Person Detection in Thermal Images using Deep Learning

Erik Valldor
Abstract

Person Detection in Thermal Images using Deep Learning

Erik Valldor

Deep learning has achieved unprecedented results in many image analysis tasks. Long-wave infrared (thermal) images is still a little-explored area of application, and is the main subject of investigation in this thesis. To this end, a case study is performed where the goal is to detect persons in infrared images using deep learning. Two different deep learning based approaches are implemented and benchmarked against a baseline cascaded classifier. Due to the large amount of unlabelled data available, an autoencoder setup is used to pretrain the deep learning based detectors. One of the detectors greatly outperforms the baseline, while the other (an experimental approach) lagged slightly behind the baseline. The main difficulty concerning the ability of the detectors to generalize was determined to be the wide dynamic range of infrared images, together with the many different contrast situations that can occur due to weather and ambient temperature.
## Contents

1 Introduction ................................................. 2  
  1.1 Task description ........................................ 2  
  1.2 Contributions ......................................... 3  

2 Neural networks ........................................... 5  
  2.1 The artificial neuron .................................. 5  
  2.2 Layers .................................................. 6  
  2.3 Feed forward neural networks ......................... 6  
  2.4 Learning ............................................... 6  
  2.5 Backpropagation ....................................... 7  
  2.6 Convolutional neural networks ....................... 7  
  2.7 Deep neural networks ................................ 10  

3 Review of existing methods ............................... 12  
  3.1 Brief overview ......................................... 12  
  3.2 Analysis of relevant methods .......................... 14  
      3.2.1 Type A detector .................................. 15  
      3.2.2 Type B detector .................................. 16  
      3.2.3 Type C detector .................................. 16  

4 Data set ................................................... 17  
  4.1 Infrared quirks ....................................... 18  
  4.2 Annotations ........................................... 19  
  4.3 Refinement ............................................ 19  
  4.4 Partitioning .......................................... 20  

5 Pretraining ................................................. 22  
  5.1 Autoencoder ........................................... 23  
      5.1.1 Convolutional autoencoder ...................... 24  

6 Detector implementation .................................. 26  
  6.1 Base network .......................................... 26  
  6.2 Sliding window detector ............................... 26  
  6.3 Deconvolutional detector ............................. 31  

7 Evaluation ................................................. 34  
  7.1 Experimental setup ................................... 34  
  7.2 Training ............................................... 35  
  7.3 Results ............................................... 37  
  7.4 Analysis .............................................. 38  

8 Conclusions ............................................... 42  
  8.1 Future work ........................................... 43  

A Technical recipe ........................................ 44
1 Introduction

Given an image and a set of object classes, the goal of object detection is to determine whether the image contains any objects of the specified classes, as well to indicate where in the image these objects are located. This is in contrast to image classification, that only concerns the presence or non-presence of these objects, and to semantic segmentation, that seeks to classify individual pixels as being or not being part of an object of the specified classes. Image classification is hence a subtask of object detection, which in turn is a subtask of semantic segmentation.

Consider a surveillance scenario where cameras are stationed around an object of interest. Traditionally this requires an operator to monitor the feed from all these cameras for suspicious activity. Not only does this require absolute attention from the operator, making it prone to error, but as the number of cameras increases beyond a certain point, manual analysis of all camera feeds becomes an intractable task for a single person. A system capable of automatically analyzing all these video feeds, and alerting the operator when suspicious objects are visible would be of huge benefit in a scenario like this. One possible way of solving this problem is to apply an object detection system to each frame of the video feeds supplied by the surveillance cameras. Unlike image classification, object detection is not only concerned with the presence of an object within the image, but also its location and size. This makes it possible to infer information about for example the objects movement and location.

The use of deep learning for these types of image related analysis tasks has had huge success in recent years [26]. Unlike classical neural networks that typically contain one or two hidden layers, deep learning employs neural networks with up to as many as a thousand hidden layers [19]. It has been shown that these deep networks have the ability to learn discriminative features from raw input data, and thereby replace the traditional labor-intensive task of designing hand-engineered feature extraction methods [26, 27]. This makes deep learning extremely attractive, as it moves one step closer to providing end-to-end learning agents requiring minimal human intervention. Their main drawback lies in the large amount of data needed for training due to the many parameters present in the model.

1.1 Task description

The focus of study in the present work is how deep learning can be applied to the task of detecting humans in infrared images. Deep learning has had an unprecedented success on the task of object detection. However, in the time of writing there is very little research on how it can be applied to infrared images. There is some work where deep learning is explored for infrared images [17], but it does not consider any of the current commonly used methods for object detection, and will not be relied upon in this work. The detection of humans is chosen as a case study mainly because of the availability of annotated data, but also because of its interesting applications.
Related work on object detection (Section 3) typically employs a pretraining scheme, where a large data set of annotated images is used. In this work, none of these data sets are directly applicable because of the difference between infrared images and images in the visible spectrum. Some other means of pretraining will have to be conceived due to the relatively low number of annotated examples.

The task of detecting humans in the provided data set is most closely related to the task of pedestrian detection. What differs in this data set compared to common pedestrian detection data sets, besides the fact that it consists of infrared images, is that the humans in this data set are very small compared to the overall size of the image. In this work, a human can be as small as 10px in height. Small objects are not well handled by the common deep learning detection methods today [22].

This thesis consists of:

- the analysis, composition and refinement of available data into a data set suitable for machine learning applications (Section 4). This includes converting data from raw formats as output by the infrared cameras into a uniform format that makes the data easy to work with, manually reviewing image annotations to assure their quality, as well as partitioning the data into training, validation and test sets;

- an investigation of the current methods used to perform object detection using deep learning (Section 3);

- implementation of two different deep learning models for the task of detecting humans in infrared images (Section 6);

- training and benchmarking the implemented methods together with a classical (non-deep learning) approach previously used for this task (Section 7). This also includes the implementation and use of a convolutional autoencoder for the purpose of pretraining the models (Section 5).

The work is performed at the Swedish Defence Research Agency that also provides the data.

1.2 Contributions

The contributions of this thesis can be summarized as follows:

- Object detection using deep learning is applied to infrared images. An application that is sparsely appearing in current literature.

- Object detection is performed on objects as small as 10px in height. This is considered to be very small objects when comparing to common data sets used in mainstream research such as MS-COCO [30].

- A deep convolutional autoencoder is used to pretrain the feature extraction part of the object detection model. This allows the feature extractor to be trained in an unsupervised manner, which is needed when annotated data is scarce.
• A detection network based on sliding windows at multiple positions in the convolutional feature hierarchy is implemented. It is shown to outperform the baseline by a large margin.

• A novel approach to object detection is explored by applying deep learning techniques used for semantic segmentation to the task of object detection. It alleviates many of the design choices needed for the sliding window detector, and requires fewer hyperparameters.

• The wide dynamic range of infrared images is shown to cause difficulties for the detectors to generalize.
2 Neural networks

Deep learning essentially refers to the use of neural networks with many hidden layers. In theory, a single hidden layer is sufficient to represent any function to any degree of accuracy [21], but may however require an infinite number of neurons. Using multiple hidden layers can be motivated by comparing neural networks to logic circuits. The number of units needed to represent some functions decreases as the depth of the circuit increases [2]. More complex functions can therefore be represented by fewer neurons when the depth of the network is increased.

2.1 The artificial neuron

Figure 1: Artificial neuron.

The basic computational building block of a neural network is the artificial neuron, depicted in Figure 1. It takes a fixed number of scalar input values, represented as the elements of a vector \( \vec{x} \) and outputs a single scalar value \( y \). Each input position of the neuron is associated with a scalar parameter called a weight, represented here as the elements of the vector \( \vec{w} \). It is these weights that constitute the parameters of neuron, and are what is adjusted when the neuron is learning.

