correspond to the size of the kernel for this layer. +2(S) correspond to using a downsampling by a factor of 2 after this layer. The boxes labled conv correspond to convolution layers. The boxes labled maxPool are max pooling layers. The boxes labled LocalRespNorm are local response normalization layers. What these layers do is described in more detail in Appendix A. The inception layer is just a group of normal convolution layers and pooling layers in a configuration described in [13]. Note that all ReLU layers are not shown here, but they are placed after almost all convolution layers. Each layer uses the output from the layer below them as input.*

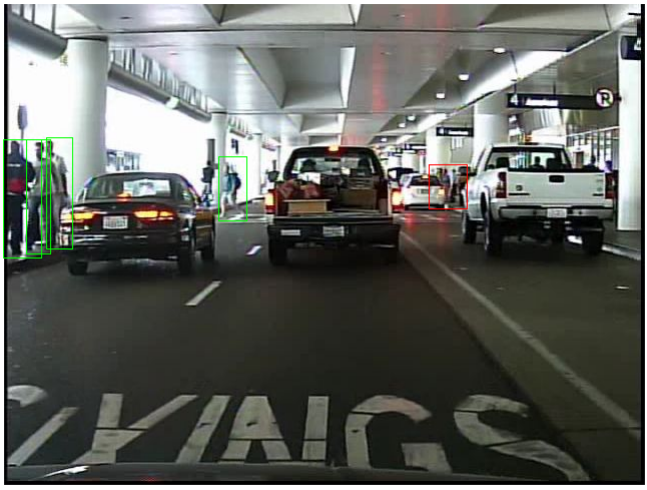


Figure 2.5: *This figure illustrates what an input frame from Caltech dataset might look like. 'pedestrian' bounding boxes are green. the label 'pedestrian?' is red. The size of the image size is 640x480*

2.3.2 Ground truth

In order to be able to use sliding window training the size of the ground truth has to match the size of the output from the CNN. The network used in this thesis use downsampling after several of the layers. For this reason the ground truth will be smaller than the input image.

The labels for all pixels inside a bounding box with the label "pedestrian" is set to be 'pedestrian', all other pixels will get the label 'non-pedestrian'. This can be seen in Figure 2.1b.

2.3.2.1 Classification weights

For several reasons discussed in this section, the weights assigned to the different desired outputs for the training are not set to be equal. The reason for this is that the different classes occur with different frequency, also some areas have ambiguous ground truth labels. This section will describe how these weights are determined and why they are set to these values.

Inverse probability weighting

One downside with using sliding window training is that the training set will usually be unbalanced. The reason being that some classes are more common in the input images. For example in pedestrian detection almost all frames will contain fewer pixels corresponding to pedestrians compared to background.

For this reason inverse probability weighting will be used for the different classes. That means that a training example of class y_n will get the weight

$$W_n = \frac{1}{P(Y = y_n)} \tag{2.1}$$

where Y is the random variable corresponding to the class of that area. This is similar to creating a new training by resampling with replacement in such a way that all classes are equally probable.

Lower weights at edges of a bounding box

At the edges of the bounding box the label will likely change. In other words a small translation at this location will result in a different desired output. It does not make sense to penalize incorrect outputs at the edges of the bounding box. For this reason the weights at the edges of the bounding box will have low weights and the weights in the middle of a bounding box will be larger.

For this thesis the weights were set to be proportional to the value of a two dimensional gaussian distribution. The mean is set to be in the middle of the bounding box. The standard deviation in the width direction is set to be the width of the bounding box and the standard deviation in the height direction was set to be the height of the bounding box. There is no theoretical justification for using a gaussian distribution other than that it gives higher values for the middle of the bounding box and small values at the edges.

Ignoring unclear cases

Weights for pixels inside bounding boxes with the label "person?" and "people" are set to be 0. The reason for this is that in the first case it is not clear whether the object is a pedestrian or not. For the second case it is not possible to get a bounding box corresponding to a single pedestrian.

Having equal total weights for all scales

Since outputs higher up in the network will be smaller due to downsampling, these output layers will get a higher weight to compensate for this. The number of pixels in an image decrease proportionally to the square of the downsampling factor. In order to compensate for this the weights are multiplied by the square of the downsampling factor.

Weights for pixels which are not in any bounding box

The weights for the pixels which are not in any bounding box is set to be a constant in such a way that the conditions described in Section 2.3.2.1 are satisfied.

2.3.2.2 Bounding box regression

Ground truth for regression

In order to train a network to find the offsets to the bounding boxes a corresponding ground truth has to be created. The desired output for this task is a vector in $\mathbb{R}^4$ for each pixel in the output label map. Each of the four values in this desired output correspond to the distance to the top, bottom, left and right side of the bounding box which this pixel is inside.

Weighting for regression

The network will produce offsets for bounding box candidates for all locations in the output label map. These values will not be relevant for training unless they are inside a bounding box. For this reason pixels outside a bounding box will get a weight of 0 during the training phase. In other words training will not be performed on these pixels. If there is an overlap of many bounding boxes these bounding boxes will also be assigned a weight of 0 for the regression. The reasoning behind this is that there are two correct answers for these cases.

2.3.2.3 Splitting sizes

Scale invariance is one of the major differences when comparing pedestrian detection with general classification. Pedestrians occur at many very different scales in the Caltech dataset. For this reason two different methods will be tested to see if there is any difference between the results. The first approach is to have one only one output level and let this output detect pedestrians for all scales. The second approach is to have several output levels. The task for each of these layers is to correctly classify pedestrians of a certain size.

There are several reasons why the second approach should be used. One reason is that each pooling layer loses accuracy for the localization of the features. Therefore the localization for pedestrians far away could be worse if the detection is done after too many downsamplings. Another reason for using multiple scales is that the problem of finding good bounding boxes become simpler. The reason for this is that there are upper and lower bounds for how large the bounding box can be for the different output levels.

In order to train the network for the second approach training data has to be generated so that the different layers only train on pedestrians of the correct size. This is done by applying a function on the height of each pedestrian for all training bounding boxes. This function will return a value in $\mathbb{R}^n$ where the value for the different indices correspond to the weight which this bounding box will get for the different output levels. n is the number of output levels for the net. In this thesis $n = 5$. A plot of this function can be seen in Figure 2.6. The size of pedestrians which a output layer should detect is loosely based on the size in the input image which contributes to the output at this level. All the output layers have two 5×5 convolution layers which means that they take a 9×9 area of their bottom layer into account for determining the output.

