Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2017

Detecting Rails in Images from a Train-Mounted Thermal Camera using a Convolutional Neural Network

Magnus Wedberg

LINKÖPING UNIVERSITY

Master of Science Thesis in Electrical Engineering

Detecting Rails in Images from a Train-Mounted Thermal Camera using a Convolutional Neural Network

Magnus Wedberg

LiTH-ISY-EX--17/5058--SE

Supervisor: **Jörgen Ahlberg**
ISY, Linköpings universitet
Amanda Berg
Termisk Systemteknik AB

Examiner: **Michael Felsberg**
ISY, Linköpings universitet

Computer Vision Laboratory
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Sammanfattning

Nu och då inträffar tågolyckor. Kollisioner mellan tåg och objekt såsom djur, människor, bilar och nedfallna träd kan resultera i olycksfall, allvarliga skador på tåget och kraftiga förseningar i tågtrafiken. Följaktligen är tågkollisioner ett stort problem med konsekvenser som påverkar samhället avsevärt.

Företaget *Termisk Systemteknik AB* har på uppdrag av *Rindi Solutions AB* undersökt möjligheten att detektera avvikande objekt på rälsen med hjälp av en tågmonterad värmekamera. Även räls detekteras för att kunna avgöra om ett avvikande objekt befinner sig på eller bredvid rälsen. Rälsdetekteringen fungerar emellertid inte på ett tillfredsställande sätt på långt avstånd.

Syftet med detta examensarbete är att förbättra den föregående rälsdetektorn på långt avstånd genom att använda maskininlärning och i synnerhet *deep learning* och ett neuralt faltningsnätverk. Av intresse är också att undersöka om det finns några fördelar med att använda tvärmodal *transfer learning*.

Ett annoterat dataset för träning och testning producerades manuellt. Dessutom togs en kostnadsfunktion anpassad för räls fram. Kostnadsfunktionen användes både för att förbättra rälsdetektorn under träning och för att utvärdera rälsdetektorn under testning. Slutligen utvärderades åtta olika angreppssätt som vart och ett resulterade i en egen rälsdetektor.

Flera av rälsdetektorerna, och i synnerhet de rälsdetektorer som använde sig av tvärmodal *transfer learning*, presterade bättre än den föregående rälsdetektorn. Således visar de nya rälsdetektorerna stor potential för detektion av räls i termiskt infraröda bilder.

Abstract

Now and then train accidents occur. Collisions between trains and objects such as animals, humans, cars, and fallen trees can result in casualties, severe damage on the train, and delays in the train traffic. Thus, train collisions are a considerable problem with consequences affecting society substantially.

The company *Termisk Systemteknik AB* has on commission by *Rindi Solutions AB* investigated the possibility to detect anomalies on the railway using a train-mounted thermal imaging camera. Rails are also detected in order to determine if an anomaly is on the rail or not. However, the rail detection method does not work satisfactory at long range.

The purpose of this master's thesis is to improve the previous rail detector at long range by using machine learning, and in particular deep learning and a convolutional neural network. Of interest is also to investigate if there are any advantages using cross-modal transfer learning.

A labelled dataset for training and testing was produced manually. Also, a loss function tailored to the particular problem at hand was constructed. The loss function was used both for improving the system during training and evaluate the system's performance during testing. Finally, eight different approaches were evaluated, each one resulting in a different rail detector.

Several of the rail detectors, and in particular all the rail detectors using cross-modal transfer learning, perform better than the previous rail detector. Thus, the new rail detectors show great potential to the rail detection problem.

Acknowledgments

This thesis work was possible thanks to the contribution of many persons, and I appreciate all the help I have been given. First of all, I would like to thank my supervisors Amanda and Jörgen for always being there whenever needed, be it reading thesis drafts on Saturday nights or deriving mathematical expressions on sticky notes in a hotel across the Atlantic.

I would also like to give thanks to all employees at *Termisk Systemteknik AB* for showing interest in my work and for being part of appreciated distractions during the coffee breaks.

Finally, I would like to thank my wife for being patient with my studies throughout my whole university education, and for all your love and support during the writing of this thesis.

Linköping, June 2017
Magnus Wedberg

Contents

Notation

Abbreviations

Abbreviation	Interpretation
CNN	Convolutional neural network
ReLU	Rectified linear unit
SGD	Stochastic gradient descent

1 Introduction

1.1 Background

Now and then train accidents occur. Collisions between trains and objects such as animals, humans, cars, and fallen trees can result in casualties, severe damage on the train, and delays in the train traffic. Thus, train collisions are a considerable problem with consequences affecting society substantially. Therefore, collisions should be avoided if possible.

A train often travels at a speed where the stopping distance is about one kilometre. Because of obstacles (anomalies) in the terrain and tough weather conditions it can be difficult for the train driver to see anomalies on the railway far away, especially in the dark. Thus, it is often hard to stop in time to avoid a collision.

The company *Termisk Systemteknik AB* has on commission by *Rindi Solutions AB* investigated the possibility to detect anomalies on the railway using a train-mounted thermal imaging camera to reduce stopping distance [5]. A thermal camera was used because of its capability to see in the dark, since it is independent of illumination.

The system implemented in [5] detects both anomalies and rails. The rails are detected in order to determine if an anomaly is on the rail or not. However, the rail detection method proposed does not work satisfactory at long range, among other reasons because of the assumption that the rail has constant curvature, which only holds to a certain extent. It is immensely important to find the rail at long range. One reason is that the train's large stopping distance makes it difficult to avoid or reduce the consequences of a collision if the train brakes too late. An-

other reason is to make a better estimate of how threatening detected anomalies really are, judging by their rail vicinity.

Earlier research treating the rail detection problem with visual cameras has to some extent successfully approximated a railway track by a model based on graph theory [23]. In a dataset with 362 frames containing only curved tracks, 60% of the tracks are found completely or partially. However, in another dataset with 2044 frames (including only straight tracks), 90% of the tracks are found completely, and 7% of the tracks partially. Thus, since promising rail models have been designed, semantic segmentation is not the only obvious approach to find the rail in images.

A suggested improvement in general is to take the environment around the rails into consideration, since humans can surmise the rail localization well even though it is not in clear sight from the camera [23]. It is also pointed out that illumination changes are a significant problem with visual cameras. It has been noted that sharp curves are problematic, especially in combination with occlusions [41].

Over the past few years deep learning networks have proved to be a great success in many pattern recognition and object detection problems, winning numerous competitions [30]. It has also been suggested that deep learning with a convolutional neural network (CNN) should be the main candidate in practically any visual recognition problem [25]. Moreover, CNNs have been used for lane detection with success [14]. Lane detection problems have some similarities with rail detection, although there are significant differences [5]. CNNs are inspired by the human visual system [13], a system which can see structures and patterns even though they are partially occluded, see Figure 1.1. As [24, 40] indicates, CNNs could possibly be used advantageously to find aforementioned patterns in images despite occlusion.

1.2 Problem description

The purpose of this master's thesis is to evaluate whether it is possible to improve the rail detection in the current system [5] by using a CNN trained with supervised learning. Here, improvement means to be able to detect the rail more accurate further away in an image. It is also of interest to evaluate real-time performance and how much training data is needed.

The methodology is as follows:

1. Execute a literature study on object detection and machine learning in general, especially deep learning and CNNs, in order to thoroughly apprehend the problem at hand.

2. Define a suitable representation and a way to evaluate the performance.

Figure 1.1: *Thermal image of a railway. Although the rail is occluded by a hill to the left, it is easy to perceive the direction and location of the rail in the curve, judging by the surrounding cliffs.*

3. Extend existing labeling software to support rail labeling according to chosen representation.

4. Determine a reasonable amount of training data and label it with above mentioned labeling software.

5. Investigate available code frameworks for machine learning and choose an appropriate candidate.

6. Implement and train a CNN in the chosen framework.

7. Design and evaluate a suitable loss function.

8. Test the implemented CNN, evaluate, and suggest improvements.

1.2.1 Labeling

Since the approach is supervised learning, labeled images are needed. The labels should hold information about where the rail is located. Since the representation is yet to be decided, labels have to be produced manually as part of the thesis work.

1.3 Difficulties

There are many problems with detecting the railway far away in an image. The rail might be occluded by the surrounding terrain, snow can cover the rails, or the rails might even be outside the image. Tough weather conditions might also reduce image quality, especially at large distances. Moreover, in a curve, both rails will not necessarily be visible. Also, the resolution representing the rail far away is lower in relation to the area closer to the camera, which is problematic. Especially in the thermal case, since thermal cameras in general capture images with lower resolution than modern visual cameras. Another problem is that one cannot always assume that the railway lies in a plane since the ground is not entirely flat.

In this thesis work, a CNN is used to detect rails. CNNs require lots of data and time to be trained. The large amount of time needed to be spent on training is a problem if access to state-of-the-art hardware is limited or non-existent.

The rail detector implemented as a CNN should ideally be usable in real-time in order to avoid delaying the braking of the train even more. Another difficulty is how to measure the detection performance in a reasonable way.

1.4 Limitations

This thesis work is limited to 20 weeks. Obviously, some aspects have therefore been treated more lightly:

- Labeling of data cannot take unreasonable proportions of available time.
- Training time is highly dependent on accessible hardware. Since access to state-of-the-art hardware is limited, the result is affected thereof.
- Only a few different ways of initiating the weights of the CNN are tested.
- Railway switches are not treated, only one railway track will be estimated in each frame.

1.5 Thesis outline

The thesis is structured as follows:

Chapter 2 provides background theory required for the understanding of this thesis.

Chapter 3 presents the dataset, the choice of representation, and the labeling procedure.

Chapter 4 introduces the network architecture as well as the application specific loss function.

Chapter 5 describes the tests that were performed and what results that were acquired.

Chapter 6 discusses the results and relates them to the problem description. Finally, the chapter proposes future work.

2
Theory

This chapter describes the theory required for the understanding of this report. Starting out with basics concerning differences between thermal and visual imaging, it is succeeded by fundamentals and definitions of machine learning in general. Towards the end, the training and construction of CNNs will be explained.

More information about the machine learning concepts mentioned in this chapter are available in [6, 12, 13] and information about thermal imaging is found in [26].

2.1 Thermal imaging

In this thesis work, the images are collected by a thermal camera. Even though this fact is not crucial for the neural network design, there are still some aspects which are important to point out.

2.1.1 Thermal images

Thermal cameras capture images in the thermal infrared (TIR) band, a part of the electromagnetic spectrum which is not visible for the human eye and (mostly) consists of emitted radiation. This band can be subdivided into midwave infrared (MWIR, 3-5 μm) and longwave infrared (LWIR, 8-12 μm) [4]. Other parts of the infrared (IR) band such as near infrared (NIR, 0.7-1 μm) and shortwave (SWIR, 1-3 μm) consist (mostly) of reflected radiation.

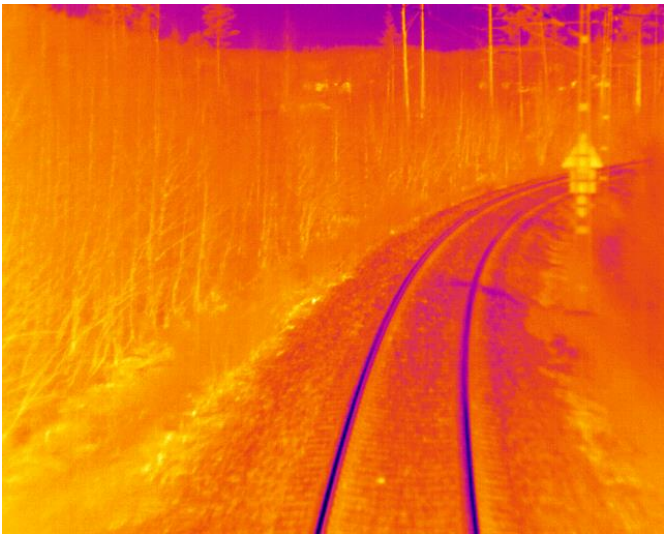


Figure 2.1: *Thermal image of a railway captured in the LWIR band, using pseudo-colors [12] to visualize temperature in the scene. The rails appear to have different temperature at the sides compared to the area pointing upwards, when in fact the shiny metallic surface in the upward pointing area reflects the thermal radiation from the sky, and reflectance affects the calculation of temperature. As a result, the rails can easily be spotted in the image.*

Emissivity is used along with the atmospheric attenuation of radiation in order to determine the temperature in a scene captured with a thermal camera [4]. When the emissivity of an object in a scene is not known, it is approximated.