The computation performed by the neuron consists of two steps. The first step computes a linear combination of the input values and their corresponding weights, which is equivalent to the dot product of these two vectors:

\[
s = \vec{x} \cdot \vec{w}. \tag{1}
\]

The second step is performed by feeding the result of the first step through an activation function \( \sigma \). The output from the second step is then declared the final output of the neuron:

\[
y = \sigma(s). \tag{2}
\]
Commonly used activation functions include the sigmoid shaped logistic \((y = \frac{1}{1+e^{-x}})\) and hyperbolic tangent \((y = \frac{1-e^{-x}}{1+e^{-x}})\) functions. For deep neural networks, the ReLU \((y = max(x, 0))\) activation function is a popular choice. The activation function introduces a non-linearity to the output of the neuron, making it capable of non-linear approximations.

2.2 Layers

When more than a single output dimension is required, such as for multi-class classification, several neurons can be used in conjunction. The output of each neuron then corresponds to an individual dimension in the output space. This can be represented by a layer consisting of as many neurons as the desired output dimension. Each neuron receives an input vector, and preforms independent computations to produce their output. For a fully connected layer, i.e when each neuron in the layer receives input from all neurons of the previous layer, the computation performed can be described by a vector-matrix multiplication followed by an element-wise application of the activation function:

\[
\bar{y} = \sigma(W\bar{x})
\]

where \(W\) is the matrix of weights in which each row represents the weights of a specific neuron in the layer.

2.3 Feed forward neural networks

To increase the representational abilities of the network, several layers can be used, and are then connected together in a series. The input signal is then propagated layer by layer in such a way that the output from the first layers is fed as input to the second layer, whose output in turn is fed as input to the third layer etc. This is the general structure of what is referred to as a feed forward neural network. This is probably the most common type of neural network today, and is what is often meant when talking about a "neural network".

When conceptualizing such a network it is common to introduce a non-computational layer as the first layer. This is referred to as the input layer, and its only function is to distribute the input signal to the neurons of the second layer. The last layer is referred to as the output layer, and all layers in between are referred to as hidden layers.

2.4 Learning

The artificial neuron is trained to produce a specific output given a specific input. This is done by iteratively adjusting the weights of the neuron using a set of training examples.

Consider a set \(T = \{t : t = (\bar{x}, d)\}\) of training examples where \(\bar{x}\) is the input vector and \(d\) is the desired output of the neuron when given \(\bar{x}\) as input. When feeding \(\bar{x}\) to the neuron, an output value \(y\) is obtained. The output error \(E\) can
then be computed by using any function that can quantify the error between \( y \) and \( d \), such as the squared difference (i.e. \( (y - d)^2 \)). The goal when training the neuron is to minimize this error for the examples in the training set by adjusting the weights \( \bar{w} \) of the neuron:

\[
\min_{\bar{w}} \sum_{i \in T} E(y, d).
\]

(4)

This is a minimization problem that can be solved using a nonlinear optimization strategy such as gradient descent.

When performing classification, the artificial neuron can be considered to define a hyperplane in the input space that during training is aligned to separate the classes of interest. If the activation function is the logistic function, this is equivalent to a logistic regression.

### 2.5 Backpropagation

The most popular way of training a feed forward neural network is by use of the updating scheme referred to as backpropagation [28, 38]. It consists of two phases called the feed forward phase, and the backpropagation phase.

In the feed forward phase, a training example is input and propagated through the network. The error of the output of the network is then quantified using an error function that is appropriate for the application, such as the mean squared error, cross entropy, or similar.

In the backpropagation phase we seek to adjust the weights of the network in such a way that the output error is decreased. This is done by calculating the gradient of the error with respect to the weights, and updating the weights in the negative direction of the gradient. However, it is not possible to directly calculate this gradient with respect to weights other than in the last layer, since there is a dependence between weights in different layers. This is why the updating of the weights begins by calculating the gradient of the error with respect to the last layer, after which the gradient of the weights in the layer before can be calculated using the chain rule. This is done all the way up to the first layer, and can be though of as propagating the error signal backwards through the network from the last to the first layer, hence the name "backpropagation (of errors)".

### 2.6 Convolutional neural networks

Convolutional neural networks are by far the most popular type of neural network when it comes to image processing [39]. The principles are the same as for normal feed forward neural networks but the convolutional network employs two different types of layers with some special characteristics that make them well suited for the task of image analysis. These are the convolutional and pooling layers described below.
The convolutional layer. A layer in a typical feed forward network is what is referred to as fully connected because every neuron in one such layer receives the output from all neurons in the previous layer. This means that each neuron in this layer performs their computation on the whole input vector. In a convolutional layer on the other hand, each neuron only receives a small local portion of the input to perform their computation on. Figure 2 illustrates these two types of connection topologies. The spatial window visible to a particular neuron is called its receptive field, and the convolutional layer greatly reduces the receptive field of the neurons compared to a fully connected layer. In a convolutional layer, the receptive fields of all neurons tile the input vector in (usually) overlapping windows. The reason for imposing this type of restriction comes from the realization that, in a signal with spatial correlation such as an image, locations in close proximity to one another are probably more correlated than locations spaced further apart.

In the case of an image where an object’s location within the image is arbitrary, the above implementation of the convolutional layer means that each neuron in the layer would have to be trained to detect the same thing. This allows for an optimization that greatly reduces the number of parameters of the convolutional layer. Each neuron within the same layer can share their weights with each other. This means that the computation performed by a convolutional layer can be efficiently implemented as a convolution operation, where a filter with values representing the shared weights of the neurons in the layer are convolved with the input image.

The convolutional layer addresses a very important issue in image analysis,
namely translation invariance. The fact that the filter is applied to all possible locations in the image means that it is able to detect patterns in the image regardless their location. In practice this means that a convolutional layer performs a convolution of the input with filters that are learnt during training. A single convolutional layer typically contains many filters, each of which produce their own output. This is in order to allow the layer to be able to learn to extract several different types of features from the input. The output from a convolutional layer applied to an image is referred to as a feature map. It is essentially a multichannel image, where each channel corresponds to the output from individual filters.

The pooling layer. In addition to extracting meaningful features from the input as done by the convolutional layer, it is often also desirable to reduce its dimensionality. To this end, convolutional networks also contain a subsampling layer called a pooling layer. Its purpose is to reduce the spatial size of the signal, and is effectively a downsampling of the input feature map.

![Max pooling](Figure 3: Max pooling)

Different types of pooling layers have been explored but the one most often used today in the field of image analysis is the max pooling layer. This type of pooling layer has been shown to have a positive influence on the features learnt when used in conjunction with convolutional layers [33]. The layer, as the name suggests, subsamples the input image by selecting the maximum value within a local window that is convolved over the image. This is in contrast to e.g. bilinear subsampling, that would compute a linear interpolation of the values in this local window to produce the output value. The max pooling operation is illustrated in Figure 3, where a window size of 2x2, and a stride (i.e. the distance between the locations where the filter is applied during the convolution) of the same size is used. This is a very common configuration of the max pooling layer,
and results in a subsampled image with half the width and height of the original image.

Besides reducing the dimensionality, this subsampling also has the effect of making the network moderately invariant to object scales, as convolutional layers will have the opportunity to extract features from successively subsampled images, in which large patterns become gradually smaller, and eventually small enough to be detected by a filter. However, if an object is too small to begin with, the subsampling might destroy the little information that is available before it has reached a filter with a sufficiently high abstraction level, effectively making the object invisible to the network.

### 2.7 Deep neural networks

One of the mayor difficulties that has long plagued the task of performing image analysis is the need to create functions that extract discerning features from the image that can reliably be used by a classifier. This is a very time consuming task, and often what makes or breaks a good system. Neural networks with many hidden layers have shown to remedy this problem by being able to automatically extract such features, given enough training data.

Such deep networks have traditionally been considered intractable to train because when the gradient is propagated through many layers it gets more and more "diluted" and eventually vanishes due to the many parameters of the model. Recent advances in hardware as well as architectural inventions have allowed for training these deep networks.