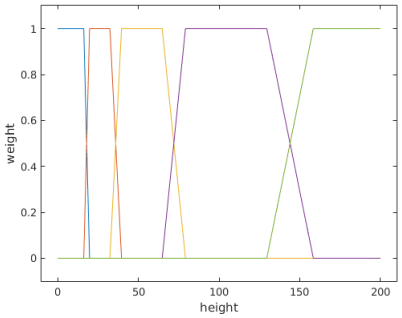


Figure 2.6: *This illustration shows which output level should try to find the location of a pedestrian given the height of the pedestrian. The blue line is the weights assigned for the first output level given the height. The orange line is the weights for the second output level. The yellow line is the weights for the third output level. The purple line is the weights for the fourth output level and the green line is the weights for the fifth output level. From this image we can for example see that the fourth level will try to detect pedestrians with a height between 50 and 160 pixels.*

The output level at depth i will produce a label map which is downsampled by a factor of 2^i compared to the input image. Therefore this level will try to find pedestrians of a smaller size than $9 * 2^i$.

In order to get a smoother transition for which output is supposed to find which pedestrian there will be a linear interpolation from $0.9*9*2^n$ and $1.1*9*2^n$. A visualization of this can be seen in Figure 2.7

2.3.3 Input data

The input to the network will be whole frames from the camera. The only requirement for producing a valid output is that the final size of the output should have an integer size. The input images from Caltech pedestrian dataset are 640×480. The only layers which reduce the size of the output for this network is the pooling layers. The output of the last pooling layer will be downsampled by a factor of 32×32. Therefore each frame from the Caltech dataset will be a valid input. If the output is of a size which is not a multiple of 32 then that input has to be resized to a multiple of 32.

2.4 Training

2.4.1 Sampling of training examples

In this thesis two different sampling methods are tested. The first method is to pick random frames from the training set at each iteration. The second approach

Figure 2.7: *Image showing the ground truths and weights used for the different layers. The pedestrians which are close to the camera are supposed to be found in the 5:th level, pedestrians further away are set to be found at lower levels. There are no pedestrians smaller than 40 pixels for this frame and for that reason the first two layers are not supposed to find any pedestrians in this frame.*

is to keep track of which frames which caused a large loss and then retraining on those samples.

For both of these methods images will be flipped in the width direction in order to get twice the training data.

2.4.1.1 Random sampling

This sampling policy picks a frame from the training set at random. Then a ground truth is extracted as described in Section 2.3 for this frame. This input and ground truth is then used for forward and backward propagation in order

to update the weights for the network. This sampling policy is similar to batch stochastic gradient descent.

2.4.1.2 Retraining on frames which give a large loss

When using this sampling policy the training phase is separated into two phases. One random sampling phase and another hard training phase. During the random sampling phase frames are picked in the same way as in Section 2.4.1.1. The loss for this iteration is then compared to the losses of the samples used for the hard training phase. If the random training sample is shown to have a higher loss than any of the training samples used in the hard training this sample replaces sample in the hard training set with the lowest loss. During the hard training phase the net is trained one iteration for each of the samples stored in the hard training set.

2.4.2 Network optimization

For training the network a moment based stochastic gradient descent is used. The learning rate set to be strictly decreasing, but is lower bounded asymptotically by $1/n$ in order to allow the optimization to converge to a local minimum. The learning rate for a given iteration is $learning\_rate(t) = \frac{1}{(1+\gamma * t)^{0.5}}$

2.4.3 Computing weighted backward propagation

Since the different labels will have different weights, thereby giving different output pixels different training weights a weighted loss has to be used. The equation for the weighted loss is

$$J = \sum_{n,c,x,y} W(n,c,x,y) * l(t(n,c,x,y), \hat{t}(n,c,x,y)) \tag{2.2}$$

where n is the frame used c is the index for output channels, and x and y are spatial indexes. t is the output, and $\hat{t}$ is the ground truth. l is a function which return a small value if t and $\hat{t}$ match well and a large value if the two in arguments differ by much. When differentiating this with respect to $t(n,c,x,y)$ one get $\frac{dJ}{dt(n,c,x,y)} = W(n,c,x,y)\frac{dl(t(n,c,x,y),\hat{t}(n,c,x,y)}{dt(n,c,x,y)}$. In other words the weights are pointwise multiplied with the derivative of the unweighted loss in order to get the derivative for the weighted loss. This is illustrated in Figure 2.8

2.5 Obtaining predictions from a trained network

In order to extract bounding boxes from a test image the image is sent trough the network using forward propagation. Then local optima are found on the probability map if that area contains a pedestrian. For each of these local optima the value of each optimum is compared to a threshold. If the value of the optimum is

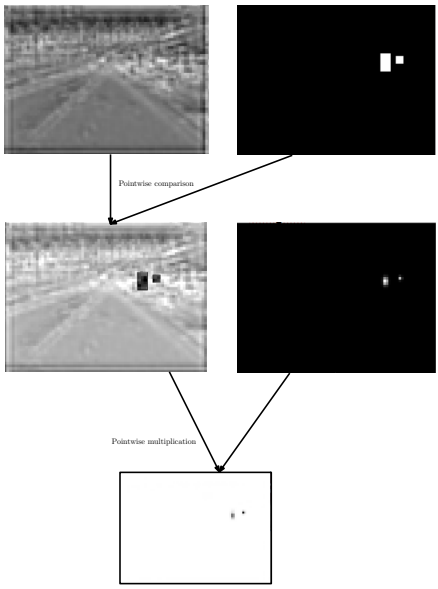


Figure 2.8: *Illustration showing how the weighted loss layer computes the gradients needed for backward propagation. The gradients are computed by first computing unweighted gradients by applying a loss function pointwise where the output of the network is compared to the ground truth. The unweighted gradients are then pointwise multiplied with the weights of the training samples in order to get the weighted gradients. The weighted gradients are then backpropagated through the network in order to update the parameters. In this figure all images are normalized so that the smallest value maps to black and the largest value maps to white.*

large enough the values of the regression offsets are read at this location in order to find the output bounding box for this location.

3
Results

This chapter presents the results of our experimental evaluation. There will also be a comparison between different approaches which have been tested in this thesis. How this evaluation measure is defined is described in Section 3.1.