Emissivity is a measure of an object's ability to emit thermal radiation, defined as the ratio of its thermal radiation to that of a blackbody at the same temperature [26]. Emissivity is a dimensionless number between 0 and 1 and it depends both on the material itself as well as of the structure of the surface. A value of 0 corresponds to a perfect reflector while a value of 1 is a perfect emitter. The importance of reflectance in thermal images of railways is substantial, as pointed out in Figure 2.1. Rails are metallic and metals usually have a high reflectance, while more earthly materials such as soil, gravel, trees, and foliage usually have low reflectance [26]. This difference implies that rails easily can be spotted in the thermal images even though they probably have approximately the same temperature as the surrounding ground.

Thermal imaging cameras are expensive when compared to visual cameras. The resolution is often lower, and typically only one channel is available. On the other hand, since a thermal camera measures thermal radiation, the camera does not need an illumination source to depict a scene. Since most of the measured radiation is emitted, no shadows will occur in thermal images. However, effects similar to shadows can occur in terms of thermal radiation reflections in materi-

als [22].

As to noise, thermal images typically have more blooming and more dead pixels compared to a visual camera [4]. Also, when temperature is to be measured, the heat generated from the camera itself (especially during startup) cannot be neglected as it often stands for a substantial percentage of the incoming radiation. Internal sensors are therefore used to calibrate the camera in order to handle these unwanted effects.

2.1.2 Sensor used for data collection

The sensor used for data collection is a FLIR A65 capturing images with 640x512 pixels and having a spectral range of 7.5-13 μm (LWIR) [1]. The A65 takes ~50 images per second and has an uncooled detector. The single channel images are stored with 16 bits (since ~14 bits are used) per pixel.

2.2 Machine learning

Machine learning is a wide subject of research and only the parts relevant for this thesis are covered in this section. In Section 2.2.1, the fundamentals of machine learning are introduced, succeeded by representation learning in Section 2.2.2, which explains systems that learn features themselves.

2.2.1 Fundamental definitions

Machine learning is a field within computer science which studies algorithmic approaches to learn from data. Some fundamental definitions are described below.

Classification vs. regression

Classification is the task of specifying which class a certain sample (e.g. an image) belongs to [6]. It can be as simple as the case with only two classes, i.e. the binary class belonging of a sample $\mathbb{R}^n \rightarrow \{0, 1\}$, where a sample either belongs to the first or the second class. The result is thus discrete, in contrast to the continuous result from regression. For example, when the sample is an image, classification can be used to determine if the image depicts a bacteria or not, while regression can be used to determine how many bacteria there are in the image.

Representation and features

The data that is used by a machine learning algorithm must be represented in some way. For example, if a face is to be detected in an image sample, a *representation* [12] of a face must be determined. In [13], a *feature* is described as one piece of information in the representation. Thus, the representation can consist of a feature vector formed by many features.

Supervised learning

Supervised learning is a learning paradigm where the system learns from a series of labeled samples called *training data* [6]. By using the training data as input to a function, one can compare the corresponding output with the desired labels (also *ground truth*) and adjust the function parameters by optimization using a *loss function*.

Another set of labeled data called *test data* is then used as input to the learned function. The output of the function with test data as input is used to evaluate its performance. The performance is a measure of how well the function can generalize new unknown data, based on what the function has learned from the known training data. Notably the system can never become better than the supervisor that labeled the training data w.r.t. the original loss function.

Neural networks

A neural network is a machine learning approach that is based on how the brain makes decisions [12]. The brain consists of a huge amount of neurons that are connected and transmit information with electrical and chemical impulses. A neural network is a model of the brain, with artificial neurons representing biological neurons in terms of mathematical functions. The network is organized in layers and the artificial neurons are connected fully or locally between the different layers. Usually, the inputs to a neuron are summed together and weighted to form a new signal, which is then fed to an activation function which introduces a non-linearity [6]. An example of an artificial neuron model (the *perceptron*, introduced already 1958 [27]) can be seen in Figure 2.2.

The depth of a network, meaning the number of layers of neurons between the input and output layer, depends on the function that is to be represented. Using multiple layers the same function can be represented as with a shallow net but with fewer neurons. The layers between input and output layers are called *hidden layers* [13]. The reason for calling them hidden is that the desired output of these layers is not given in the input data. An example of a neural network with one hidden layer is shown in Figure 2.3.

Neural networks where data is flowing in one direction from input to output are called feedforward neural networks [13]. In contrast, networks where layers

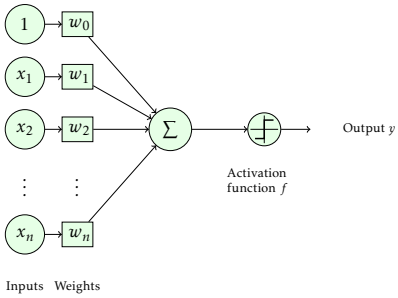


Figure 2.2: *Model of an artificial neuron. The inputs, x, are weighted with weights, w, and the summed signal is mapped through an activation function, f, to produce the output, y. The first input value is a bias term.*

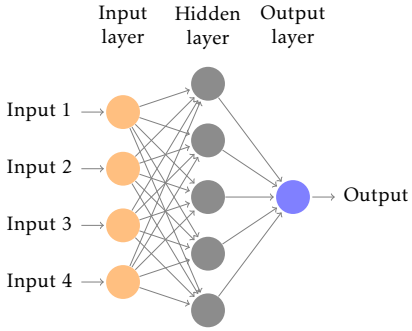


Figure 2.3: *A neural network with one hidden layer. The neurons are fully connected, meaning all neurons in one layer is connected to all neurons in the next layer.*

can be connected in a directed cycle are called recurrent networks. The key to finding a desired neural network is to determine the weights w that give the desired output [12]. One common method is described below.

Backpropagation

Backpropagation is a method in which the error obtained from the loss function propagates back through a network and adjusts the weights based on the contribution of each neuron to the total loss [12]. One method is to use *stochastic gradient descent* (SGD). In that case, the gradient of the error is used to update the weights iteratively after every training sample propagation. Using the chain rule in calculus, the derivatives result in different magnitude of change at different depths in the network. The weight update uses (2.1), where w^t is the current weight at iteration t, η is the learning rate and ϵ is the total loss.

$$w^{t+1} = w^t - \eta \frac{\partial \epsilon}{\partial w} \tag{2.1}$$

Deep learning

Deep learning aims among other things to solve problems that are intuitively easy for humans to solve [13], e.g. identifying a car in a picture. It is often hard for a human to formulate the solution to these problems by mathematical rules. By breaking down the mentioned identification into a hierarchy of concepts, connected in many layers at different abstraction levels, it is possible for a system to learn how to solve these problems. If a graph is drawn illustrating the connection between the layers, the graph is deep. This motivates the term *deep learning*. An example of a deep network can be seen in Figure 2.4. Some deep learning architectures, called *deep neural networks*, are based on neural networks, naturally with many hidden layers [13].

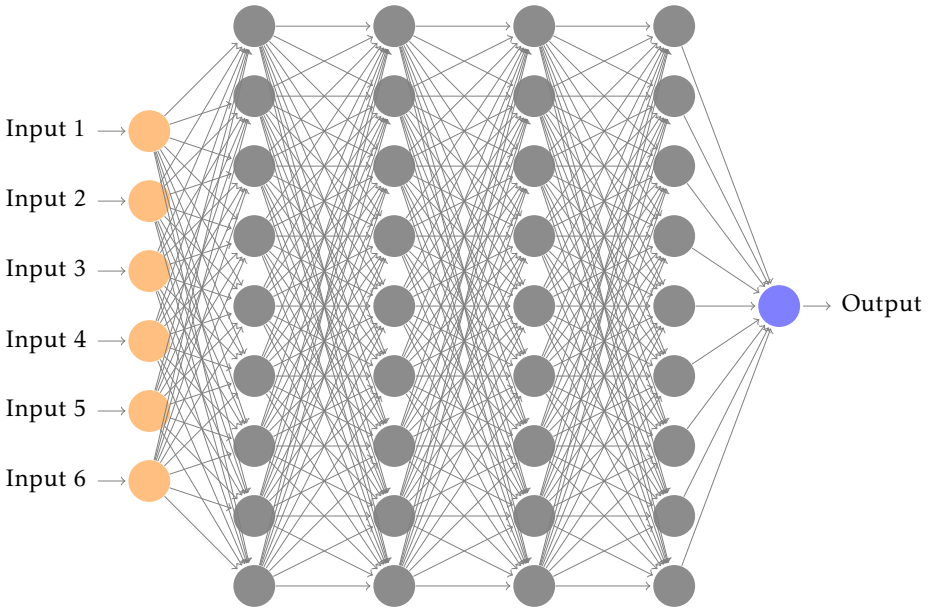


Figure 2.4: *A deep graph with many layers illustrating the principle of deep learning.*

Transfer learning

Transfer learning treats the ability of transferring knowledge of problem solving in one domain to another, related domain [13]. Previously obtained knowledge in

one application can be useful in other applications as well, since there are often similar relevant properties.

In the case of learning a deep neural network which consists of many layers, and if images are used as input to the network, it is desired to be able to find relevant properties in the input images by weighting the input with the learned layer neurons. The first layers often find general image features such as edges and corners. In this case, it might suffice to use a pre-trained network with given layers and only fine-tune the last layers when solving a new problem. This approach has been used with success in [3] where the effect of how much of a pre-trained network that is used (the number of layers) was investigated, along with how the similarity between a new problem and the original problem can be advantageous. It has also been shown that a generic CNN originally designed for classification tasks can be applied to regression tasks such as localization [35].

An example of transfer learning with a deep neural network is when an animal dataset for classification with some animal classes is extended with more animal classes. The first layers in the network can still be useful, but the last layers need to be changed, among other reasons in order to handle a different number of classes.

2.2.2 Representation learning

Representation learning is a term for systems that learn features themselves, opposed to manual feature engineering where features are found by using extensive knowledge of the data at hand. Below, the concept of convolutional neural networks which falls under representation learning are explained.

Convolutional Neural Network

A convolutional neural network, or a CNN, is a special kind of a neural network where the mathematical operation *convolution* is used in at least one of the layers [13]. A layer involving convolution is called a *convolution layer*. The input to the CNN is a 2D matrix, and in a convolution layer, convolution is carried out in a regular image processing manner. Any array data where nearby values are correlated, e.g. in 2D images, sound (reshaped to a 2D matrix in order to have correct dimension), video, etc., can be fed to a CNN. The convolution with the input array is calculated with kernels consisting of weights that are learned by backpropagation. The resulting 2D outputs obtained from all convolutions build a feature map layer. An illustrative image of a CNN architecture can be seen in Figure 2.5.

One type of layer is called *pooling layer*. The pooling layer downsamples the input by representing a local region (in a feature map) with a smaller size but still keeping important information, e.g. choosing the max value. The *stride* value is then used to determine the distance between the centers of the regions.