One technique that has shown to remedy this problem is the \textit{rectified linear unit} (ReLU) activation function that is defined as $\text{ReLU}(x) = \max(0, x)$. Unlike the sigmoid shaped activation functions, ReLU allows the gradient to be propagated without being diminished due to the activation function [14].

Another invention that has helped training deep networks is \textit{batch normalization} [23]. It is a method by which the signals that propagate through the network are normalized. This is done in two stages:

- During training, the whole minibatch of examples are normalized so that the signal has zero mean and unit variance. The parameters used to do this normalization are computed during training, and saved using a running mean.

- During evaluation, the saved parameters are used to perform the same normalization.

Batch normalization allows deeper networks to be trained, and the convergence of the training becomes faster.

Deep neural networks have shown some very interesting properties in the way that information is extracted from the input. In a trained network, the first layers learn to detect very general features that are assembled into higher level feature detectors by the succeeding layers. This is particularly visible in convolutional networks applied to image data, since this information is easy
for a human to interpret visually. Here, the first layer typically concerns itself with detecting edges and corners, succeeding layers then combine these detected edges into higher level structures. Layers further from the input combine features from preceding layers to form higher and higher levels of abstraction and specialization [47].
3 Review of existing methods

The task of object detection can be defined as follows: Given an image and a set of object classes, localize and classify all instances of these object classes present in the image. Note that the image does not necessarily contain any such objects. The location of an object is typically represented by means of a bounding box. This is a rectangular region defined by the pixel coordinates of its four corners, that encapsulates the object of interest. The bounding box is desired to be minimal in the sense that it is as small as possible while still containing the whole object of interest.

Since the success of deep learning for image classification [39], a natural extension is to apply it to object detection that, besides classification, also concerns the localization of these objects. When performing image classification, two sought after properties of the classifier are translation and scale invariance. It should be possible to detect an object regardless of its size and position within the image. Detection on the other hand, has to be both translation and scale variant, since the result of the detection also has to include a bounding box that specifies the location and size of the object.

Most of the current work on object detection uses one of the handful of widespread image data sets that are available today. The existence of an open data set more or less regulates the amount of literature available on the specific subject. Most of the available data sets are very general, and contain many different object classes without being directed toward any specific practical application. These data sets include ImageNet [5], PASCAL VOC [10] and MS COCO [30]. This work however is only concerned with a single object class, i.e. “human” and more resembles the specialized task of pedestrian detection. There are a couple of such data sets available, and the most popular ones include Caltech Pedestrian [6] and KITTI [11]. The methods used on these more specialized data sets however, still use the same basic principles as the object detection methods used on the more general data sets.

3.1 Brief overview

All reviewed work is based on taking a convolutional neural network as used for image classification, and subsequently modify it in some manner in order to also allow for the extraction of the spatial information for each object. The main concern of object detectors using deep neural networks is hence how the spatial information about the object is to be acquired.

One of the first approaches to solving this problem using deep learning was to first extract candidate regions from the image, and subsequently classify these regions as if they were individual images. This way, the object location is given by the region from which it originates. This method was first proposed by Girshick et al. [13] who named it Regions with CNN (R-CNN). In this initial proposal, class independent region proposals are extracted in a pre-processing step using “classical” image analysis methods. Each region is then fed to a CNN for feature extraction, whose output in turn is input to a series of binary SVM
classifiers, each trained on a single object class. In order to refine the crude object location given by the region proposal, a regression is performed on the bounding box coordinates to produce the final detection area. This regression predicts the coordinates given the feature vector output by the CNN.

Building on this initial idea, several other models were proposed, trying to account for some shortcomings of the aforementioned method, the dominant ones of which are a cumbersome training process involving several different stages, and slow processing. SPP [18] improved performance of R-CNN by running the whole image through the convolutional network, and sharing the produced feature map for all region proposals, instead of running each region proposal separately though the convolutional network one at a time. This provided an immense speedup to the network. Fast R-CNN [12] introduced a similar improvement as the latter, but also simplified the architecture by streamlining the training process. The resulting network is trainable end-to-end, assuming that the region proposals are done externally.

Another approach introduced by Erhan et al. [9] performs the detection in reverse order to the above method. The detection is done by training a CNN to perform regression of the coordinates of a set of class agnostic object bounding boxes. The contents of these bounding boxes are then classified by a separate classification network. This approach has the advantage of requiring less processing than the above methods, since the region proposal preprocessing step can generate a lot of false positives. On the other hand, since bounding box regression is performed first, the number of bounding boxes the network should produce has to be hard coded into the model, which makes it less flexible.

Similar in spirit, Szegedy et al. [42] trains a network to produce a binary mask that indicates positions within the image that contain objects of interest. By running the input through the network in multiple scales, the results can be aggregated to produce the final detection mask.

YOLO [35, 36], predicts the presence of an object, as well as a bounding box, for a fixed size grid that tiles the input image, similar to that of [42] except that only a single pass through the network is sufficient for detection. To handle small objects, the input image is up-scaled before fed to the network.

One early example of using a sliding window on the last feature map produced by a convolutional network is Overfeat [40]. To accommodate for detection of different scales the input is scaled to multiple sizes before it is fed to the CNN.

The latest incarnation of R-CNN, Faster R-CNN [37], replaces the “manual” region proposal method by a Region Proposal Network (RPN), that shares convolutional layers with the feature extraction network. The RPN is trained to produce region proposals that are likely to contain objects of interest. In contrast to the above methods, region proposals can now be trained, instead of relying on the fixed method used in earlier versions. This greatly increases training and inference speed as the old region proposal method was a heavy bottleneck.

The RPN is basically a sliding window applied to the last feature map output by the feature extraction network. This window is in turn connected to a binary
classifier that is trained to classify positions as potential regions or not. These regions are then used in the same way as in Fast R-CNN. The sliding window has associated with it a default bounding box called an anchor box. The final bounding box coordinates are predicted as an offset from this anchor box. The RPN contains many sliding window classifiers, each trained to detect objects of different sizes and aspect ratios.

R-FCN [29] propose an improvement to how the features from the base CNN are pooled in Faster R-CNN in order to improve detection performance.

SSD [31] also uses sliding windows in a similar fashion to RPN, but applies them to several different levels in the feature hierarchy produced by the feature extraction network. Due to the successive down-sampling done by the CNN, this allows detection of objects at different scales similar to that of an image pyramid.

MS-CNN [4] is another example of using sliding windows in multiple positions in the feature hierarchy. Unlike SSD, which implement the classifier and bounding box regressor as a single convolution operation, this approach only uses a class agnostic classifier for the sliding window, and the positive results from this operation are then fed to a separate network that performs the classification and regression. It can be seen as an implementation of Faster R-CNN but with multiple sliding windows in different levels in the convolutional feature hierarchy.

In the related area of semantic segmentation, an approach that has shown to yield promising results is to use a deconvolutional network to produce a segmentation map of the full image [1, 32, 34]. The input image is first fed to a CNN, for feature extraction, the output of which is subsequently fed to a deconvolutional network. The deconvolutional network consists of convolution and upsampling operations, and outputs a segmentation map of the same size as the original image, where each pixel is classified as belonging to a specific class. The main difference between these approaches is how the upsampling in the deconvolutional network is performed. The reason this method is included in this review of detection methods is because of its simplicity, and if it could be applied to the detection problem, it would make a very attractive alternative.

## 3.2 Analysis of relevant methods

Common for all reviewed methods is the use of a convolutional neural network for feature extraction. This network is referred to as the base network and can be viewed as the workhorse of the detector. It is responsible for converting the raw image into meaningful features that can be subsequently used by a classifier to perform detection. The exact type of architecture used for this purpose varies between methods and tasks, but is of less importance as the general structure with convolutional and pooling layers is the same. It should be mentioned that the data sets used in the reviewed papers generally contains significantly larger objects than the ones of interest here. For example, in the ImageNet dataset, the object(s) that are to be detected are centered within the image, and make up the majority of the whole image. The MS-COCO dataset is more “realistic”
in the sense that the objects are generally smaller than in ImageNet. Here, any object with a pixel area less than $32^2$ is considered a small object, but only about 41% of the objects in the dataset are within this range.