3.1 Evaluation metrics used

As previously mentioned Caltech pedestrian dataset has a standardized way of determining the accuracy for a detector. This is done by letting each algorithm output bounding boxes with a corresponding score. In this thesis the score is set to be the value at the local optimum as described in Section 2.5. The accuracy is then evaluated where only bounding boxes with a score over a threshold are considered. Different values for this threshold is then used to compute the miss rate for different number of false positives allowed. The output curves are created by plotting curves where the average number of false positives per frames are compared to overall percentage of pedestrians detected for that test set for different thresholds. There are several different test sets in Caltech pedestrian dataset. The ones used for this thesis are shown in Table 3.1

The overlap for a bounding box candidate and a ground truth bounding box is computed as [17]

$$a_0 = \frac{area(BB_{dt} \cap BB_{gt})}{area(BB_{dt} \cup BB_{gt})} \tag{3.1}$$

If this value is larger than the overlap value in Table 3.1 that pair is set to be a match. Otherwise it is considered a false positive.

The percentage score which an algorithm is assigned in the caltech dataset correspond to the miss rate when the average number of false positives for each frame is set to be 10^{-1}.

Test set	Occlusion	Overlap	Minimum size	Maximum size
Reasonable	< 35%	> 50%	50	-
Overlap=x	< 35%	> x%	50	-
large	none	> 50%	100	-
near	none	> 50%	80	-
medium	none	> 50%	30	80
far	none	> 50%	20	30

Table 3.1: *This table shows some of the different test sets for the caltech pedestrian dataset. How much occlusion a pedestrian can have while still counting as a missed detection is described in column 'Occlusion'. The amount of overlap needed for a ground truth bounding box and a bounding box candidate in order for them to count as a match is described in the column 'Overlap'. The minimum height for pedestrian considered for matching is described in column minimum size. The maximum height is similarly described in the column 'Maximum size'*

3.2 Detection accuracy

The overall accuracy of the approach used in this thesis is relatively low. From Figure 3.1 it can be seen that the overall accuracy is higher than Viola and Jones [2], comparable to Sermanet et al. [6] and much worse than state of art. However this approach works well for the dataset Overlap = 25%. This indicates that poor localization of bounding boxes contributes a lot to the poor accuracy for the other test sets. Figure 3.2 shows that for this set this thesis gets competitive results. The network which gave the best accuracy used the architecture with multiple output levels.

3.3 Evaluating resampling policies

This section evaluates the two different sampling strategies discussed in Section 2.4.1. The architecture used for these evaluations is the one which has multiple output levels. The test set "reasonable" is used to compare the results for these tests.

3.3.1 Evaluating the policies on an untrained network

When using a network which is initialized with the googLeNet weights it is obvious that the random sampling gives a better result. The random sampling starts to fit the data well fast. When retraining on hard examples the network quickly become very bad at the task before it starts to improve again. A comparison of the losses over time for these two methods can be seen in Figure 3.3. Figure 3.4 shows that the random sampling training does not only give a lower loss during the training phase for this case, but the evaluation accuracy is better as well.

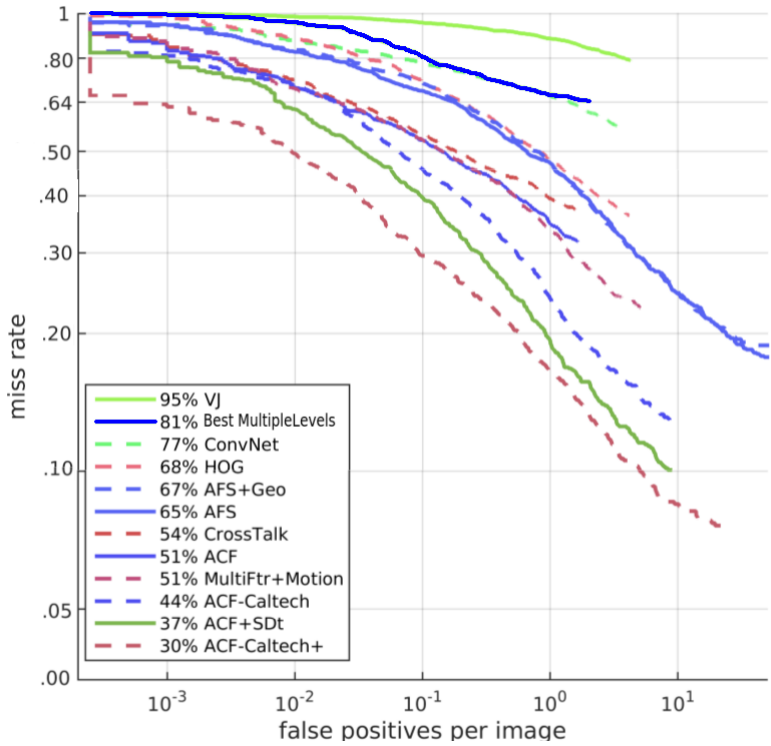


Figure 3.1: *The accuracy for the network which gave the best results (Best MultipleLevels) compared to other methods. The tests are done on the test set "reasonable"*

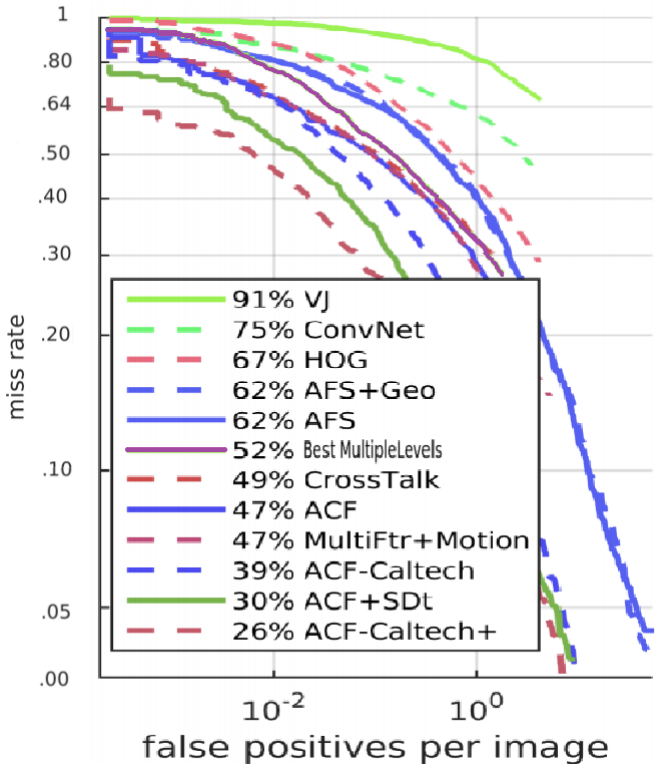


Figure 3.2: *The accuracy for the network which gave the best results (Best-MultipleLevels) compared to other methods. The tests are done on the test set "25% overlap"*