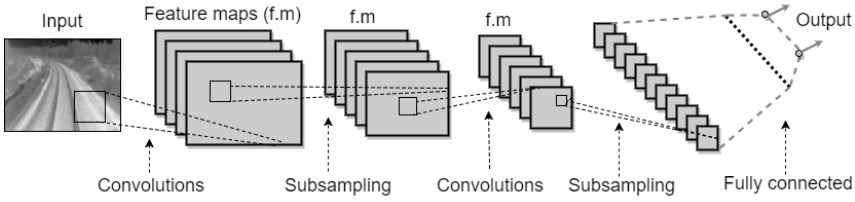


Figure 2.5: *A typical CNN architecture. The input image is convolved with different kernels, producing feature maps. The feature maps are subsampled by means of pooling and processed by a non-linear activation function. These steps are usually repeated a few times. Then the feature maps are further processed by a fully connected layer. In the end, the fully connected layer produces an output of appropriate size depending on application.*

The *dropout layer* is used to prevent over-fitting [32]. When a dropout is performed, a neuron in the net is prevented to participate further in the training, and its weight is locked. This happens with a certain probability. By doing this randomized preservation of weights, over-fitting is reduced in a simple and effective way.

Another common layer in CNNs uses a non-linear activation function, such as the rectifying linear unit (ReLU) [11] to adjust the values from a neuron. The reason for introducing a non-linearity is that a generic approximation requires non-linear behaviour, e.g. when solving the XOR problem [13]. The ReLU is defined as

$$g(z) = \max(0, z) \tag{2.2}$$

This activation function is recommended for the use with a majority of feedforward neural networks [13]. The ReLU is nearly linear since it is a piecewise linear function, therefore it preserves some of the properties that are advantageous with linear models when it comes to optimization with gradient based methods.

A commonly used heuristic to initialize weights in a CNN is

$$w \sim U\Big[-\frac{1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}}\Big] \tag{2.3}$$

where $U[-a, a]$ is the uniform distribution in the interval $(-a, a)$ and n_{in} is the number of inputs to a neuron from the previous layer [10]. Weights initialized with (2.3) have the variance

$$\mathrm{Var}(w) = \frac{1}{3n_{in}} \tag{2.4}$$

which changes with the number of inputs to a neuron. The variance of the back-propagated gradient when using only random weights like this might vanish or explode as deep networks are considered.

Xavier initialization is another way of initializing the weights in a neural network [10]. This method is preferable since it assigns weights in appropriate sizes (in many cases). Xavier initialization lets the gradient of the loss (e.g. when using SGD) backpropagate deep into the net during the early iterations by formulating conditions of the weight variances, forcing them to be the about the same size in all layers. The derived constraint of the weight variance is

$$\mathrm{Var}(w) = \frac{2}{n_{in} + n_{out}} \tag{2.5}$$

where n_{in} is the number of inputs and n_{out} is the number of outputs of a neuron. The proposed initialization of weights to retain the variance size throughout the net is the normalized initialization

$$w \sim U\Big[-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}} \Big] \tag{2.6}$$

The derivation of (2.6) is based on assuming that the activation functions used in the net are linear, which is not true for e.g. ReLU. Therefore, another initialization method has been proposed, which take into account the rectifier nonlinearities [15]. This method , called *MSRA initialization* in [16], has outperformed Xavier initialization when it comes to convergence on a very deep net with 30 layers (27 convolutional layers and 3 fully connected layers).

3

Data, representation, and labeling

This chapter starts with describing the dataset in Section 3.1. Later, the chosen representation is motivated and explained in Section 3.2. At last, in Section 3.3, the annotation procedure is covered.

3.1 Description of the dataset

The dataset used in this thesis consists of thermal infrared video sequences recorded from the front of a train [5]. The data was collected in northern Sweden in April 2015.

3.1.1 Data collection

The thermal camera is mounted as in Figure 3.1. Note that the thermal camera cannot be placed inside the train since the windshield is made of glass which is opaque to LWIR. The housing is designed to make use of aerodynamical effects in order to protect the (expensive) lens from damage caused by particles in the air.

3.1.2 Diversification

The data obtained was recorded during different times of the day. Four video sequences are used in the dataset. All frames in these sequences are labeled, except the frames containing multiple railways. Examples of frames from all

(a) *The train-mounted camera (encircled) with a smoother housing than in the previous system [5].*

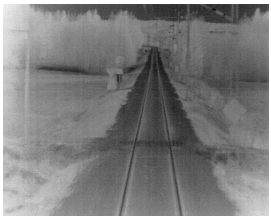
(b) *The train-mounted camera (encircled) in an outdoor environment.*

Figure 3.1: *Both (a) and (b) depicts the train-mounted thermal camera with a smooth housing, protecting the camera from damage caused by precipitation and/or particles in the air.*

(a) *Frame from sequence 1 (around 4:30 a.m.).*

(b) *Frame from sequence 2 (around 6:00 a.m.).*

(c) *Frame from sequence 3 (around 1:00 p.m.).*



(d) *Frame from sequence 4 (around 8:30 p.m.).*

Figure 3.2: *Frames from all sequences in the dataset.*

sequences in the dataset, visualized with a grayscale colormap, can be seen in Figure 3.2.

3.1.3 Pixel intensity interpretation

The intensity of the images is adjusted independently for each frame. That is, the dynamic range of the chosen colormap (in this thesis a grayscale colormap is used in the dataset images) is adapted to the minimum and maximum radiation measured in each frame. The images can therefore appear quite different from one another even within the same video sequence, if the conditions are ever-changing. The pixel values before rescaling represent apparent temperature.

This intensity rescaling is done in order to visualize the depicted scene in a more appealing way. Also, since the rail is clearly visible for a human in the rescaled images, the CNN is trained with these images, as it is interesting to see if the CNN can perceive the rail with the same image information as a human.

3.2 Representation

In this part, the chosen representation is described and motivated in depth.

3.2.1 Feature selection

One possible representation of the rail is a binary mask over the whole image. Rail detection and segmentation could be performed as in [38] where they detect and localize faces in images simultaneously with a CNN. Since railway switches are not treated in this thesis and only one railway track will be estimated in each frame, this representation could be problematic. The rail is always connected, and this representation might result in a rail estimate that is split into separate pieces all over the image, making it hard to determine how to parse the representation into a position estimate of only one railway. Another disadvantage with this representation is its high dimension (> 32000D).

In view of the background explained in Section 1.1 and the above mentioned inappropriateness, it is reasonable to represent the rail with *keypoints* along the rails, placed at specific rows. This representation has similarities with the facial keypoint problem [21, 33]. Since the keypoints are placed at specific rows (y-coordinates), all that is needed to be determined in each frame are the corresponding x-coordinates, reducing the size of the feature vector significantly. The keypoints are chosen as the midpoints of the rail (the points of the railway equidistant to the left and right rail).

Inspection of the dataset reveals that even though the rail is not in clear sight, it is possible to see where the rail is located anyway, judging from the surroundings. Therefore, rail midpoints are annotated despite the rail being occluded by bushes, pillars, trees, hills etc.

Since the width of the rail naturally is of interest where it is visible in the image, it is part of the representation as well, for the corresponding midpoints (not all midpoints have visible widths). Thus, the representation ends up as follows:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ x_{N+1} \\ x_{N+2} \\ \vdots \\ x_{N+M} \\ w_{N+1} \\ w_{N+2} \\ \vdots \\ w_{N+M} \end{bmatrix} \tag{3.1}$$

where $x_i, i = 1, 2, ..., N + M$ denotes the horizontal image coordinates of rail midpoints, $w_j, j = N + 1, N + 2, ..., N + M$ denotes rail widths in whole pixels for M midpoints with visible widths, and N and M are positive integers. Bold letters denote vectors.

3.2.2 Normalization and image coordinate system

All features in the representation are normalized by division with the image width (in pixels). This is preferable because it is easier to work with data of different spatial size. Moreover, since transfer learning is used with weights from a net with output in the same range ([0,1]), it is easier to adapt the old weights to this particular problem.

The image coordinate system is defined with $(x, y) = (0, 0)$ in the upper-left corner.

3.2.3 Rail estimation area

Due to camera positioning and environmental variations the rail will not be visible at all rows in an image. The row with the lowest index (indexed from the top of the image) at which the rail can be located is denoted as $y_{\min}$. The variation of $y_{\min}$ between frames can e.g. depend on whether the train is in the middle of a sharp curve or whether the ground is tilted. Just some degrees of tilt changes $y_{\min}$ notably. This is troublesome since it is desirable to label the entire visible rail in each image, and at the same time only estimate the horizontal coordinates of the rail.

In order to handle this problem it was decided to only estimate keypoints in a specific interval of y-coordinates for all frames. This interval is limited by the image height and $y_{\min}$ averaged over a subset of images in the dataset.

3.3 Labeling

Labeled data was produced manually. Here, the labeling procedure is explained in detail.

3.3.1 Labeling accuracy increase

In the image rows where the left and right rails are distinguishable, it is useful to mark them separately during labeling and then extract the midpoint afterwards. The reason is because it is easier to perceive the left and right rails' exact locations, in contrast to the midpoint, when the resolution is high (which is the case in the image rows closest to the bottom of the image).

3.3.2 Vertical keypoint placement

In order to determine which image rows to use, i.e. the vertical placement of the keypoints, the dataset was inspected thoroughly, bearing in mind that the region of most importance to represent is the part where the rails are inseparable, since this is the region where the old system had problems to find the rail [5]. Obviously, the keypoints should lie denser in areas further away in 3D world coordinates (corresponding to lower y-coordinates).

By testing different number of points and inspecting the points overlaid on the rail in the dataset, it was concluded that a feasible number of points is eight points. Four of the points are marked more accurately by marking the left and right rail when annotating (and later extract the corresponding midpoints). It was concluded that more than eight points would increase the size of the representation at a high cost in relation to what the extra points would contribute with in terms of increased accuracy of the rail approximation.

The chosen number of keypoints are placed in the y-direction according to a rule which changes the relative distance between keypoints in the mentioned direction. When the lowest y-coordinate is determined, the available interval spans from that coordinate to the height of the image. This available interval is split among the points that are to be placed on the left and right rail (points pairs) and the midpoints. The first pair of points in the former category are placed in the bottom-most row of the image. Then the next pair of points are placed in the middle of the currently available interval. The available interval is split in two as long as there are point pairs left to insert, and the procedure is repeated accordingly. When there are no more point pairs left, the remaining row interval is split equally among the midpoints. The interval splitting procedure is illustrated in Figure 3.3. This procedure is used because it is easy to adapt to images of different spatial size (if using another dataset), it is simple to implement, and it takes into consideration the varying importance of keypoints by changing the relative distance between the keypoints.

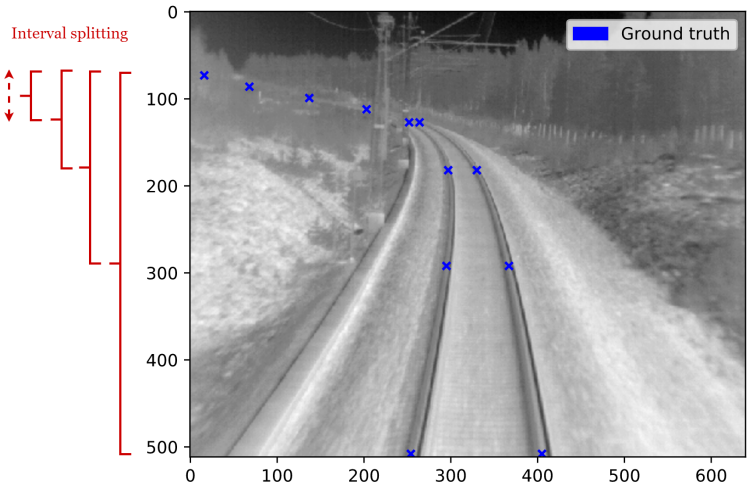


Figure 3.3: *Placement of keypoints along the rails in the chosen rail estimation area (y-coordinate interval). The interval is split by half with decreasing y-coordinates among the point pairs. The smallest interval is split to place the midpoints equidistant to each other (in the area marked with a dotted arrow).*

With $y_{\text{min}} = 73$ and using the placement rule mentioned above, the y-coordinates ended up as follows:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N+M} \end{bmatrix} = \begin{bmatrix} 73 \\ 86 \\ 99 \\ 112 \\ 127 \\ 182 \\ 292 \\ 512 \end{bmatrix} \tag{3.2}$$

with $N = 4$ and $M = 4$.