3.2.1 Type A detector

The first group consists of those methods that by some means, connects a classifier to the final output of this base network. Models using this approach include Faster R-CNN [37] that connects a sliding window to the output of the base network and YOLO [35, 36] that connects a fully connected classifier to the output of the base network. One obvious drawback of these methods is the limited scale at which detection can be confidently performed. This stems from the successive subsampling performed by convolutional neural network. For example, if the base network contains five subsampling layers of ratio (2x2):1, a single pixel in the final feature map produced by the network would correspond to a potential area of origin equal to 32x32 pixels. It is therefore unclear if any object smaller than this would be representable this way. To remedy this, most work up-scales the image to enlarge small objects before performing detection. This not only increases processing time, but also has the potential to introduce...
artifacts not present in the original image. Nevertheless this has been shown to work for detecting objects of smaller scale.

3.2.2 Type B detector

The second group of detectors connect classifiers to multiple levels in the feature hierarchy created by the base network and includes methods such as SSD [31] and MS-CNN [4]. Both use sliding window classifiers on these feature maps to perform detection. This way of utilizing feature maps of different scales partially solves the small objects issue mentioned above.

One of the most important decisions for this model is at what levels to perform the detection and what sizes to use for the sliding windows. When performing the classification, it is desired to do this as close to the output layer as possible to allow for as many convolutional layers as possible to perform feature extraction. This is a problem when wanting to perform detection of small objects, because of the successive subsampling performed by the base network. Small objects will be subsampled to sub-pixel size and in a sense "disappear", as described above. This leads to a dilemma because the detection has to be performed at a feature level where desired object sizes are still visible, meaning that there is less opportunity for feature extraction. Related work solves this by upscaling the input image before feeding it to the network.

It should be noted that this type of network requires careful tuning of the configuration for the different classifiers so that all objects of interest are visible to at least one of them. It is therefore not a very general approach, since the size of the objects one wants to detect often changes with the application. In the time of writing, this approach accounts for state-of-the-art results on popular benchmarking data sets [10, 39].

3.2.3 Type C detector

The third type of detector has only been used for semantic segmentation tasks as of today. Here, a deconvolutional network is connected to the output of the base network. The deconvolutional network is trained to output a segmentation map of the whole image. It is interesting because it does not concern itself with any explicit detection of different feature scales. This task is left up to the network itself, which makes it a very general approach. Although the present work is concerned with detection and not segmentation, it might be possible to perform some approximate segmentation using this approach and the available data. Intuitively, it may be possible to segment the whole rectangular region given by the object bounding box since, averaged over many examples, the invariant object of each segmentation annotation would be the human. Even though it does not seem likely to produce very precise segmentation maps this way, it may be sufficient for the purpose of detection.
4 Data set

The data set consists of video sequences filmed with long-wave infrared (8–15µm) cameras. The videos are filmed exclusively in outdoor settings, and include both city and country environments with a mix of stationary and moving cameras. Some sequences contain scenes created by actors, and others of “ordinary” people. Most sequences originate from cameras filming in a resolution of 320x240, but a small subset were filmed in a resolution of 640x480. Figure 5 shows some example frames from the data set.

The individual frames consist of a single channel, where each pixel is represented by a 14 bit integer value. In total there are about 1 000 different sequences of varying length, amounting to a total frame count of roughly 1.4 million. However, only about 70 out of the 1 000 sequences contain human annotations, and in some sequences, only a subset of the present humans have been annotated.

One prominent characteristic that spans through the whole data set, and that distinguishes it from typical pedestrian data sets, is that the people in the images are overall very small compared to the size of the image (see Figure 7). One possible reason for this is the different applications these data sets were created for. In pedestrian data sets, the goal is typically to detect humans in close proximity to a vehicle, whereas this data set is created for the purpose of long range surveillance.

Figure 5: Example frames from the data set. Since displays normally do not support the channel width of the original image, each image has been normalized independently for clarity.
4.1 Infrared quirks

As stated above, the data consists of infrared images filmed in the long-wave infrared spectrum. This type of images are also called thermal images, as the main source of electromagnetic signals in this spectrum is the heat radiated out from objects. As can be seen by the examples in Figure 5 it is possible to represent these images using a grayscale coloring, where brighter pixel values correspond to warmer areas, and darker pixels correspond to colder areas. It is clearly the case that these images can be interpreted by a human without much effort, i.e. we can easily distinguish the humans present in these images, as well as what type of environment they were taken in.

When looking at the numeric data that represent these images however, one characteristic that becomes very clear is the extremely wide dynamic range of pixel values that can be used (14 bit). A single image typically only occupies a small portion of this possible range of pixel values, and this is mainly determined by what weather conditions the images were taken in. When comparing two images that originate from different sequences, they typically occupy completely different portions of this dynamic range.

![Figure 6: Visualization of dynamic range.](image)

The images in Figure 5 have been individually normalized to allow for easy interpretation. If they instead had been normalized jointly, they would become extremely bright or dark in relation to each other, and make interpretation next to impossible. Figure 6 illustrates this by showing two images that have been normalized jointly, together with their unnormalized intensity value histograms.
If this is not dealt with, it could be the case that the same effect is seen by the neural network, which would make training much harder. In this work, this problem is solved by simply normalizing the images individually so that they have zero mean and unit standard deviation before they are fed to the learning agent. In fact, some form of normalization is more or less standard practice when it comes to deep learning applied to images. This has previously been shown to aid the networks learning capabilities [28].

4.2 Annotations

The annotations were created by marking two points, the locations in the image where each human has its head and feet. Only about every tenth frame or so contain annotations made by an actual human. The frames in between are annotated by interpolating the head and feet positions. This could potentially cause some annotations to be of poor quality if, for example the camera is shaking a lot. It is up to the person annotating the sequence to insert sufficiently many manual annotations so as to ensure that the interpolated annotations are acceptable. Frames containing no humans are simply marked as “background”.

In this work the detection is based on bounding boxes, and since the annotations contain no information about the width of the human, this has to be estimated based on height. In the Caltech Pedestrian data set, it was found that the mean width to height aspect ratio of a human bounding box was 0.41 [7]. As this value is unlikely to change between data sets due to the similarity of humans in general, this value was used to create ground truth bounding boxes based on the head and feet positions. The current data set mostly contains humans that are in upright position, i.e. standing, walking or biking. No special treatment is therefore done for the small subset of humans that may be in other positions where this width to height constant does not apply, such as crouching or crawling. For simplicity, the generated bounding boxes are produced so that they are aligned with the image borders, so called axis-aligned.

4.3 Refinement

The available data is still somewhat rough, and some refinements have to be done in order to obtain a set suitable for training:

- All sequences are converted to have a consistent resolution of 320x240. This means that the sequences filmed with a higher resolution camera are down-sampled to match the resolution of the lower resolution sequences.
- Because the solution relies on processing whole images during training (as is standard practice in deep learning models), any frame where only a subset of the present humans are annotated has to be discarded. If this is not done, any non-annotated human found by the method will be considered an error by the evaluation and training metrics.
• By visual inspection it was determined that any human smaller than 10 pixels in height cannot be distinguished confidently without being able to perceive its movement over time. All frames containing humans smaller than this are thus discarded as well.

• Some sequences filmed by a stationary camera have an almost static background. Frames within these sequences labelled as background would be almost identical. To avoid duplicates in the refined data set, only a single background frame is extracted from these sequences.

Figure 7: Histogram of annotation heights in the refined data set. The height is calculated by taking the vertical difference between the head and feet positions.

The result of this refinement is a fully annotated data set containing about 120k individual frames. 40k of these are images containing humans, and the remaining 80k are background. The distribution of human size in this data set is shown in Figure 7. The histogram reveals that the majority of humans are between 10 and 40px in height. This also reflects the intended practical application of the method, i.e. long range surveillance. Any target larger than 40px is not a major concern. The full range of available heights are still included in this work because it might tell something about the generality of the methods.