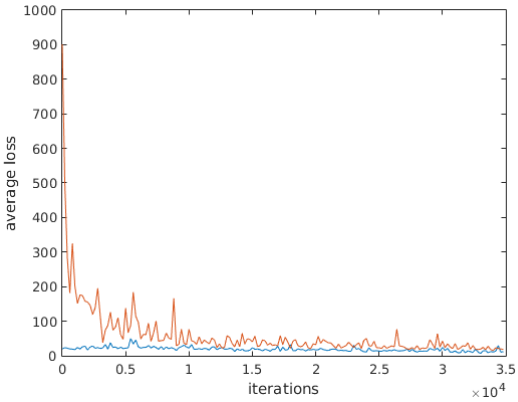


Figure 3.3: *Plot of the average of the latest 100 losses when training on random samples, compared to number of iterations trained. The red curve correspond to when retraining on examples with high loss, the blue curve correspond to the loss when using random sampling.*

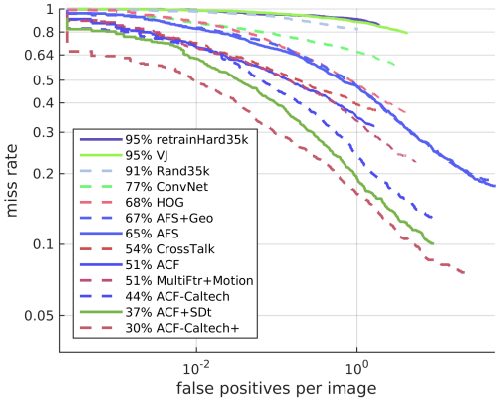


Figure 3.4: *The results on the test set reasonable when training a untrained network by doing random sampling (Rand35k) and when retraining on examples which give a large loss (retrainHard35k)*

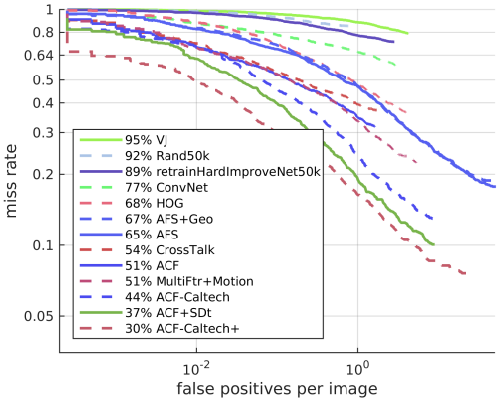


Figure 3.5: *The results on the test set reasonable when using the weights from 35 000 iterations of random sampling as a starting point for the optimization. When the training is done for an additional 15 000 iterations by doing random sampling (Rand50k) and when retraining on examples which give a large loss (retrainHardImproveNet50k). From this plot we see that both networks have similar performance, but retraining on hard smaples gives slightly better results.*

3.3.2 Evaluating the policies on an trained network

For this experiment the weights the net got after 35 000 iterations of random sampling will be used as an initialization for the network. The network is then trained by using the two different sampling policies for an additional 15 000 iterations. From this experiment we can see that the accuracy is better for the network which did retraining on samples which gave a large loss, see Figure 3.5.

3.4 Number of output levels evaluation

This section will compare the performance of using 5 output levels as described in Section 2.2 and using one output level. This will be done by comparing the results after training the network with the random sampling policy for 35 000 and 70 000 iterations.

From Figure 3.6 we can see that the multiple output level approach gives significantly better results early on. This seems to hold true even when letting the training continue for a longer time. When training the network with one output layer to convergence the accuracy is comparable to Viola and Jones. See Figure 3.7. When multiple outputs are used the accuracy is similar to Sermanet et al. See Figure 3.1.

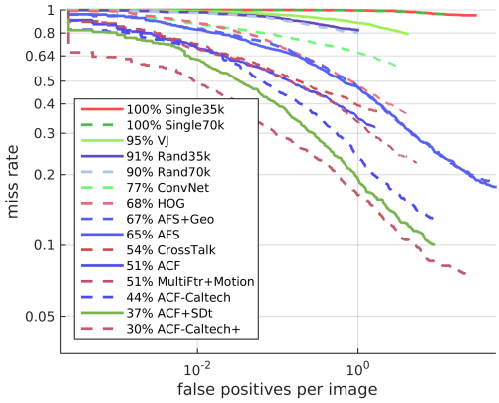


Figure 3.6: *The accuracy for doing random training after 35000 and 70000 iterations as well when using a single output level (Single35k and Single70k) and multiple output levels (Rand35k and Rand70k). From this we can see that using multiple output levels clearly outperforms only using a single output level.*

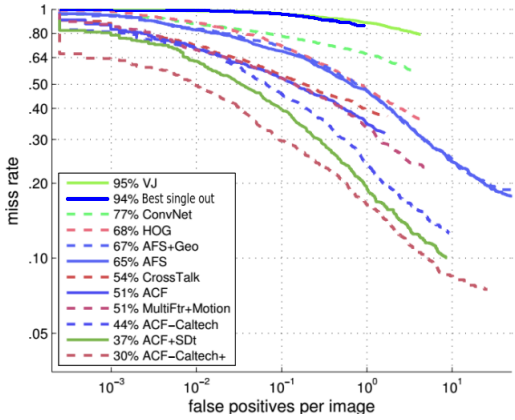


Figure 3.7: *The accuracy for the test set "reasonable" when the network using one output level has been trained to convergence (Best single out). Here we see that using one output level achieves similar detection results as Viola and Jones [3]*

4

Discussion

There is an intrinsic randomness in the sampling of the training examples and initialization of the weight. For this reason further experimental verifications are required to fully validate the merits of the methods described in this thesis.