3.3.3 Keypoint usability

Since the rail estimation area is preset (see Section 3.2.3 and Section 3.3.2), there are occasions when a rail physically cannot be located in some of the preset rows (within the image). The corresponding keypoints will receive a value which has no meaning. In order to prohibit these undefined keypoints from ruining the

training of the CNN, a binary *usability vector* is used in combination with the representation vector. The undefined points will be assigned the usability value 0, whereas the valid keypoints will be assigned the value 1. This binary mask vector will then be used as supplementary input to the CNN, and the mask removes the undefined points from the optimization procedure when calculating the loss. The usability vector is defined as:

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N+M} \end{bmatrix} \tag{3.3}$$

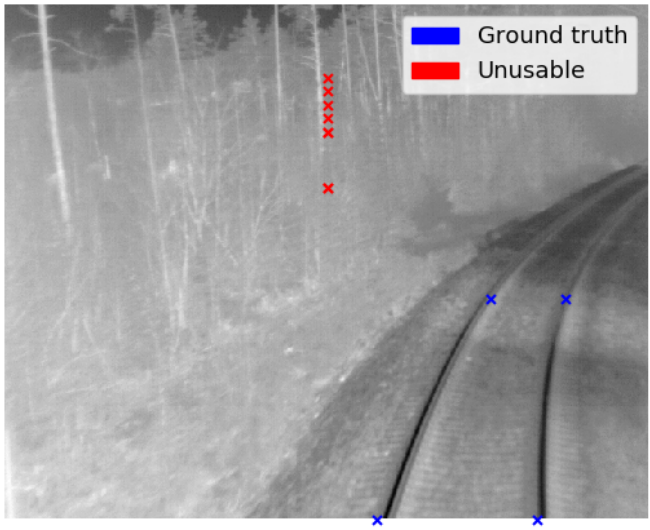
where u_i, $i = 1, 2, ..., N+M$ denotes the binary usability of the rail midpoints.

An example of images with undefined keypoints can be seen in Figure 3.4.

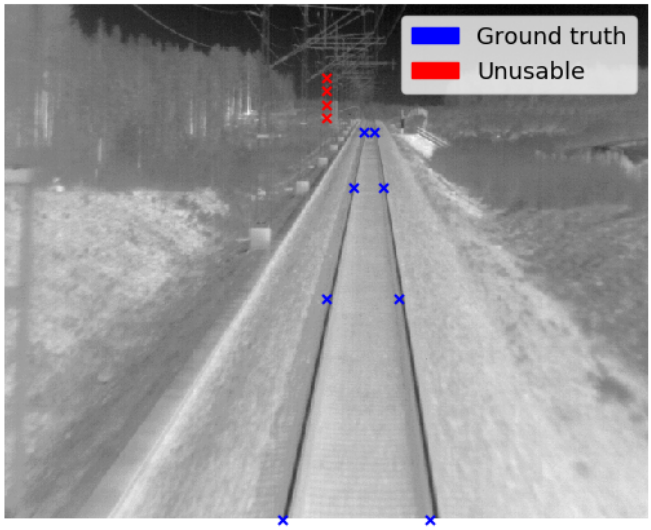
3.3.4 Augmentation

In order to extend the size of the dataset without needing to annotate more frames manually, data augmentation is used. All images (and their corresponding labels) are mirrored horizontally. One advantage with this augmentation is that the amount of left and right curves is made even.

Another augmentation technique used is adding noise to keypoints with low row index. This is useful since it is difficult to mark the rail location exactly (especially when the rail is occluded). A variance of a few pixels pertains in the labeled data, therefore the dataset can be extended with even more labeled frames with small adjustments of the keypoints horizontal location.



(a) Unusable keypoints because of a sharp curve.



(b) Unusable keypoints because of slightly tilted railway.

Figure 3.4: *In (a), the six topmost keypoints cannot be placed on the rails within the image. Therefore, they are marked as unusable. The problem in this particular image is the sharp curve. In (b), the ground is slightly tilted in vertical direction, enough to move the camera to a position where the topmost four keypoints are located above the ground. Hence, the keypoints are unusable.*

4

Network architecture

In this chapter, the design of the CNN is described, as well as the loss function which is tailored for this application.

4.1 CNN architecture

The used net, henceforth called *RailNet*, is a slightly modified version of *CaffeNet* [19] which is based on *AlexNet* [20]. CaffeNet was chosen as the mainstay net architecture, among other reasons because it is included in the Caffe framework which was used for training and because CaffeNet has been used with success classifying images in the large database *ImageNet* [19]. The entire net is visualized in Figure 4.1. Using an existing net is preferable in this thesis work which main purpose is not to design a new complex net.

RailNet is in some of the performed tests in Chapter 5 either initiated or fine-tuned with weights from a training performed with CaffeNet [7]. These pre-trained weights, *CaffeNet weights*, are in these tests used for all layers except the fully connected layers. The weights trained from scratch are in all layers initialized with Xavier filling [10].

The RailNet architecture consists of pooling layers, convolution layers, ReLU layers, normalization layers, drop-out layers, and fully connected layers. A complete specification of the parameters in each layer as well as training parameter choices is found in Chapter 5.

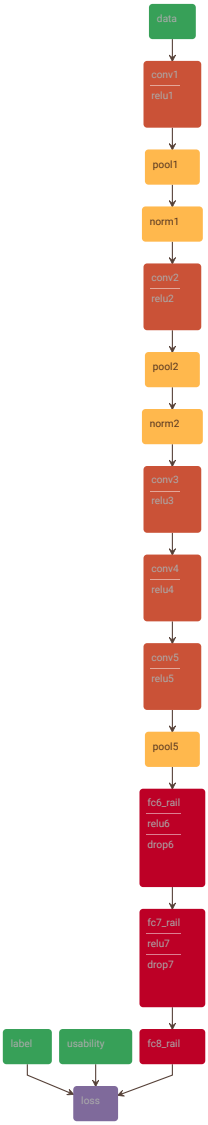


Figure 4.1: *The CNN net architecture. The only differences from CaffeNet is the number of outputs from the last fully connected layer and the usability data which is input to a new loss layer. RailNet consists of pooling layers (pool), convolution layers (conv), ReLU layers (relu), normalization layers (norm), drop-out layers (drop), and fully connected layers (fc). The processed data are input, label and usability data. The usability data is used to filter out the unusable points when calculating the loss, see Section 3.3.3.*

4.1.1 Fully connected layers

The fully connected layers have the same architecture as in CaffeNet but are trained from scratch. The number of outputs from the last fully connected layer is the same as the number of features in the representation, which in this thesis is 12.

4.1.2 Transfer learning by fine-tuning

Before the fine-tuning of CaffeNet weights is performed, the fully connected layers are trained with the pre-trained weights for the convolutional layers as *frozen* [37] (i.e. the learning rate is zero). This is done since it is desirable to train the fully connected layer weights for a few *epochs* (after one epoch all images are used one time) in order to not use too large gradients in the early backpropagations [2].

When the fine-tuning takes place, the n first pre-trained layers are still frozen, while the others are updated. The reason to keep the n first layers frozen throughout the whole training is that the first layers can be considered as *general* since they extract general features such as lines and edges [3, 37]. It has also been shown for many varying problems and datasets that when training the first layers from scratch, the first layers tend to end up with kernels similar to Gabor filters and color blobs [37]. Therefore, it is generally advantageous to leave the first layers unchanged.

4.2 Loss function

The loss function to be minimized is tailored to the particular problem at hand. The desired output space of possible rail keypoints are constrained in many senses. Some of these constraints are therefore used in the loss function, in order to penalize bad estimates in an appropriate way. The loss function is split into three terms that each have a different weight in order to produce a good overall penalization. The loss function is defined as:

$$L = a \cdot L_{\text{wpl}} + b \cdot L_{\text{wwl}} + c \cdot L_{\nabla} \tag{4.1}$$

where L_{wpl} is weighted Euclidean point loss, L_{wwl} is weighted Euclidean width loss, L_{∇} is gradient loss, and a, b, and c are real-valued constants. Each loss term is described in more detail below.

The usability vector defined in Section 3.3.3 is one of the inputs to the loss layer. The usability vector is used to remove the unusable points when calculating the loss.

4.2.1 Weighted Euclidean point loss

Euclidean errors in keypoints corresponding to rail locations far away from the camera in world coordinates are more important in the optimization process. An Euclidean error of a few pixels in this area correspond to a larger deviation in world coordinates, than the same Euclidean error in pixels closer to the camera (and the train). Therefore, the Euclidean error for each keypoint is weighted with a factor reflecting its relative importance.

The first term in the proposed loss function, the weighted Euclidian point loss, L_{wpl}, is defined as:

$$L_{\text{wpl}} = \sum_{k=1}^{K} h(k) \|\hat{x}_k - x_k\|_2^2 \tag{4.2}$$

where the weighting function $h(k) = \frac{1}{k}$, $\hat{x}_k$ denotes the estimated horizontal image coordinates of rail midpoints, x_k denotes the corresponding labels, and K is the number of usable rail midpoints (at most $N + M$).

The weighting function $h(k)$ used, depicted in Figure 4.2, is inversely proportional to the index of keypoints in the representation. In this way keypoints closer to the rail estimation limit y_{min} affect the loss more. The first four keypoints (counted from y_{min}) cover 75% of the used weight interval $(0, 1]$ which is good considering they usually do not have visible widths in the images, making it even more crucial to find good estimates. Therefore, it is reasonable that these keypoints have considerably larger weights. If using a different method of placing keypoints in the vertical direction, the system might benefit from another weighting function.

4.2.2 Weighted Euclidean width loss

For the same reason as in the Euclidean point loss, the width errors are weighted (with the same weighting function).

$$L_{\text{wwl}} = \sum_{m=1}^{M} h(m) \|\hat{w}_m - w_m\|_2^2 \tag{4.3}$$

where $h(m) = \frac{1}{m}$, $\hat{w}_n$ denotes the estimated rail widths, w_n denotes the corresponding labels, and M is the number of rail midpoints with corresponding rail widths.