4.4 Partitioning

Due to the fact that the data consists of video sequences it is fair to assume that the frames of each sequence are somewhat correlated. This should be taken into account when partitioning the data set into training and test sets. A problem
here is that the number of annotations in each sequence is highly varying due to the difference in sequence length, as well as the fact that the size of the annotated humans largely depends on the characteristic of the sequence. Any partitioning that works on a sequence level would most likely not be an ideal representation of the training data in terms of the human sizes, mainly due to the small number of sequences that actually contain any annotations. Although this is the case, the sequence level partitioning scheme is chosen in order to avoid correlation between test and training sets.

A test set is created by randomly sampling sequences from the refined data set such that the sampled set contains roughly 15% of the total number of annotated frames. A validation set is also created by using the same sampling scheme on the remaining sequences, such that the sampled set contain about 10% of the number of annotated frames remaining. The remaining sequences are declared as the training set. The final partitioning is presented in Table 1.

Table 1: The final partitioning of the data set.

<table>
<thead>
<tr>
<th>set# of</th>
<th>sequences</th>
<th>pos. frames</th>
<th>tot. humans</th>
<th>neg. frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training set</td>
<td>723</td>
<td>30453</td>
<td>46034</td>
<td>63687</td>
</tr>
<tr>
<td>Validation set</td>
<td>124</td>
<td>2269</td>
<td>2368</td>
<td>6794</td>
</tr>
<tr>
<td>Test set</td>
<td>15</td>
<td>4317</td>
<td>9556</td>
<td>12528</td>
</tr>
</tbody>
</table>
5 Pretraining

As stated earlier, deep networks require lots of training data in order to avoid overfitting. For the task of object detection, this amount of data is usually not available, possibly due to the long time it takes to manually annotate examples with location information. To overcome this, practically all reviewed work on object detection uses some form of pretraining that allows training the base network in some other way, before training on the detection task. The intuition behind this is that the network learns very general features during the pretraining, that also have application for similar tasks. These features can then be fine-tuned by training on the limited amount of available detection data.

The most popular approach used in the reviewed papers is to pretrain the base network on a classification task where lots of annotated data is available, such as the ImageNet [5] data set. There are some concerns with using this approach in the present work however. The ImageNet data set consists of images with three color channels, i.e. RGB, whereas infrared images have a single channel. One possible way of action would be to simply convert the RGB images to single channel gray scale images, and train the network on these. The effect of doing this is unclear because this approach disregards the fact that infrared images are visually very different than gray scale images taken of visible light. For example, the infrared images used here have a dynamic range of 14 bit, which is much higher than typical gray scale images that only have a 8 bit range of possible intensity values.

Pretraining is not exclusive to the object detection task. Before the existence of large labelled data sets stands a long tradition of using unsupervised learning for the pre-training of neural networks [2, 3, 8]. Unsupervised learning refers to the training of a model on unlabelled data. This is a very attractive learning scheme because it remedies the problem of having to obtain large amounts of annotated data. This is also well suited for the present work, as the majority portion of available data is unlabelled and hence unusable if purely supervised learning is used.

There are a couple of proposed ways in which this type of unsupervised pretraining can be performed. One quite recent invention is the generative adversarial network [15]. Here a generative model is pitted against a discriminative model. The goal of the generative model is to generate samples from the wanted distribution, and the goal of the discriminative model is to detect which samples are real, and which originate from the generative model. In this work however, an autoencoder will be used for the unsupervised pretraining of the feature extracting base network used by the detectors. The reason for choosing to use the autoencoder scheme is mainly due to practical reasons, as its architecture is very similar to that of the segmenting detector to be implemented later. Another reason is that it is fairly well documented in current literature, which helps when one is concerned with implementation details.
5.1 Autoencoder

An autoencoder (also called an autoassociator) is a neural network that learns in an unsupervised fashion by having the network reconstruct its input during training [2, 3, 20, 44].

It consists of two main parts, an encoder and a decoder, both of which can be implemented as neural networks (Figure 8). They are connected in such a way that the input is fed to the encoder, whose output in turn is fed to the decoder, that produces the final output. The whole network is trained to reconstruct its original input, that is, the target output of the decoder is the same as the input fed to the encoder. It has been shown that by doing so, the hidden layers of the network learn good representations of the data [20, 25, 33].

The encoder has the same structure as a typical feed forward neural network and its function can be viewed as producing a compressed representation of the input. The goal of the encoder during training is to produce a representation that captures as much information as possible about the input, i.e. it will become a very general feature extractor. The decoder is effectively a “mirroring” of the encoder, that tries to undo the operations performed by the encoder, and seeks to reproduce the original input value given the encoded representation created by the encoder.

By discarding the decoder part of the trained network the encoder alone can function as a feature extractor that can be utilized by for example a classifier. This is what is done when using the autoencoder for pretraining.

One potential hazard that has to be guarded against is that the network may learn the identity function, in which case no useful information about the data has been learned by the network. To prevent this, some regularization technique is applied. The most direct way is to limits the networks representational capabilities by introducing a ”bottleneck” for the encoded representation. This is implemented by having a reduced number of neurons in the output layer of the encoder, thereby forcing it to project the input signal onto a lower dimensional space. Other regularization techniques that has also been used is to add a small
perturbation to the input, so called denoising autoencoder [45], or to impose
sparsity on the hidden layers of the network by introducing a regularization
term to the objective function that limits the activations of the neurons, an
approach utilized in sparse autoencoders [25].

One regularization that is sometimes used is to let the weights be shared
between the encoder and decoder. In that case, the decoder uses the transpose
weight matrix of the encoder for each corresponding layer. This reduces the
total number of parameters of the model, and makes training faster.

5.1.1 Convolutional autoencoder

As a convolutional network will be used in this work, the corresponding solution
would be to create a convolutional autoencoder. Even though there are many
eamples of convolutional autoencoders being used in the literature [25, 33, 43],
details of exact implementation in these papers vary widely, or are event not
mentioned. The main concern for implementing a convolutional autoencoder
is how the encoder is to be implemented. A convolutional network consists of
two main operations, convolution and down-sampling. Following the paradigm
of having a decoder that reverses these operations, the decoder would have to
perform corresponding deconvolution and up-sampling.

The architecture for this type of decoder is remarkably similar to that of the
deconvolutional networks as proposed by Zeiler et al. [47, 48]. Here the decon-
volution operation is implemented as a transpose convolution. This is a normal
convolution operation, but where the filters are transposed before performing
the convolution. Having the deconvolution use transpose filters is mainly a
product of the application for which these deconvolutional networks were used
in this work, where sharing of weights between the encoder and decoder is re-
quired. When the filters are learnt, this operation can be implemented as a
normal convolution. The upsampling suggested here is implemented as a max
unpool operation. When using max unpooling, the max pooling operation is
modified to yield two different outputs, namely an index map that described

![Diagram of max pooling and unpooling](image-url)
the location of the values selected, as well as the downsampled feature map. The max unpooling operation later uses this index map together with the feature map output by the previous layer in the decoder to restore each value to their original position. This max pool-unpool scheme assumes that the decoder is symmetrical to that of the encoder when it comes to the number of up- and downsampling operations. For each max unpooling in the decoder exists a corresponding max pooling operation in the encoder that handles feature maps of equal size. The principle of this type of max pool-unpooling is depicted in Figure 9.

As it turns out, the approach described above has also been suggested as the architecture for convolutional autoencoders [43, 49], and later used for semantic segmentation [1, 34], where the segmentation network builds on the same basic principles as the convolutional autoencoder.
6 Detector implementation

For the purpose of this thesis, two detectors building on very different approaches are implemented and benchmarked. The first detector is based on using sliding windows in multiple levels in the feature hierarchy created by the convolutional base network, as described in Section 3.2.2. This detector is considered here to be a "safe" approach due to its proven success [4, 31]. The second detector is based on the deconvolutional approach used for semantic segmentation described in Section 3.2.3. This is an experimental approach that applies techniques currently used to solve the semantic segmentation problem. The reason for also implementing the second detector is due to its much simpler architecture.