4.1 Detection accuracy

The results, despite being promising are still inferior to state of art. Some configurations managed to get an accuracy similar to Sermanet et al. [6], which is a recent work and also uses CNN for pedestrian detection. For this reason it makes sense to compare the results of this thesis to that method. However, a lot of the poor results can be attributed to bad localization of bounding boxes in approximately the correct area, as we can see form the relatively good performance for the test set Overlap=25%. Therefore it should be possible to increase the results significantly by tuning the localization better. To increase the accuracy further it might be possible to apply methods used in other works such as Tian et al [7].

4.2 Sampling policy

The experimental results indicate that the two sampling policies tested have a similar performance when training on a pretrained network. The random sampling outperforms the policy which retrain on hard examples for a untrained network. The reason why retraining on hard examples early on gives poor results might be that the size of the gradient is proportional to the loss. Therefore retraining on hard examples for an untrained network causes the weight update policy to be too aggressive.

It should also be noted that this resampling policy alters the objective function

of the optimization. Random sampling tries to find the optimal weights W^* by approximating.

$$W^* = argmin_W \sum_{n=1}^{N} f(x_n; W) \tag{4.1}$$

Where x_n is the n:th training example. f is the loss given a training example and the net weights and N is the number of training examples. Instead the objective function for the retraining on hard examples policy becomes

$$W^* = argmin_W \sum_{n=1}^{N} f(x_n; W) + Cmax_j \sum_{n=1}^{H} f(x_{j_n}; W) \tag{4.2}$$

where $j \in \mathbb{N}^H$ such that $j_i \leq N \; \forall \; i$. H is the number of hard examples saved in the hard training set. and C is a constant corresponding to the ratio of hard training compared to random training.

One problem with using this resampling policy is that when retraining on hard examples images with pedestrian present might yield a high loss more often and therefore the inverse probability weighting, see Section 2.3.2, would become incorrect.

4.3 Network architecture

4.3.1 Output extraction layers

For the output extraction it should be possible to have the classfication output and the offset output share the first convolution layer. This should reduce the time needed for forward and backward propagation as well as decreasing the degrees of freedom for the offset regression. It should be possible to do this since the features needed for detecting a pedestrian and localizing them should be similar.

4.3.2 Using multiple outputs

The approach of using only one output for detecting multiple scales give much worse results then using multiple output levels for detecting pedestrians of different scales. There are other approaches too for detecting multiple scales, one such method is to downsample or upsample the input image by some factor and then detecting pedestrians for some given size in this image. This method should have a similar time complexity as using multiple output levels in the CNN. This method would also be similar to the common approach of taking region proposals and resizing them to a fixed size for classification.

4.4 Sliding window training

When training with the sliding window training there is a problem that it is not clear what class some regions are. One way to handle this problem could be to

set the weights at edges of bounding boxes to be 0 instead of a low value which is used for this thesis. Overall the tuning of the weights have not been the main focus of this thesis so there might possible to improve the results by investigating this further.

4.5 Time complexity

On a NVIDIA GTX 750 it takes 0.34 seconds to process one frame of size 640x480 using the architecture described in Section 2.2.3. For the architecture described in Section 2.2.2 it takes 0.12 seconds. This means that although this network is currently too slow for a real time application it might be possible to use a similar approach for this purpose.

Intuitively one might think that reason why the architecture with multiple outputs is slower is because of the many layers high up in the net. But the complexity for one convolution is proportional to the input size, as we will see in the following sections. Therefore a lot of computations will be used for detecting pedestrians for the lower levels. For example the if the output extraction furthest down is removed from the network the forward propagation for this modified network become 0.24 seconds.

One of the advantages of this approach is that it will take constant time for evaluating a single frame, which makes it suitable for real time applications.

The number of operations which are needed for forward propagation through an entire net is not hard to compute, but the total number of operations used for a given net is not very interesting since if the architecture is modified this number will change. Therefore this section will instead analyze the time complexity for single layers. The most computationally expensive layer, out of the common layers, is the convolution layer. For this reason only the complexity of this layer will be analyzed. The notation used will be
W: width
H: height
X: kernel width
Y: kernel height
N: channels in
N: channels out

4.5.1 Direct convolution

The complexity of doing a direct convolution is $\mathcal{O}(WHXY)$. For a convolution layer this operation is repeated for the number of for each input channel in order to produce each output channel. Therefore the complexity of a convolution layer with an image of size W×H, kernel size X×Y, N in channels and M out channels is

$$\mathcal{O}(WHXYNM) \tag{4.3}$$

When using direct convolution.

4.5.2 Convolution by using fast fourier transform

The complexity for doing a convolution on a one channel image using fast fourier transform (FFT) is $\mathcal{O}((W+X)(H+Y)log((W+X)(H+Y)))$ [21]. The complexity of an entire convolution layer using FFT will by the same reasoning as used in the direct convolution section be

$$\mathcal{O}(NM(W+X)(H+Y)log((W+X)(H+Y))) \tag{4.4}$$

One can see that the main advantage of using FFT for convolution instead of direct convolution is when the kernel size is large. For CNN this is generally not the case. Some optimizations can be done in this case since the FFT for the input image only has to be computed once. This will, however, not change the overall complexity since the complexity for computing the FFT for the N × M kernels has this complexity as well.

4.5.3 Convolution by using fast fourier transform on subimages

One problem with the method described above is that computing the FFT for the $M \times N$ kernels is a bottleneck for the complexity. By instead dividing the image into subimages of size $w \times h$ it is possible to further reduce the complexity. First $N\frac{WH}{hw}$ FFT of size $(w+X) \times (h+Y)$ has to be computed. The time complexity of this is $\mathcal{O}(\frac{NHW}{hw}(w+X)(h+Y)log((w+X)(h+Y)))$. Computing the FFT for the kernel will have a complexity of $\mathcal{O}(NM(w+X)(h+Y)log((w+X)(h+Y)))$. Doing the pointwise multiplication will have a complexity $\mathcal{O}(\frac{MNHW}{hw}(w+X)(h+Y))$. The inverse fast fourier transform (IFFT) to get the results will have the complexity $\mathcal{O}(\frac{MHW}{hw}(w+X)(h+Y)log((w+X)(h+Y)))$ therefore the total complexity is

$$\begin{aligned}\mathcal{O}(&\frac{(N+M)HW}{hw}(w+X)(h+Y)log((w+X)(h+Y))+\\ NM(w+X)(h+Y)log((w+X)(h+Y)) &+ \frac{MNHW}{hw}(w+X)(h+Y))\end{aligned} \tag{4.5}$$

It is possible to differentiate this expression with respect to w and h, setting the gradient to 0 and solve with respect to w and h. However, the resulting equations does not seem to have a simple analytical solution. Instead a heuristic for choosing w and h is used here to show that for a suitably chosen w and h this approach has a lower complexity, than the other methods described here.