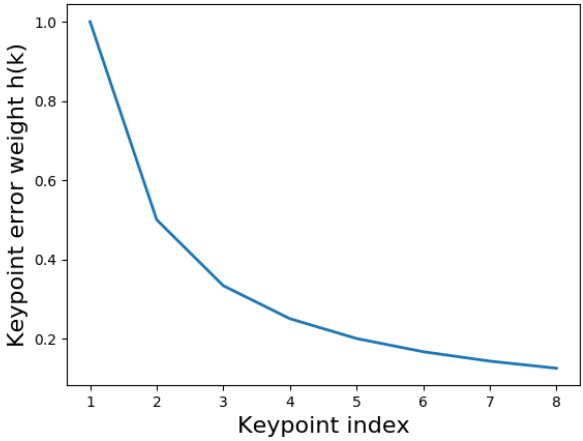


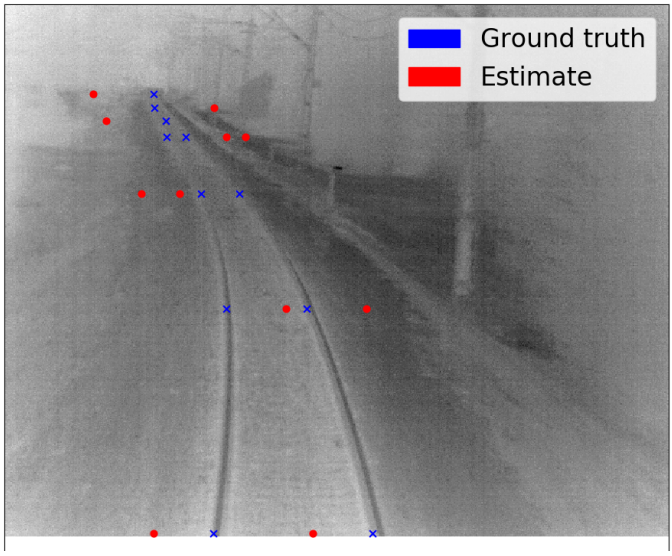
Figure 4.2: *The weighting function $h(k)$ used in L_{wpl} in order to induce the relative importance of keypoints. Keypoints with higher index are closer to the bottom of the image. Their Euclidean error are not as important as the Euclidean error of keypoints with lower index.*

4.2.3 Gradient loss

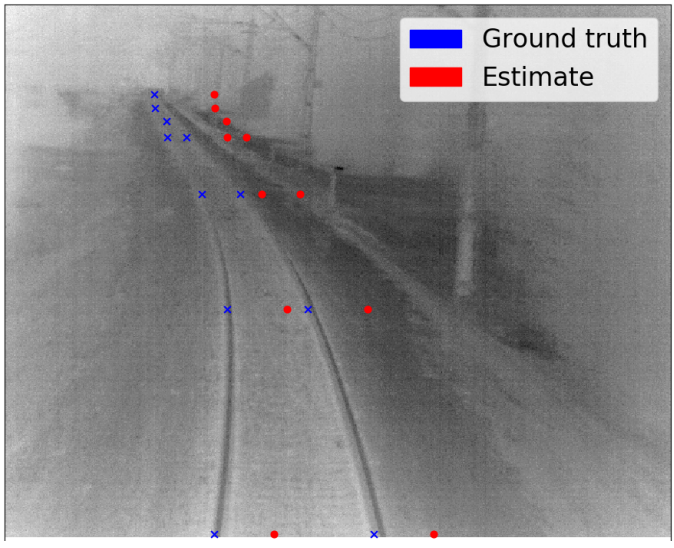
There are many physical constraints of the rail. The rail is smooth and with no sudden direction changes. In order to penalize zigzag estimates which are physically impossible, an approximate derivative of the relative error for the point loss is used in the loss function. An example of a zigzag estimate can be seen in Figure 4.3a. A rail estimate with a zero point loss gradient has the correct shape, but all keypoints could be translated in the same horizontal direction in the image. An example of such an estimate can be found in Figure 4.3b.

$$L_{\nabla} = \sum_{k=1}^{K-1} \|\hat{x}_k - x_{k+1}\|_2^2 \tag{4.4}$$

where $\hat{x}_k$ denotes the estimated horizontal image coordinates of rail midpoints, x_k denotes the corresponding labels, and K is the number of usable rail midpoints (at most $N + M$).



(a) Zigzag estimate.



(b) Horizontally translated but correctly shaped estimate.

Figure 4.3: *In (a), the gradient error for the Euclidean point loss is large since the relative error is large due to a zigzag estimate. In (b), the gradient error is zero since the overall shape of the rail is correct, but all keypoints are translated equally much in the horizontal direction. Note that the Euclidean width loss is zero too. The translation error is instead penalized in the Euclidean point loss.*

4.2.4 Matrix formulation

The loss function can suitably be formulated with matrices as follows:

$$L = \|W\tilde{\mathbf{x}}\|^2 + \|A\tilde{\mathbf{x}}\|^2 = \tilde{\mathbf{x}}^T W^T W\tilde{\mathbf{x}} + \tilde{\mathbf{x}}^T A^T A\tilde{\mathbf{x}} = \tilde{\mathbf{x}}^T (W^T W + A^T A)\tilde{\mathbf{x}} \tag{4.5}$$

where

$$W = \begin{pmatrix} \sqrt{h_1} & 0 & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & \sqrt{h_2} & 0 & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & 0 & \ddots & 0 & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & 0 & \sqrt{h_K} & 0 & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & 0 & \sqrt{h_1} & 0 & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & 0 & \sqrt{h_2} & 0 & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 & \ddots & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & 0 & \sqrt{h_M} \end{pmatrix} \begin{pmatrix} \sqrt{a} & 0 & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & \ddots & 0 & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & 0 & \ddots & 0 & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & 0 & \sqrt{a} & 0 & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & 0 & \sqrt{b} & 0 & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & 0 & \ddots & 0 & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 & \ddots & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & 0 & \sqrt{b} \end{pmatrix}$$

where the dimension of the weight matrix W is $(K + M) \times (K + M)$ and

$$A = c \begin{pmatrix} 1 & -1 & 0 & \dots & \dots & \dots & \dots & \dots & \dots & 0 \\ 0 & \ddots & \ddots & 0 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & 0 & \ddots & \ddots & 0 & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & 0 & 1 & -1 & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & 0 & 0 & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & 0 \end{pmatrix}$$

where the dimension of the derivative matrix A is $[(K-1) + (M-1)] \times (K + M)$ and

$$\tilde{\mathbf{x}} = \hat{\mathbf{x}} - \mathbf{x}$$

where bold letters denote vectors. If a keypoint is not usable, all rows in A involving that keypoint are zeroed.

The derivative of L w.r.t. $\hat{\mathbf{x}}$ is used in the backpropagation (see Section 2.2.1):

$$D = \frac{\partial L}{\partial \hat{\mathbf{x}}} = 2(W^T W + A^T A)\tilde{\mathbf{x}} \tag{4.6}$$

5
Experiments and results

This chapter describes the tests that were performed and what results that were acquired. The questions arisen from the problem description of this thesis are answered below. The questions are as follows:

- Is it possible to find a better estimate of the rail in the area where only the general shape of the rail is visible compared to in the previous system? See Section 5.2.4.
- Is the rail detector usable in real-time? See Section 5.2.5.
- Is it advantageous to use transfer learning in order to generalize better? See Section 5.2.2.

The experiments performed, described, and evaluated below are as follows:

- Fine-tuning the net with pre-trained weights, see Section 5.1.7.
- Training the net from scratch, see Section 5.1.8.
- Varying the loss function, see Section 5.2.3.

5.1 Training approaches

Eight different approaches were used when training RailNet, each one resulting in a different rail detector. The approaches are different combinations of the following factors:

- Hyperparameters
- Early stopping
- Noise augmentation labels
- Image mean subtraction (used in all approaches)
- Horizontally centered origo
- Loss term weighting (same weighting in all approaches)
- Fine-tuning
- Training networks from scratch

The approaches are as follows:

1. Fine-tuning and using early stopping.
2. Fine-tuning, using early stopping, and using noise augmentation labels.
3. Fine-tuning and using centered origo.
4. Fine-tuning, using centered origo, and using noise augmentation labels.
5. Initializing network with pre-trained weights and using centered origo.
6. Training network from scratch and using noise augmentation labels.
7. Training network from scratch and using more noise augmentation labels.
8. Training network from scratch and using centered origo.

The approaches and the evaluations of them are later specified in Table 5.5, and the factors are described below.

5.1.1 Hyperparameters

The hyperparameters in the RailNet architecture can be found in Table 5.1. The batch size (i.e. the number of network outputs averaged when calculating the loss) was set to 60 due to memory limitations of the GPU. This means that one iteration in the training process corresponds to forward-backward passes of 60 images. Thus, one epoch is passed after $\frac{8910}{60} = 148.5$ iterations. The number of iterations/epochs chosen is presented and motivated in Section 5.1.7.

Learning rate and optimization solver

The learning rate α when using SGD has to be chosen manually. Initial tests were done using SGD and the best performance was obtained with $\alpha = 0.00001$. Higher values of α resulted in NaN loss after a few iterations. However, the convergence for the validation loss was considered too slow when using SGD.

Table 5.1: *The hyperparameters of the CNN architecture. The output sizes are given in the format* $channel \times width \times height$*. The network takes images of size* $3 \times 640 \times 512$ *as input. The dropout layers and normalization layers not mentioned in the table uses the default values from CaffeNet [7].*

Type	Output size	Kernel size	Stride
conv1	96x126x158	11	4
pool1 (max)	96x63x79	3	2
conv2	256x63x79	5	1
pool2 (max)	256x31x39	3	2
conv3	384x31x39	3	-
conv4	384x31x39	3	-
conv5	256x31x39	3	-
pool5 (max)	256x15x19	3	2

In addition to SGD, *AdaDelta* [39] was used. AdaDelta is an adaptive learning rate method that uses first order information to adjust the learning rate over time, with only a slightly longer computation time than SGD [39]. The choice of an adaptive learning rate algorithm is motivated [29] by the fact that the size of the training set for the target task is much smaller (thousands of images) than the dataset in the reference task, where the dataset is ImageNet [9] (millions of images). The AdaDelta method still has some hyperparameters that have to be chosen. The parameters used were the conditioning constant $\epsilon = 10^{-8}$ and decay constant $\mu = 0.95$ (similar to the momentum term in SGD). All networks that are part of the evaluation were trained with AdaDelta.

5.1.2 Early stopping

It was considered that overfitting had occurred when the validation loss did not decrease (but training loss might have continued to decrease). When the validation loss ceased to decrease, training was discontinued, although with one exception. When it was desired to compare the currently trained system with another system after the same number of iterations, training was continued.

5.1.3 Noise augmentation labels

The original dataset was augmented with Gaussian noise with zero mean and a variance of 3.84 pixels added to the four first keypoints (counted from $y_{\min}$). The variance was estimated by inspecting the augmented labels on top of the corresponding images of some frames with different choices of variance and judging

Table 5.2: The weighting of the loss terms.

Loss term	Weight	Value
L_{wpl}	a	540
L_{wwl}	b	60
L_{∇}	c	20

the labels physical plausibility.

5.1.4 Image mean subtraction

The mean of the training dataset was subtracted from all images during training. This is recommended when using CaffeNet [8].

5.1.5 Horizontally centered origo

The network outputs corresponding to rail midpoint coordinates can be interpreted as offsets from the horizontal center of the image, instead of offsets from the left edge of the image. By changing the interpretation like this, the convergence time of the test loss was significantly reduced, see Table 5.5.

5.1.6 Loss term weighting

The weighting of the loss terms can be seen in Table 5.2. The weights were determined without training, by brute force testing different values and using labels with noise of various degree as estimates. By calculating the loss for these estimates and inspecting them, it was possible to make a decision of what is reasonable to penalize when it comes to the relation between gradient, point, and width loss.

5.1.7 Fine-tuning

The official pre-trained weights to CaffeNet that come with the official distribution are trained with 3-channel (RGB) images. In order to use these weights, the input data has to be 3-channel images as well. Therefore, the single channel thermal images were triplicated to fill up all channels.

The evaluated networks using fine-tuning are called RailNetTransfer1 and RailNetTransfer2, and the training protocol for each fine-tuned network is explained in the remark column of Table 5.5. Also, pre-trained weights were used for weight initialization only in one of the networks, RailNetTransfer3.

Table 5.3: *Summary of fine-tuning with phases. The original and mirrored (Augmented 1) datasets contain 8910 images in total. Augmented dataset 2 and 3 contain 8910 images respectively, using the same images but with noise augmentation labels. The noise in the augmented datasets is independently generated, and two datasets were used instead of one in order to extend the training data even more.*

Training phase	# of frozen convolution layers	Dataset	# of iterations
1	5	Original + Augmented 1	500
2	2	Original + Augmented 1	6500
3	2	Augmented 2	5000
4	2	Augmented 3	7000

Fine-tuning with phases

Fine-tuning with phases is based on the fine-tuning outline mentioned in Section 4.1.2, and is summarized in Table 5.3. The reason for splitting up the training datasets and train in phases is that the input images and corresponding labels were converted to a memory mapped database (in order to speed up input data fetching). The conversion of images was time consuming, and since only the labels were changed in the noise augmentation process, the existing converted images could be used. The decision to use noise augmentation was made after half the thesis work.