6.1 Base network

Both detectors use the exact same base network. The base network is based on the convolutional part of the VGG16 [41] network. This is chosen as the base architecture because of its proven success, and popularity among related work [4, 31]. This network is built exclusively from convolutions with a filter size of $3 \times 3$ and stride $1 \times 1$, and max-pooling layers with a sample window of $2 \times 2$, and stride $2 \times 2$. The only modifications made to this architecture for the purpose of the present work is that the number of filters in each layer is slightly reduced. This is done to accommodate for the fact that it was designed for 3 channel (RGB) images, and not single channel images. A batch normalization [23] step is also added after each convolution operation. In total there are 13 convolutional layers and 5 max pooling layer arranged in the configuration depicted in Figure 10. For an input image of size $240 \times 320 \times 1$ the network outputs a feature map of size $8 \times 10 \times 256$.

The base network is trained separately using the convolutional autoencoder scheme introduced in Section 5. This allows for the utilization of the large amount of unlabelled data available. The loss function used for this training is the mean squared error of the pixel values.

6.2 Sliding window detector

The sliding window detector takes the encoder part of the base network and extends it by inserting classifiers in several levels in the convolutional feature hierarchy in order to allow for detection of objects of different scales. The implementation adopted here is based on SSD [31] and MS-CNN [4].

Classifiers. The classifiers in this case consist of convolutional filters that act as sliding windows to allow for detection at all possible locations in the feature map. Each classifier is trained to predict both a confidence score that tells how confident the classifier is that there is a human at the current location, and the coordinates for the bounding box of the object.
Figure 10: Architecture of the base network, based on the convolutional part of VGG16.
Two binary classifiers are used for the confidence score prediction. One is trained to classify a location as background/non-background, and the other to classify a location as human/non-human. These two classifiers will have the inverted targets of one another, and the final confidence score is given after performing a softmax normalization over these two predictions for a specific location. The bounding box prediction is defined so that the classifier does not predict the global image coordinates for the bounding box, but rather an offset from a default bounding box associated with each classifier. This default box is defined separately for each sliding window and is based on its window size and in which level in the feature hierarchy the classifier is connected to.

**Architecture.** Based on the desired object sizes to be detected, the base network is extended with six different sliding window classifiers, distributed onto three different levels. In order to not be forced to apply the classifier too close to the input, the input image is upscaled so that its height and width are doubled, giving a final image resolution of 480 x 640px. This means that the smallest objects are now visible further down in the feature hierarchy and more convolutional layers can be used for feature extraction. Despite this, the classifiers connected closest to the inputs are extended to have an additional convolutional layer. This has been suggested to increase the performance by not letting the gradient from these classifiers propagate directly into the base network, which could potentially make the gradient signals coming from the other classifiers insignificant in comparison [4]. The architecture of this detector is depicted in Figure 11.

**Generating training targets.** Before the detector can be trained, proper training targets have to be generated for each image in the data set. For each image, each location that the sliding window classifiers are applied to has to be tested in order to determine if there is an object at that position, and if that is the case, also determine the bounding box regression targets based on the offset between the objects ground truth box and the classifiers default box. In this implementation, a sliding window position is considered positive if there is an intersection over union (IOU) overlap between the default box and the ground truth box that is larger than 0.5.

The regression is not performed on raw pixel coordinates, but rather in a transformed, scale invariant space that aids the regression. This transformation was introduced in the original R-CNN paper [13] and is what will be used here. For a ground truth bounding box $G$ and a default bounding box $D$, the prediction targets $R$ for the bounding box regression are defined as:
Figure 11: Architecture of the sliding window detector.

- **Input**
- **Filter size**
  - 3\(\times\)3 \(\times\) 3 \(\times\) 32
  - 3\(\times\)3 \(\times\) 3 \(\times\) 32
  - 3\(\times\)3 \(\times\) 3 \(\times\) 64
  - 3\(\times\)3 \(\times\) 3 \(\times\) 64
  - 3\(\times\)3 \(\times\) 3 \(\times\) 128
  - 3\(\times\)3 \(\times\) 3 \(\times\) 128
  - 3\(\times\)3 \(\times\) 3 \(\times\) 128
  - 3\(\times\)3 \(\times\) 3 \(\times\) 256
  - 3\(\times\)3 \(\times\) 3 \(\times\) 256
  - 3\(\times\)3 \(\times\) 3 \(\times\) 256

- **3\(\times\)3 \(\times\) 256 \(\times\) 256**
- **5\(\times\)3 \(\times\) 256 \(\times\) 6**
- **5\(\times\)3 \(\times\) 256 \(\times\) 6**
- **5\(\times\)3 \(\times\) 256 \(\times\) 6**

- **Convolution + Batch normalization + ReLu**
- **Max pooling, window size 2 \(\times\) 2, stride 2 \(\times\) 2**
- **Classifier (Convolution + ReLu + Softmax)**

Figure 11: Architecture of the sliding window detector.
where the subscripts $w$ and $h$ denote the width and height of the bounding box respectively, and $x$ and $y$ denote the $x$ and $y$ coordinates of the center of the bounding box. The first two transformations specify a scale invariant translation of the center of the default bounding box, and the last two specify log-space translations of the default bounding box’s width and height.

To transform a bounding box prediction $P$ into an actual bounding box in pixel coordinates $\hat{P}$, the transformation is inverted, and an offset is added according to the default bounding box for the specific location:

$$
\hat{P}_x = D_w \cdot P_x + D_x \\
\hat{P}_y = D_h \cdot P_y + D_y \\
\hat{P}_w = D_w \cdot \exp(P_w) \\
\hat{P}_h = D_h \cdot \exp(P_h)
$$

**Loss function.** The objective function used during the training consists of terms related to the two different tasks to solve: the location-wise classification ($l_{cls}$) and bounding box regression ($l_{loc}$). The classification loss is defined as a pixel-wise cross entropy between the classification targets $T$ and the classification prediction $C$. The localization loss for the bounding box prediction is defined as a per coordinate absolute value error.

$$
l_{cls} = - \sum_{i \in \{pos, neg\}} T_i \log(C_i) \\
l_{loc} = [T_{pos} > 0] \sum_{i \in \{x, y, w, h\}} |G_i - \hat{P}_i| \\
L = l_{cls} + l_{loc}
$$

In $l_{loc}$, $[T_{pos} > 0]$ ensures that only locations where the default box has an IOU larger than 0.5 with a ground truth box are included in the localization loss.
Extracting final detection. The final detection is determined by taking the result from all classifiers, and performing a non-maximum suppression of the predicted bounding boxes. This is implemented by sorting the predicted bounding boxes according to the confidence score predicted by the classifier, and removing the bounding boxes that have a sufficiently high IOU overlap with another bounding box with higher score (in this work an IOU threshold of 0.5 is used). A confidence threshold is then applied to filter out bounding boxes that are associated with a too low confidence.

6.3 Deconvolutional detector

The sliding window approach described above is cumbersome for several reasons because (i) the network has to be trained to not only output a value of detection confidence but also the coordinates of the supposed bounding box of the object, which involves having to additionally train the network for coordinate regression, (ii) the sliding windows have to be designed so that they cover the full range of possible object sizes, and (iii) the results from these different detections have to be combined in some clever way to produce the final detection output. This results in having to spend a great deal of time and care when designing the network and preparing the data, in order to allow proper training targets to be generated.

A more elegant solution that remedy these problems would be to train a network to produce a single map of the whole image where object presence, location and size are indicated by means of some graphical element. One such approach is to employ a deconvolutional network to produce a segmentation map of the whole image. This has previously been explored for semantic segmentation with promising results [32, 34].