By setting w $\propto$ X and h $\propto$ Y Equation 4.5 becomes

$$\mathcal{O}((N+M)HWlog(XY) + NMXYlog(XY) + MNHW) \tag{4.6}$$

All of the terms in this expression will be dominated by the complexities for the other cases. Therefore this approach has a lower complexity.

In this thesis direct convolution has been used for evaluating the runtime.

5

Future Work

5.1 Detection results

One simple way to improve the detection results would be to give the algorithm more time or data, but this would probably not be enough for state of the art results.

Another way to quickly improve the detection accuracy would be to find a better way to combine the output from the neural network into bounding boxes. The method used for this thesis was chosen for its simplicity and it is the only method tried. Bad localization caused a major decrease in accuracy. By finding a better way to use the output to find bounding boxes the accuracy could be increased by a lot.

Another thing which might improve the results is to try to apply methods which have achieved a higher accuracy and try to fit those methods to this framework.

One more thing which could be tried could be to add data from previous frames as additional input channels, this should make it possible to extract low level features similar to optical flow.

5.2 Resampling policy

The resampling policies described in Section 2.4.1 had a similar performance. However, it would be interesting to see how this policy performs compared to random sampling for other problems and in other conditions.

5.3 Multiple output scales

This thesis gives clear results that using only one output level when training and evaluating a network in the way described in this thesis gives inferior results compared to using multiple output levels. It might be of academic interest to use this method for other problems where scale invariance occurs such as the ILSVRC localization dataset.

5.4 Sliding window training

As mentioned in the discussion section the method used for training the net and handling edge cases might not have been optimal. The way these cases are handled and overall how the weights are chosen could be explored in future work.

5.5 Computing convolutions by using fast fourier transform on small patches

The runtime for the method described in have only been evaluated for matlab. The method gives the correct output, but in order to evaluate how effective this approach is for practical purposes a GPU implementation will be needed. Since the complexity is lower than the other approaches it would be of interest to see if an implementation using this gave a real speedup, or if the overhead would be to large.

Appendix

A

Basic CNN definitions

This chapter gives a brief background on the area of CNN as well as give the definitions of what different layers of a CNN do.

A CNN is built by several different layers. Some layers have parameters which are updated during the training phase in order to make the output of the top layer of the network as close as possible to the desired output.

The layers in the network are stacked on top of each other. In other words the output of one layer will be the input to the layers on top of it. The layers at the bottom are input layers which load data from some file structure, or generate it. The layers at the top of the network are usually loss layers. They compare their input with some desired input and produce a measure of how good they match.

A.1 Convolutional layer

A convolution layer takes a $M \times w \times h$ sized input X where M is the number of channels, w is the width and h is the height of the image. It then applies a $D \times M \times C \times R$ filter W to this image to produce a $D \times (w - C + paddingX) \times (w - R + paddingY)$ output Y. The terms $paddingX$ and $paddingY$ is the size of which the original image is zero padded on each size before the convolutions are applied. The mapping from the input and output is done through these computations.

$$Y(d,.,.) = \sum_{m=1}^{M} W(d,m,.,.) * X(m,.,.) \tag{A.1}$$

where * denotes a 2d convolution.

Writing out all all of the additions explicitly gives.

$$Y(d,x,y) = \sum_{m=1}^{M} \sum_{r=-\lfloor R/2 \rfloor}^{R-\lfloor R/2 \rfloor-1} \sum_{c=-\lfloor C/2 \rfloor}^{C-\lfloor C/2 \rfloor-1} W(d,m,c,r)X(m,x-c,y-r) \tag{A.2}$$

A.2 Max pooling layer

A max pooling layer maps the input to the output by setting the output for each pixel to be the maximum value for some neighbourhood defined by a kernel. The formula is

$$Y(c,x,y) = max_{\Delta X,\Delta Y \in D} X(c,x-\Delta X,y-\Delta Y) \tag{A.3}$$

Where D is some neighbourhood used for the kernel, X is the input and Y is the output. Usually D is a square. [22]

This can be seen as performing a grayscale dilation with a kernel D for all channels in the input to this layer.

A.3 Activation functions

Activation functions apply a non linear function for each input value to get the output.

A.3.1 Rectified linear unit (ReLU)

The ReLU layer is for example described in [22]. The output Y is computed from the input X as

$$Y(c,x,y) = max(0, X(c,x,y)) \tag{A.4}$$

sometimes leaky ReLUs are used in order to make sure that a channel does not get to a point where the output is always 0 because a too large learning rate is used. Then the output is computed as

$$Y(c,x,y) = \begin{cases} X(c,x,y) \text{if } X(c,x,y) & \geq 0 \\ C_{leak}X(c,x,y) & \text{else} \end{cases} \tag{A.5}$$

for some constant $0 < C_{leak} < 1$

When C_{leak} is equal to 0 the leaky ReLU and the normal ReLU is obviously the same.

A.3.2 Sigmoid activation function

The output Y is computed from the input X as

$$Y(c,x,y) = \frac{1}{1+exp(-X(c,x,y))} \tag{A.6}$$

A.4 Stride

In order to be able to increase size of the area considered when calculating the activation of a pixel for some hidden layer downsampling is used after some layers. The downsampling is done for some given step size. a common value for the step size is 2. The downsampling is commonly done after a pooling step. [22]

A.5 Local response normalization

This layer performs a normalization over nearby channels [22]. If applied to a color image it would be similar to normalizing by dividing each color channel by the brightness. The output Y is computed from the input X as

$$Y(c,x,y) = X(c,x,y) / \left(k + \alpha \sum_{j=max(0,c-n/2)}^{min(N-1,c+n/2)} X(j,x,y)^2 \right)^{\beta} \tag{A.7}$$

where α, β, k and n are hyper parameters of the model.