As can be seen in Table 5.3, the number of iterations varies with each phase. The first phase is about 3.4 epochs, i.e. just a few epochs as proposed in [2]. The other iteration intervals were determined by taking early stopping into consideration when training RailNetTransfer1. The same iterations were then used for the other networks as well, except RailNetTransfer2 which was trained for 1000 iterations during phase one since it was desired to investigate a longer training with all convolution layers frozen.

5.1.8 Training networks from scratch

The training performed from scratch was also performed with triplicated single channel images. The reason is because the need of converting the input images to a memory mapped database one more time (time-consuming) was eliminated. Also, the datasets used were sufficiently small to keep the training time reasonable even though 3-channel images were processed. The weights trained from scratch are in all layers initialized with Xavier filling [10].

The evaluated networks trained from scratch are called RailNetScratch1 and RailNetScratch2. The training protocol for each network is explained in the remark column of Table 5.5.

Dataset phases

The training procedure with dataset phases is summarized in Table 5.4.

Table 5.4: Training procedure when training from scratch.

Training phase	Dataset	# of iterations
1	Original + Augmented 1	7000
2	Augmented 2	5000
3	Augmented 3	7000

5.2 Evaluation

In this section, the evaluation of the implemented rail detectors is presented along with the old system. In Section 5.2.1 the used performance measure is motivated.

All experiments were conducted on a x64-based PC with 4.20 GHz Intel(R) Core(TM) i7-7700 CPU, NVIDIA GeForce GTX 1080 GPU, and 16.00 GB RAM memory. For training of the CNN, Caffe [16] was used. The custom loss layer was written in Python [28] (in the Caffe Python interface) using NumPy [36].

5.2.1 Performance measure

The tailored loss function is used as the performance measure. The loss function is constructed for one thing only, to evaluate whether its input is a good rail estimate or not. Therefore, it is appropriate to use it for evaluation as well.

Another motivation for using the loss function for comparison with the previous system, is that the region of most importance is the area furthest away from the train. If the keypoints would have been used as control points for a spline and the overlap between the embedded area with a corresponding ground truth area would have been used as performance measure, the relative distance importance (of keypoint estimations) would have been more difficult to incorporate in the evaluation.

5.2.2 Network comparison

The performance of all evaluated systems can be seen in Table 5.5. A complement to the test loss is the loss in a labeled random sequence. The random sequence

Table 5.5: *Performance of all evaluated systems including the old system. The random sequence consists of 2800 consecutive frames that are recorded in another area than the train/validation/test sets. Each loss are the average loss over the used dataset.*

System	# of iterations	# of epochs	Remark	Test loss	Random sequence loss
RailNetTransfer1	7000	47.1	2 phases	0.0186	3.841
RailNetTransfer1	19000	127.9	4 phases	**0.0045**	1.226
RailNetTransfer2	7500	50.5	Centered origo, 2 phases	0.0074	1.107
RailNetTransfer2	11000	74.1	Centered origo, 3 phases	0.0064	**1.031**
RailNetTransfer3	7000	47.1	Centered origo, pre-trained weight initialization	0.0083	1.804
RailNetScratch1	12000	80.8	3 phases	0.0087	5.556
RailNetScratch1	19000	127.9	4 phases	0.0107	4.992
RailNetScratch2	7000	47.1	Centered origo	0.0084	3.634
Old system				-	4.360

consists of 2800 consecutive frames that are recorded in another area than the train/validation/test sets.

From Table 5.5 it can be concluded that the best rail detector on the random sequence is RailNetTransfer2 after 11000 iterations. The best rail detector on the test set among the nets is RailNetTransfer1 after 19000 iterations. Examples of estimates from RailNetTransfer2 on the test set can be seen in Figure 5.1-5.3.

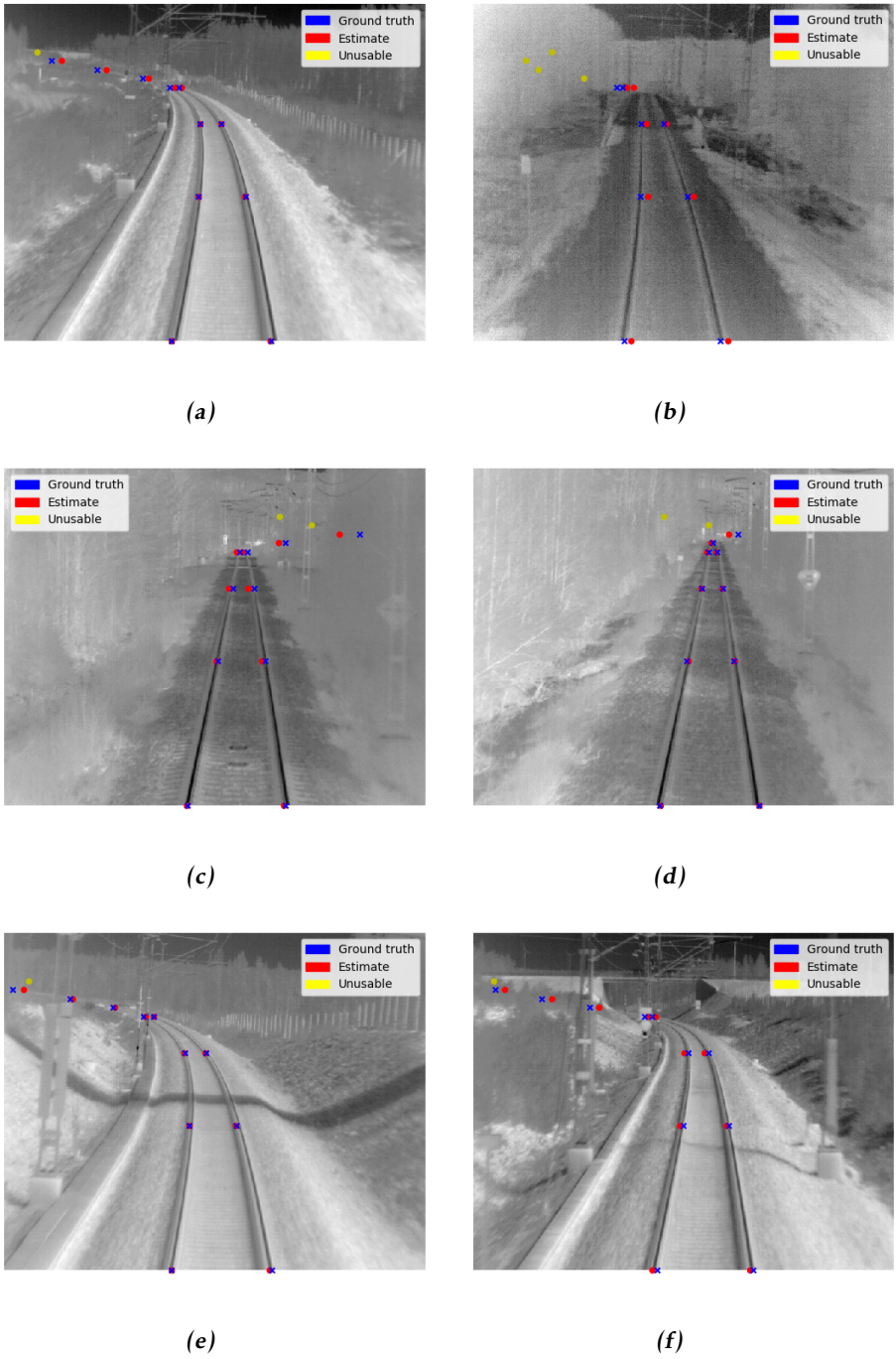
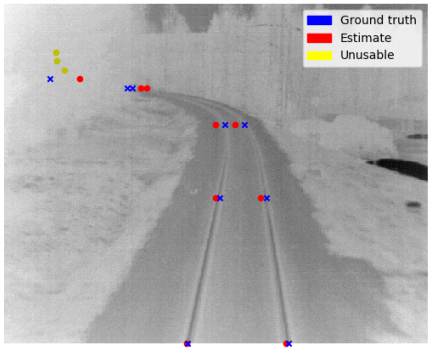
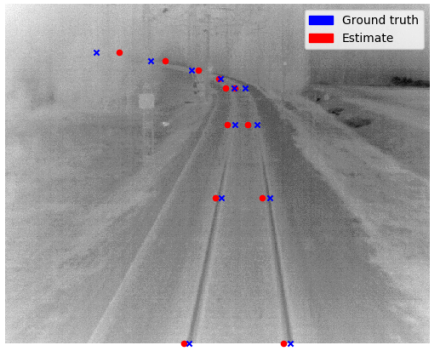


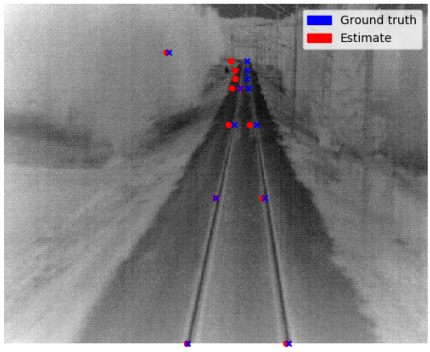
Figure 5.1: *Examples of good estimates from RailNetTransfer2 (11000 iterations) on the test set. In all images one or several estimates from the top have no corresponding ground truth.*



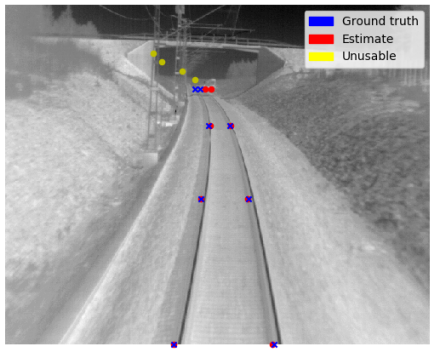
(a)



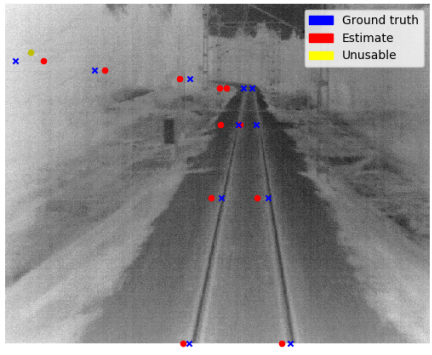
(b)



(c)