Here, this technique is adopted to allow for segmentation based on object bounding boxes. As no proper segmentation annotations are available, the segmentation maps to be used as targets during training are created by extracting the rectangular regions enclosing each human, given by the bounding box annotations. The segmentation network is essentially the same as the autoencoder, but is now trained on reconstructing only the bounding box regions containing humans, and to suppress the rest of the image. Intuitively, even though the extracted regions contain some background surrounding each human, averaged over many regions the human would be invariant, and therefore the main focus of the segmentation.

Architecture. The architecture used for this network is exactly the same as for the autoencoder used for pretraining, with one small difference. The last layer of the decoder is replaced with a pixel wise classifier. The pixel wise classifier in this case consists of a $3 \times 3$ convolutional filter. Actually, as with the sliding window detector, two binary classifiers are used. One classifies a pixel as being human or not, and the other classifies a pixel as being background or not. A softmax normalization is then performed over the result from these
two classifiers to produce the final confidence score. The architecture for the deconvolutional network used here is depicted in Figure 12.

**Generating training targets.** The targets to use during training are created by simply labelling the individual pixels that lie within a ground truth bounding box as being human, and the rest as background. An example of an input-target pair is shown in Figure 13.

**Loss function.** The objective function used for this network is a per pixel cross entropy between the target $T$ and the predicted segmentation map $C$:

$$L = - \sum_{i \in \{pos,neg\}} T_i \log(C_i)$$

**Extracting final detection.** The final stage of the detector is the extraction of bounding boxes from the segmentation map. This is done by first running Otsu thresholding on the image. The thresholded image is then used to extract connected components, which in turn are converted into bounding boxes. Any connected component with a height less than 10px, or a maximum intensity value lower than a specified threshold are filtered out.
Figure 12: Deconvolutional network used for segmentation of object bounding box regions.

Figure 12: Deconvolutional network used for segmentation of object bounding box regions.
7 Evaluation

In order to evaluate how well the resulting models perform on the task, they are trained and benchmarked on the available data. They are also compared to a non-deep learning method consisting of a cascade classifier using HAAR features.

7.1 Experimental setup

Data sets. The data set described in Section 4 is used for the training and benchmarking. The validation set is used for hold out validation, i.e. it is only used for evaluation purposes, and never to train the models.

Baseline comparison method. The two implemented deep learning models are compared to a model that uses "classical" image analysis methods. It consists of a boosted cascade of decision trees using HAAR features for classification, building on the Viola-Jones detection framework [46]. It is implemented by having a sliding window scan the image in multiple scales, and classifying each location as containing a human or not. The cascade classifier is trained and evaluated on the same data as the deep learning models. This baseline builds on the same principles as the methods currently used by the employer today. This type of detector however does not handle multiple classes "out-of-the-box", which is a disadvantage compared to the deep learning approaches studied here.

Performance measures. The output from each detector is a set of bounding boxes that predicts the location of humans within the input images. To be able to determine which of these bounding boxes are correct, the overlap between the predicted bounding boxes and the ground truth boxes is compared. The overlap measure used is the intersection over union (IOU) that gives the quotient between the area of intersection and the union of the two boxes. An overlap of 1 is a perfect alignment of the two boxes, and an overlap of 0 means that the two boxes are disjoint. A bounding box is considered to be a correct prediction if it has an overlap of 0.3 or more with at least one ground truth box. Most general
object detection tasks require an overlap of 0.5 or more [10, 30], but considering that the objects here can be very small, it was decided to be a bit leaner in that regard. Each ground truth box is mapped to at most one prediction. If there are several predictions that overlap with the same ground truth box, only one is considered correct, and the rest false. To determine the performance of a detector, the precision and recall measures are used. These measures are based on the number of true positive (TP), false positive (FP), and false negative (FN) predictions, and are defined as:

\[
\text{precision} = \frac{TP}{TP + FP} \\
\text{recall} = \frac{TP}{TP + FN}
\]

The precision can be interpreted as a measure of how many positive predictions actually are true. The recall is a measure of how many of the true objects are found. A high precision means that there are few false positive predictions, and a high recall means that the model finds many true objects.

Frequently, a trade-off has to be done regarding the precision and recall of a detector. Here, the F1-score is used to determine the absolute performance of a detector, so that they can be easily compared to one another. The F1-score is defined as the harmonic mean of the precision and recall:

\[
\text{F1-score} = 2 \times \frac{1}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}
\]

### 7.2 Training

During training, the Adam [24] optimizer is used for weight updates, with an initial learning rate of 0.01.

**Parameter initialization.** The parameters of the convolutional filters are initialized to values sampled from a normal distribution with zero mean and standard deviation 0.1. All bias parameters are initialized to 0.1.

For the batch normalization, \(\gamma\) is initialized to 1, and \(\beta\) to 0.1. The running mean and variance are updated according to an exponential moving average with a decay factor of 0.99.

**Example sampling.** A mini batch of size 8 is used, where the examples are sampled uniformly at random without substitution from the training set. When there are no examples left, the sampling starts over from the beginning.

**Data augmentation.** Besides normalizing the input images so that they have a zero mean and unit variance, some augmentation techniques are applied to further stretch the amount of variance in the data. First, with a probability
of 0.5, each example is mirrored along the horizontal axis. After this each example is cropped at a random location, with a size selected uniformly at random such that the final image has a minimum width and height equal to 0.6 of the original image. The width and height are sampled independently, meaning that the aspect ratio is not necessarily preserved. The cropped image is then rescaled to the original size of the image.

Training the base network. The base network is trained using the autoencoder setup described in Section 5.1, where the weights are shared between the encoder and decoder. It is trained on all available data except for the sequences belonging to the test and validation sets, this includes both annotated and non-annotated data. The autoencoder is trained for a total of 1 million iterations which amounts to roughly 12 epochs. This is quite low when considering related work \[16, 47\], but due to time constraints the training had to be limited.

Detector training. The weights learnt from the base network pretraining are used to initialize each of the detectors that in turn are trained on the annotated training set. This training is also limited to 1 million iterations due to time constraints, but in the ideal case it would be trained until overfitting was observed. During training, the performance of the detectors on the test and validation sets are evaluated every 1000 iteration. Figure 14 shows the performance of the detector on the validation set for every 1000 iteration, and Figure 15 shows the same for the test set.

Early stopping. The validation set is then used for early stopping by selecting the iteration where the validation performance (F1-score) was highest. The precision and recall are then plotted for different confidence thresholds for this training iteration (see Figure 16). The best F1-score was then determined for each detector and is shown in Table 2.
7.3 Results

Figure 14: Validation performance over training iteration.

Figure 15: Test performance over training iteration.
7.4 Analysis

**Unstable training.** One of the first realizations when inspecting the performance graphs is how unstable the training seems to be. This is especially visible on the performance graph for the validation set shown in Figure 14, where the precision and recall fluctuate highly between training iterations. The first suspicion was that this is most likely due to a combination of relatively small mini-batch size as well as a too aggressive learning rate, but after experimentation with different learning rates that yielded similar results, the most likely cause was concluded to be the size of the mini-batches used during training. Unfortunately, it is the hardware that sets the maximum limit for the batch size and the size used in these experiments was the maximum allowed by the available hardware.

**Partition bias.** Another thing to note is the significant difference between the performance on the validation and test sets. The test set (Figure 15) seems to be much harder than the validation set (Figure 14) for both detectors. This was a concern that arose early in the work, where it was suspected that the test set partitioning may become biased due to the small number of annotated sequences (Section 4). The primary suspicions as to what this bias may be caused by is either that there is a bias in the size of the humans between test and training set, or that there is some other visual bias that cause the examples in the test set to be harder.

<table>
<thead>
<tr>
<th></th>
<th>Deconvolutional</th>
<th>Sliding Window</th>
<th>HAAR Cascade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation set</td>
<td>0.96</td>
<td>0.94</td>
<td>0.92</td>
</tr>
<tr>
<td>Test set</td>
<td>0.68</td>
<td>0.80</td>
<td>0.69</td>
</tr>
</tbody>
</table>
Figure 17: Distribution of human heights in the final data set partitioning. Note that the y-axis is log-scale.