B

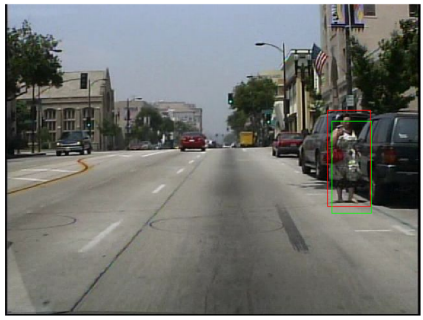
Images

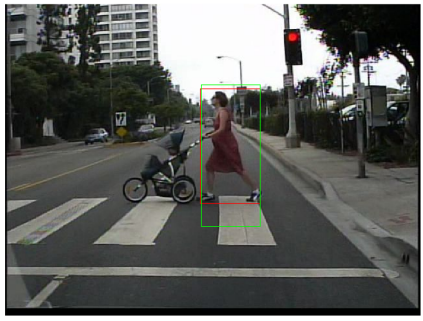
In this chapter green bounding boxes will be used for pedestrians extracted by the CNN. Red bounding boxes will be used for ground truth bounding boxes.

B.1 Images of detections

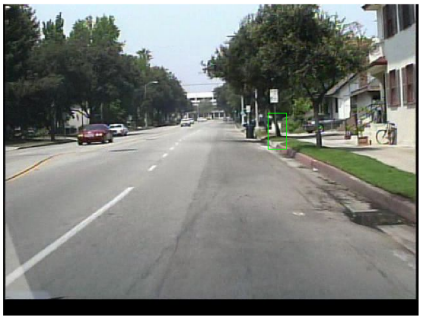
This section will show the outputs from a subset of the evaluation set. The threshold to decide which bounding boxes should be kept is manually selected so there are not too many false positives. The network architechture is the one with multiple output levels. The weights are the ones which gave the best results.

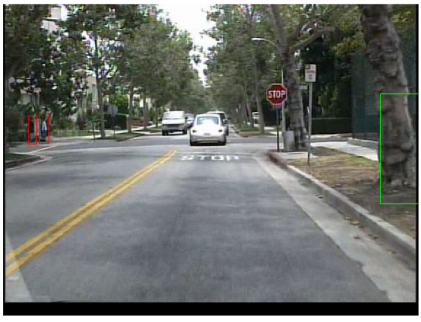
B.1.1 Good detections

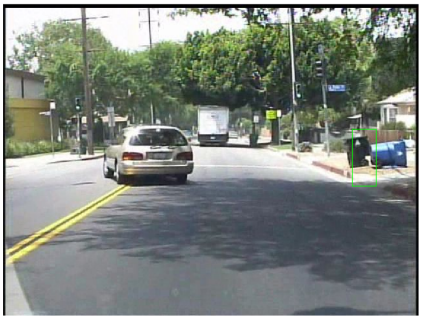




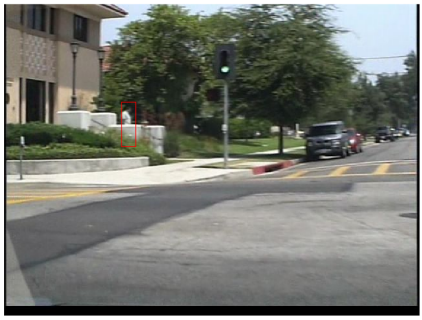
B.1.2 False positives

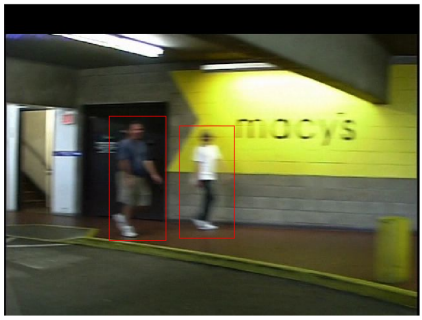


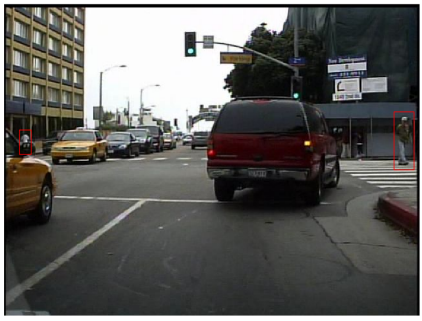


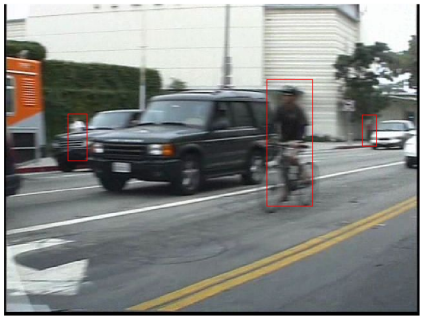


B.1.3 Missed detections



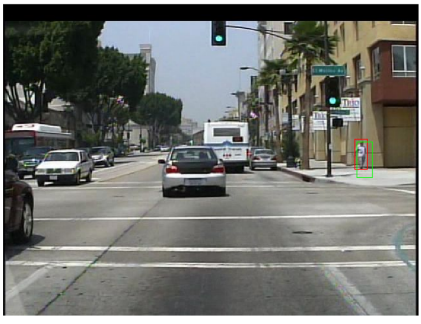
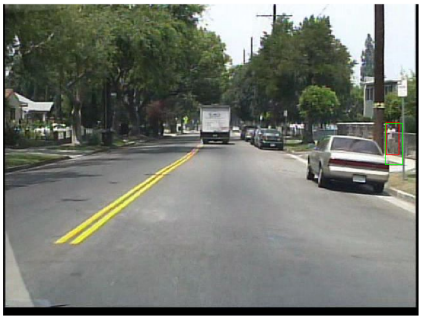
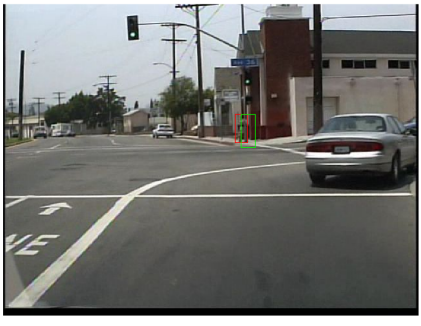
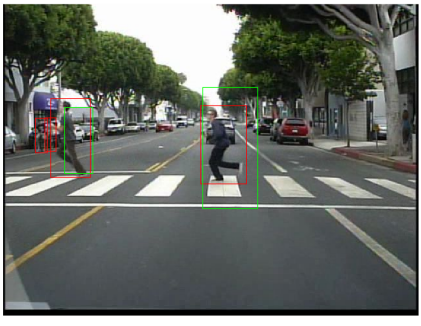