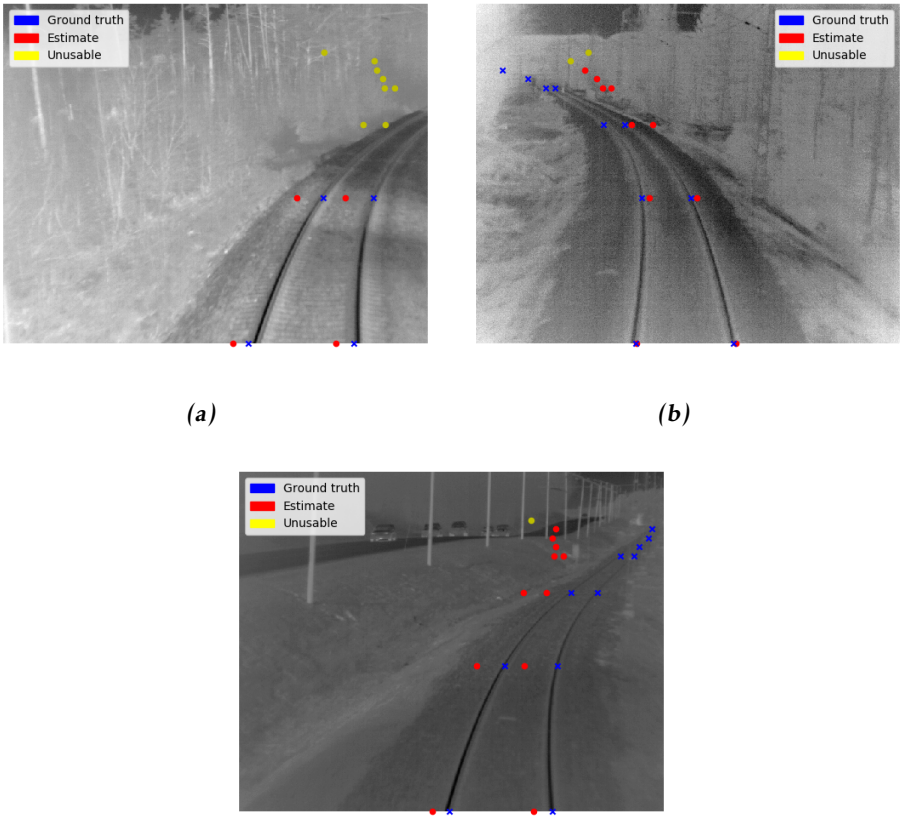


(d)



(e)

Figure 5.2: *Examples of decent estimates from RailNetTransfer2 (11000 iterations) on the test set. In some images one or several estimates from the top have no corresponding ground truth.*



(a) *(b)*

(c)

Figure 5.3: *Examples of bad estimates from RailNetTransfer2 (11000 iterations) on the test set. In all images one or several estimates from the top have no corresponding ground truth.*

5.2.3 No gradient loss

In order to analyze the contribution of using the gradient of the point error to make more physically plausible estimates, RailNetTransfer2 was trained for 50.5 epochs without the gradient term in the loss function. The weights of the remaining terms were unchanged. Since the scale of the loss function with the gradient term removed is changed, the obtained loss is not directly comparable to the loss received with all terms. However, it is possible to see if the gradient error is minimized by penalizing it in the training, and see if the loss in the complete loss function (all three terms) is smaller even when penalizing the gradient. The result of training the networks with and without the gradient term can be seen in Figure 5.4.

In Figure 5.4a the gradient loss has increased and the total loss is bigger, opposite to Figure 5.4b. In Figure 5.4c both the gradient loss and the total loss is smaller, opposite to Figure 5.4d. In Figure 5.4e the gradient loss is bigger but the total loss is smaller, compared to Figure 5.4f where the gradient loss is smaller but the total loss is bigger. The comparison between Figure 5.4a-5.4b is as desired; the detector trained with no gradient loss have a higher gradient loss and a higher total loss. The two other comparisons show that the desired effect of the gradient term is not achieved in all cases.

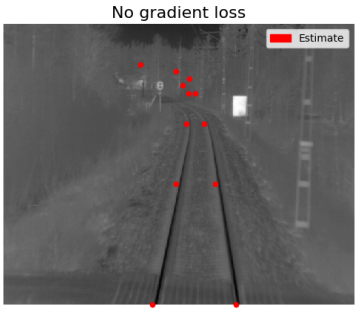
5.2.4 Comparison with old system

In order to be able to compare the new system to the previous system, the estimates from the previous system were converted to keypoint estimations. This was done by extracting the endpoints of the binary rail mask in horizontal direction at the preset rows as left and right keypoints, and use them to calculate the midpoints (and the widths when applicable). An example of keypoint extraction can be seen in Figure 5.5.

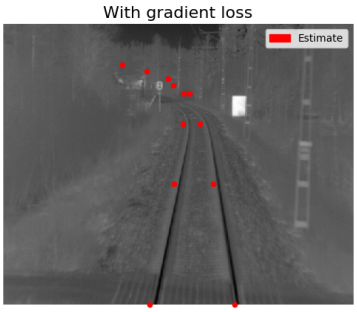
The previous system does not find estimates at all preset rows up to y_{min} in every frame. In the cases when it does not find any estimate at a specific row, the keypoint estimate is set to half the image width, considered as the least biased way since there are a lot of curves in the random sequence.

Examples of estimates on the random sequence set from both RailNetTransfer2 (after 11000 iterations) and the previous system can be seen in Figure 5.6-5.8.

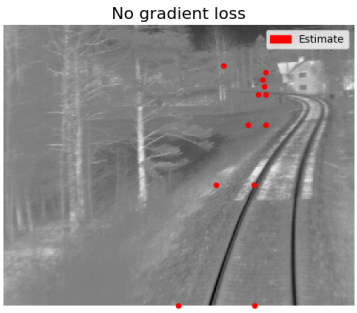
As shown in Table 5.5, several of the RailNets, and in particular all the RailNets using transfer learning, perform better than the previous system on the random sequence.



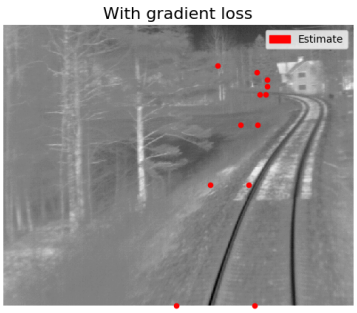
(a) L_∇: 0.689 L: 0.899



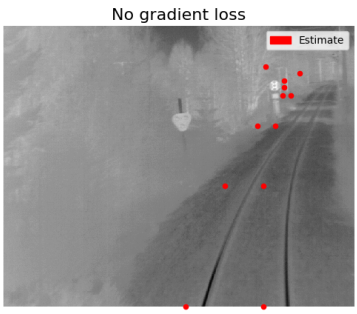
(b) L_∇: 0.638 L: 0.815



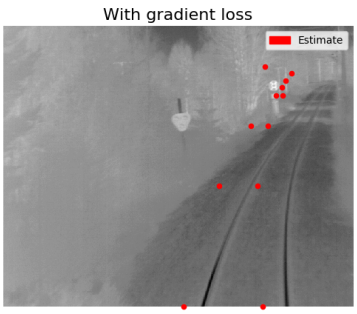
(c) L_∇: 1.855 L: 2.477



(d) L_∇: 2.055 L: 2.767



(e) L_∇: 0.047 L: 0.465



(f) L_∇: 0.038 L: 0.499

Figure 5.4: *Estimates on random frames not part of the training/validation/test sets. The estimates in the left images are from a system trained with centered origo (for 7000 iterations with the two first fine-tuning phases) but without the gradient term in the loss function. The same training procedure but with the gradient term included can be seen to the right. A particularly zigzag looking estimate can be seen in (e). Even though none of the estimates are especially good, the general shape are more physically plausible in the right images.*

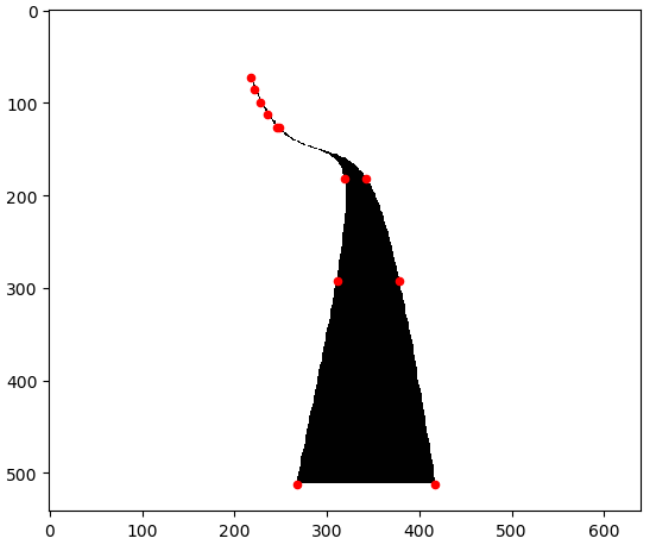


Figure 5.5: *Extraction of keypoints from the old systems' rail mask.*

5.2.5 Real-time performance

GPU Benchmarking RailNetTransfer2 at iteration 7500 on the random sequence set with Caffe, a forward propagation takes on average 4.63 ms. The used sensor takes images with ~50 Hz, i.e. images are captured every 20 ms. Thus, the rail detection is usable in real-time even if every image is to be analyzed.

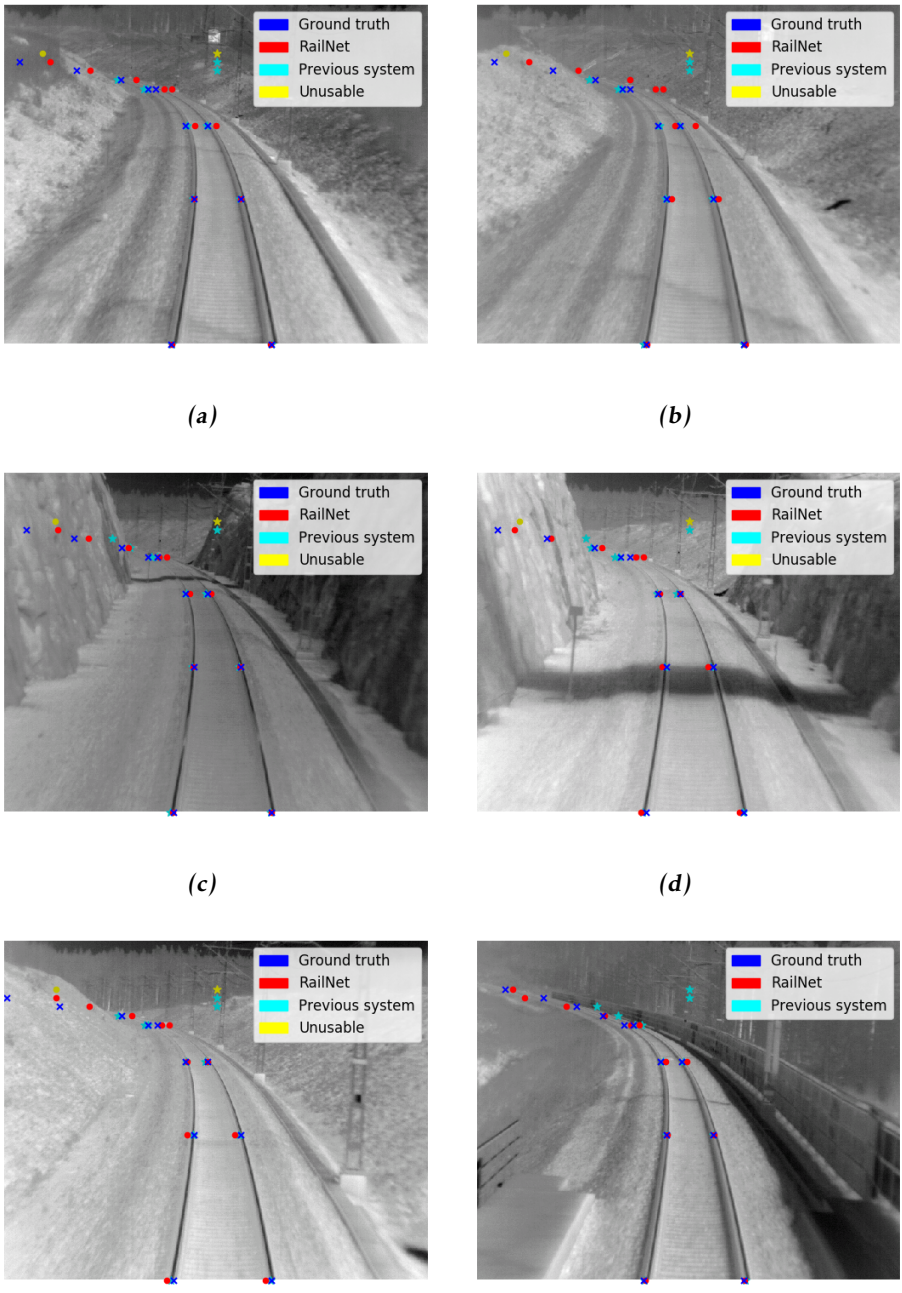


Figure 5.6: *Examples of good estimates from RailNetTransfer2 (11000 iterations) on the random sequence set. In some images one or several estimates from the top have no corresponding ground truth.*