The histogram of annotation heights for the different data set partitions shown in Figure 17 reveals that there is in fact a bias towards larger humans in the test set. Between the heights of about 100 and 170 pixels, the test set even contains more examples than the training set, while the validation set contains no examples within this range. However, looking at Figures 18 and 19, that display the height of the humans for all true positive and false negative detections for both detectors, it is revealed that the majority of missed detections are in fact within a range of heights that are well represented in the training set.
A visual inspection of the sequences in the test set for which the detectors have low accuracy further reveals that the test sequences contain some outlier sequences. Figure 20 shows two images taken from two different sequences in the test set, both of which the detectors fail to find any humans in. In the image to the left in Figure 20 the human is barely visible to the human eye. By enlarging the human it is possible to see some faint contours. Clearly, this sequence has a very bad contrast ratio between the human and the background. In the image to the right, the contrast is also low, but the humans are still visible to us. There is another problem here though; the humans are actually darker than the background. This seems to be an unusual scenario when looking at other sequences in the data set. The reason for this is that the background is warmer than the humans, something that the training set does not cover well. The training set most likely needs to be extended with more examples where the
Figure 20: Hard examples from two different sequences in the test set. Both detectors fail to find any humans in these images.

background is warmer than the humans. An interesting experiment would be to train on images where the intensity has been inverted to see if this improves the performance here.

**Notes regarding the precision/recall curve.** When looking at the precision/recall curves for the detectors in Figure 16 the curve for the deconvolutional detector has a very steep drop off in performance when going too far in either direction. This can be blamed on the very primitive solution for how the final bounding boxes are extracted from the segmentation map output by the network. Since the map is thresholded globally, regions with lower confidence scores probably gets assigned to the background cluster. Using adaptive thresholding may be a solution to this, but there may be a large penalty to the speed of the detector.

**Performance compared to baseline** Both the sliding window deep learning detector, and the cascade classifier used as a baseline, can be considered to be ”off-the-shelf” solutions to the detection problem. They are both well documented and accepted methods. In the precision/recall curve in Figure 16 we can see that the deep learning method outperforms the cascade classifier by a large margin. This is especially visible on the test set, where the deep learning method shows much greater ability to generalize. The experimental approach that consists of the deconvolutional detector, does not require the input images to be upsampled before being fed to the detector. This is a very big advantage when inference speed is of concern, since upscaling the input results in an increase in the number of computations performed by the neural network. It does however suffer from the thresholding method used to extract detections, making it perform poorly when adjusting the threshold too much in either direction.
8 Conclusions

This work has been an investigation into how deep learning can be applied to the task of detecting humans in infrared images. It consists of background research in the field of deep learning applied to object detection, as well as the implementation and benchmarking of two different approaches to solving this problem. Besides the fact that deep learning is applied to infrared images, the most significant aspect of this work is that the objects to be detected are very small compared to the more general object detection tasks such as MS-COCO[30] and ImageNet[39]. This is something that is not well covered in current literature. Furthermore, a convolutional autoencoder was used to pretrain the feature extraction part of the implemented detectors.

When applying deep learning to infrared images, it is important to make sure that the data set represents the different types of situations that arise due to the wide dynamic range of these images. In the case of detecting humans, the data set needs to contain enough images to represent the different types of contrast scenarios that may exist. The most direct way is to make sure that these types of situations are covered during data acquisition. It may be possible to perform some augmentation on the available data, but details of this would have to be subject to further study. Furthermore, when background temperatures approach the temperature of humans, the contrast between human and background becomes extremely low and makes detection hard. In such cases, the infrared sensor may have to be complemented with a visual-light sensor.

Even though no rigid experimentation was done regarding the unsupervised pretraining it was noted that the overall detection performance increased with longer pretraining, meaning that the autoencoder setup was successful.

Detecting small objects is a problem with general deep learning object detection methods. The deconvolutional method implemented in this work however seems to handle this fine without the need to upscale the input image, as required by the sliding window method. In general though, it is hard to see a possible method for scale invariant detection that does not require some kind of hyperparameter for this purpose.

In general, the capabilities of deep learning are impressive, but even with high-end hardware the developing cycle is extremely long. During the course of this work it was not uncommon for the training to go on for a full week non-stop.

More than anything, this work has given valuable insight into the rapidly evolving field of deep learning. New methods are published almost daily, and keeping up with recent work is a full time engagement. While the amount of work being published is huge, one thing that made the research harder was the fact that many of the published papers had very poor descriptions when it comes to details of the proposed methods.
8.1 Future work

The most direct way to proceed based on this initial study is to experiment with different hyperparameters and network architectures.

The extraction of bounding boxes by thresholding the output from the de-convolutional detector is not an ideal way of doing this, as shown by the performance plot in Figure 16. A more sophisticated implementation should rely on alternative methods for this purpose. An example could be to apply a nonlocal maximum suppression on the confidence map output by the deconvolutional detector, and use this as the base for bounding box extraction. Exact details of this would have to be investigated further.

Regarding the training targets for the deconvolutional detector, they were set as binary masks over the image. When considering that the network is initialized from a pretrained autoencoder these types of targets may not be ideal. The autoencoder is trained to reconstruct the image with all its details, but when the targets are changed to be binary, the last couple of layers will have to be retrained to “smooth out” these details in order to obtain the binary segmentation map. More proper targets might be obtained by basing the segmentation targets on the actual image, and instead increasing the intensity values for the regions containing humans, and decreasing the intensity values for background regions. The final pixel wise softmax would then adjust the confidence values properly.

In the beginning of this work, a choice was made where an autoencoder were to be used for pretraining of the detection network. This was considered a safe approach since the alternative was to pretrain the network on visual-light images, and the effects of doing so were unclear. However, in retrospect it may have been more effective to choose the latter, since many features learnt from visual-light images probably are transferable to the infrared spectrum after appropriate normalization of the image.
## A Technical recipe

The following tables summarize the design and parameter choices made in this work.

### Table 3: Common implementation details

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Convolutional part of VGG16 [41] (see Figure 10)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initialization</strong></td>
<td></td>
</tr>
<tr>
<td>Convolutional filters</td>
<td>$\mathcal{N}(0, 0.1)$</td>
</tr>
<tr>
<td>Biases</td>
<td>0.1</td>
</tr>
<tr>
<td>Batchnorm $\gamma$</td>
<td>1</td>
</tr>
<tr>
<td>Batchnorm $\beta$</td>
<td>0.1</td>
</tr>
<tr>
<td><strong>Training</strong></td>
<td></td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.01</td>
</tr>
<tr>
<td>Input normalization</td>
<td>Zero mean, unit standard deviation</td>
</tr>
<tr>
<td><strong>Data augmentation</strong></td>
<td></td>
</tr>
<tr>
<td>Random horizontal mirror</td>
<td></td>
</tr>
<tr>
<td>Random crop and resize</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4: Autoencoder

<table>
<thead>
<tr>
<th><strong>Training</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Loss function</td>
<td>Mean squared error</td>
</tr>
<tr>
<td>Batch size</td>
<td>16</td>
</tr>
</tbody>
</table>

### Table 5: Sliding window detector

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Based on SSD [31], MS-CNN [4] (see Figure 11)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Training</strong></td>
<td></td>
</tr>
<tr>
<td>Loss function (classification)</td>
<td>Cross entropy</td>
</tr>
<tr>
<td>Loss function (coordinate regression)</td>
<td>Absolute value of the error</td>
</tr>
<tr>
<td>Batch size</td>
<td>8</td>
</tr>
</tbody>
</table>
Table 6: Deconvolutional detector

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Based on previous work on semantic segmentation\cite{32, 34} (see Figure 12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td></td>
</tr>
<tr>
<td>Loss function</td>
<td>Cross entropy</td>
</tr>
<tr>
<td>Batch size</td>
<td>8</td>
</tr>
</tbody>
</table>
References