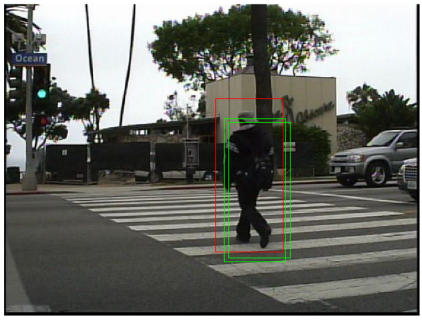


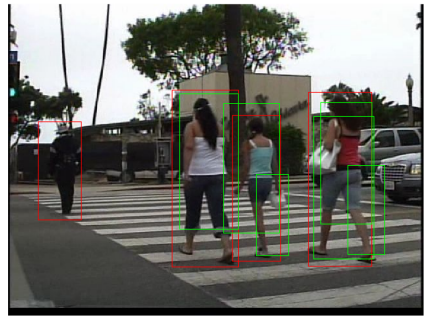
B.1.4 Bad localization for bounding boxes



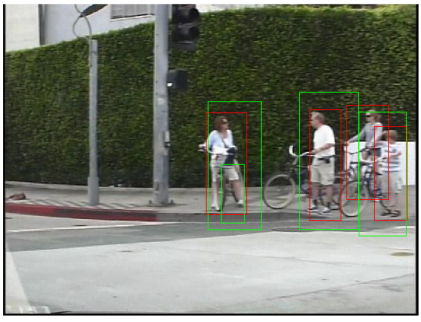





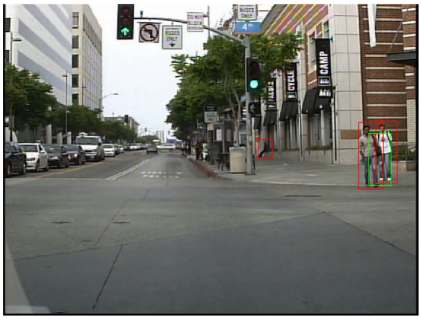
B.1.5 Multiple bounding boxes for one pedestrian



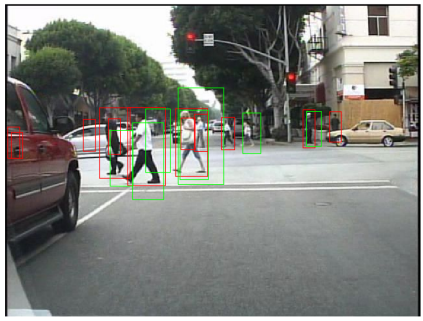
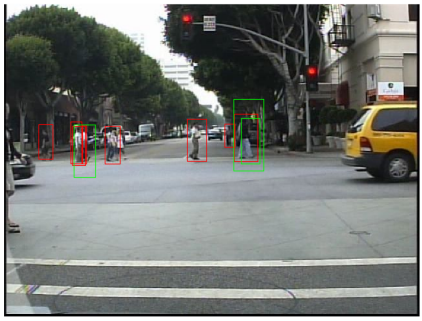
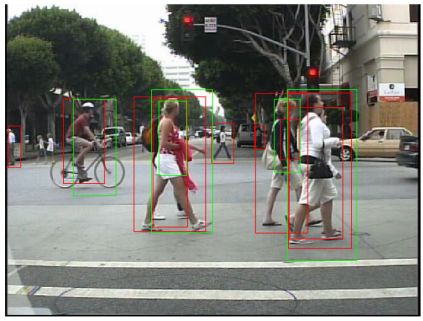


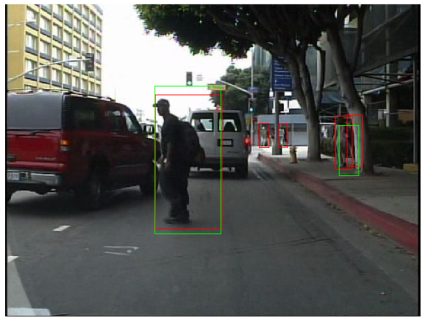
B.1.6 One bounding box for several adjecent pedestrians





B.1.7 Crowds



Bibliography

[1] Zhe Lin and Larry S Davis. A pose-invariant descriptor for human detection and segmentation. In *Computer Vision–ECCV 2008*, pages 423–436. Springer, 2008. Cited on page 1.

[2] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004. Cited on pages 1 and 22.

[3] Paul Viola, Michael J Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 734–741. IEEE, 2003. Cited on pages 1 and 27.

[4] Payam Sabzmeydani and Greg Mori. Detecting pedestrians by learning shapelet features. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007. Cited on page 1.

[5] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015. Cited on pages 2 and 8.

[6] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3626–3633. IEEE, 2013. Cited on pages 2, 7, 22, and 29.

[7] Yonglong Tian, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Pedestrian detection aided by deep learning semantic tasks. *arXiv preprint arXiv:1412.0069*, 2014. Cited on pages 2 and 29.

[8] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980. Cited on page 2.

[9] Yann LeCun, LD Jackel, Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, UA Muller, E Sackinger, Patrice Simard, et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261:276, 1995. Cited on page 2.

[10] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013. Cited on page 2.

[11] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, 2010. Cited on page 3.

[12] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE, 2014. Cited on page 3.

[13] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014. Cited on pages 3, 4, 7, 8, 11, and 12.

[14] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013. Cited on pages 3, 5, and 7.

[15] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1717–1724. IEEE, 2014. Cited on page 3.

[16] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1139–1147, 2013. Cited on page 4.

[17] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: A benchmark. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 304–311. IEEE, 2009. Cited on pages 4 and 21.

[18] A. Ess, B. Leibe, K. Schindler, , and L. van Gool. A mobile vision system for robust multi-person tracking. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08)*. IEEE Press, June 2008. Cited on page 4.

[19] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. Cited on page 4.

[20] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. *arXiv preprint arXiv:1409.5185*, 2014. Cited on page 4.

[21] Victor Podlozhnyuk. Fft-based 2d convolution. *NVIDIA white paper*, 2007. Cited on page 32.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. Cited on pages 40 and 41.