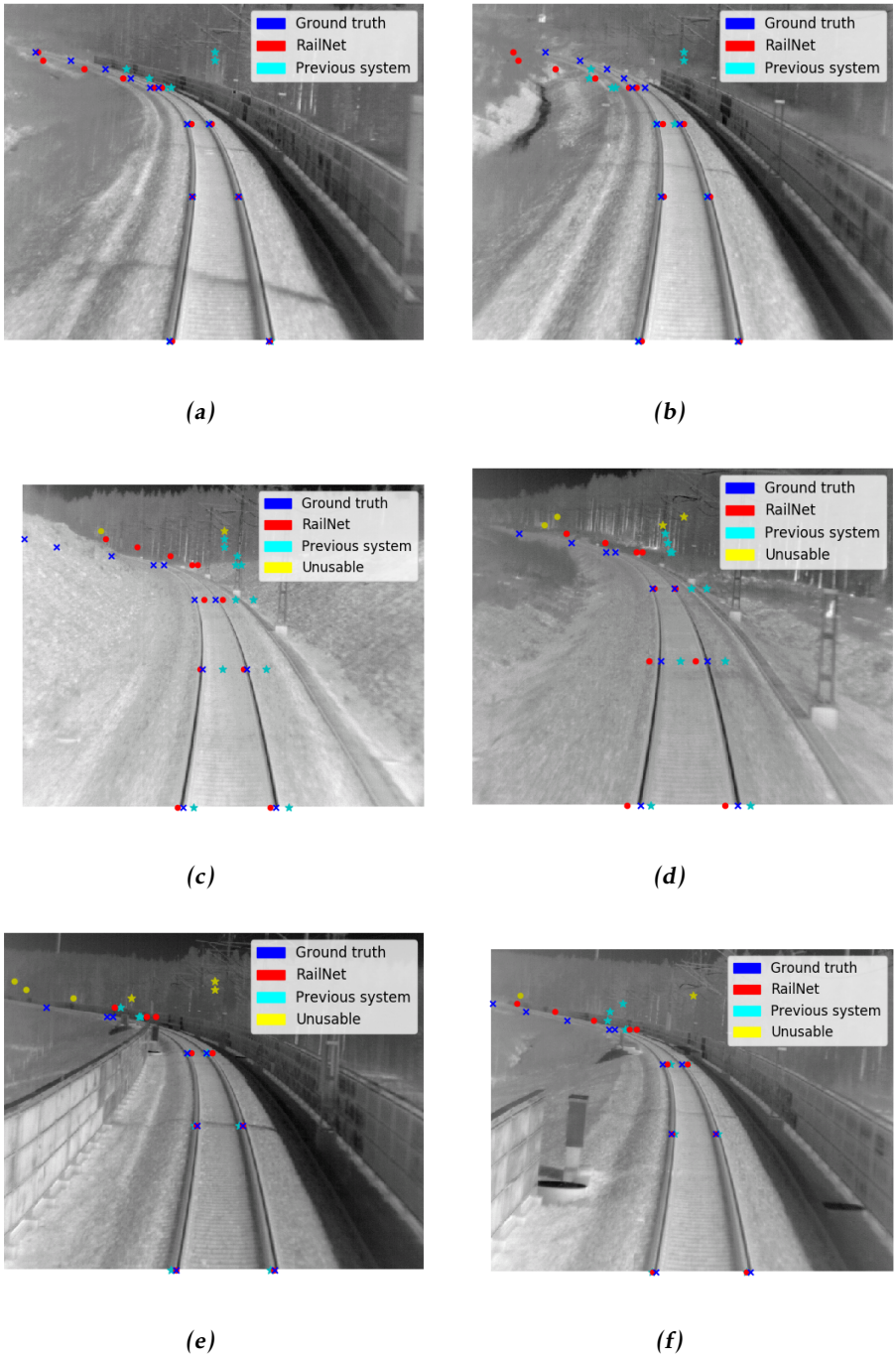


Figure 5.7: *Examples of decent estimates from RailNetTransfer2 (11000 iterations) on the random sequence set. In some images one or several estimates from the top have no corresponding ground truth.*

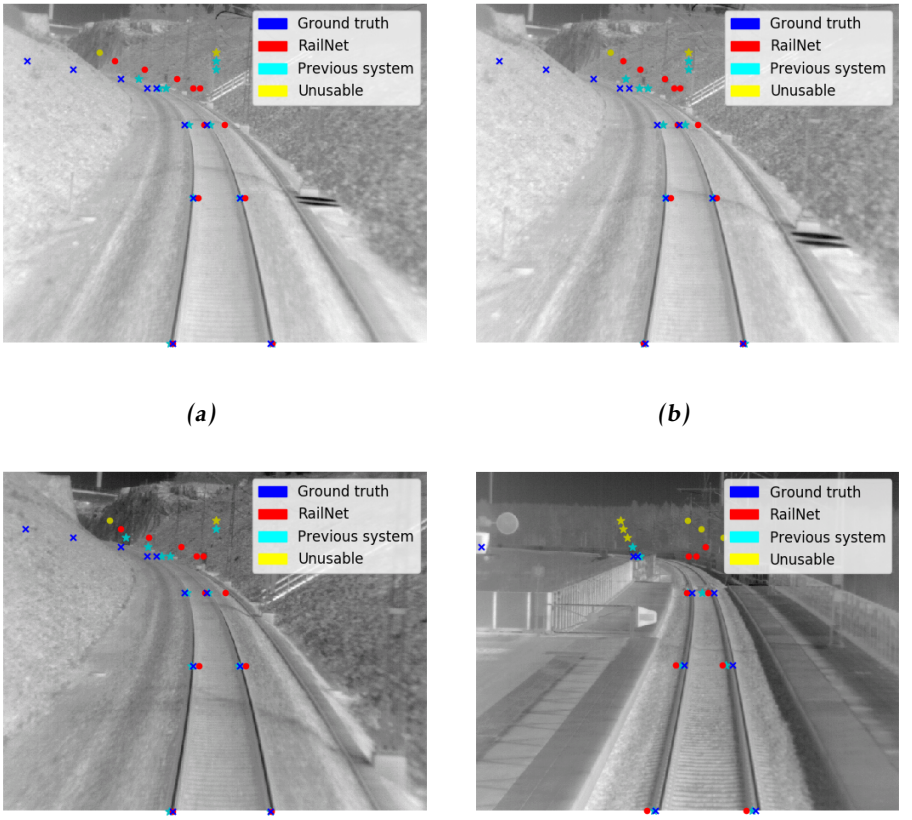


Figure 5.8: *Examples of bad estimates from RailNetTransfer2 (11000 iterations) on the random sequence set. In some images one or several estimates from the top have no corresponding ground truth.*

6 Conclusions

This last chapter discusses the results and relates them to the problem formulation. Finally it proposes future work.

6.1 Summary

The CNN rail detectors show great potential to the problem, achieving better performance than the previous system in a sequence of 2800 frames, see Table 5.5. However, the CNN rail detector needs to be evaluated on a larger dataset in order to see its potential use in a real system.

From the results in Chapter 5 it can also be concluded that the system is usable in real-time and transfer learning is advantageous when it comes to generalization in the rail detection problem.

6.2 Discussion

In this section some of the results from Chapter 5 are discussed.

6.2.1 Comparison with old system

As expected, both the best RailNet (RailNetTransfer2) and the previous system successfully find the rail in the random sequence from the bottom of the image up to (but not including) the top three keypoints most of the time. RailNetTransfer2

distinguishes itself from the previous system in the image area close to $y_{\min}$, by being able to estimate rail keypoints where the rail is not directly visible.

Many RailNet systems outperform the previous system when it comes to loss in the random sequence, as can be seen in Table 5.5. Clearly, the CNN approach is a promising way to solve the rail detection problem. However, only one sequence with 2800 frames is used for comparison. A larger dataset should be used in order to make sure if the new system outperforms the previous one in other sequences as well.

One weakness in the comparison is the use of only one performance measure. It would be preferable to have some complementary error measures, in order to test other aspects of the systems as well.

6.2.2 Labeled data

Since consecutive frames are labeled in the dataset, there are many images that have a similar appearance. This might result in overfitting on the training set. A better idea could be to label every e.g. 100:th frame and use more sequences. A drawback is that it would take more time to label the same amount of frames since interpolation of labels between frames cannot be used (interpolation was used between every 15:th frame in this thesis).

There might, however, be an advantage using consecutive frames to some extent, since translation and scaling of images is a form of data augmentation. It has been identified that data augmentation is the most important factor to obtain translation-invariant representations of images when using CNNs [17].

One problem is that the top midpoints have the highest weight in the optimization, but they also have the highest uncertainty when labeling. This means that estimates that could be considered good can be penalized heavily. In the test set it would have been preferable to use noise augmentation to average the performance. This was unfortunately not done.

Judging by the results in Table 5.5, the performance on the random sequence set is significantly lower than on the test set. It should be concluded that even the best rail detector is overfitted to the training/validation/test frames. Obviously, more labeled frames are needed, at least if the same rail detection approach used in this thesis is to be used.

6.2.3 Transfer learning by fine-tuning

The weights used for fine-tuning are originally trained on visual RGB images. In this thesis, they are used with 1-channel images copied to 3-channels as input. It is not self-evident that this would be a good idea, or advantageous at all, in relation to training the net weights from scratch. Table 5.5 shows though that the net with pre-trained weights outperforms the nets trained from scratch.

It has been shown that transfer learning by fine-tuning using weights as initialization is better than random initialization of weights even if the target task is very different from the source task [37]. The benefit of using the CaffeNet weights just as initialization of the weights is evident from RailNetTransfer3 in Table 5.5.

Alternative existing nets

There are multiple famous pre-trained CNNs released after CaffeNet, e.g. *GoogleNet* [34] and *VGGNet* [31]. It would be a good idea to test these networks as well, and compare them to CaffeNet.

6.2.4 Representation

The representation is a bit problematic around $y_{\min}$. The top keypoint was not usable in a majority of frames in the training set. Since the estimate of it is ignored in the optimization in all the unusable cases, the weight chain leading up to that output is not updated as much in the backpropagation. This is probably the reason why the top keypoint do not tend to move independently in the estimates but rather hang on to the neighbouring keypoint, see for example Figure 5.6c and 5.6d.

By also estimating how far one can see the rail in the image, one could vary $y_{\min}$ and find keypoints in a flexible vertical interval.

6.2.5 Optimization solver

There are many different solvers based on SGD, among others *AdaDelta* [39], *Adam* [18], and *RMSprop* [29]. The different solvers can differ very much in how many iterations that are needed in order to find the minimum. Some of them were tested and some of them were not. It would have been interesting to test more solvers and compare their performance.

6.2.6 Loss function

The loss function is adapted to the problem at hand. It is probably possible to find more parameters and formulate them into mathematical terms that could be added to the loss function in order to restrict the output space and penalize more bad estimate properties.

The weighting between the different terms is important. At some point, a well shaped estimate that is correct up to a translation in the horizontal direction will

by the loss function be conceived as bad as a somewhat zigzag estimate. It appears difficult to quantify the best relative weighting of the terms. In the end, it is up to the user to judge whether a specific weighting is better than another.

How the row weights in L_{wpl} are defined affect the relative importance of keypoints considerably. There are possibly better choices than the one used, especially if the vertical distribution of keypoints is changed.

The derivative of the width loss could have been used too. Although, since the net had no problem to discover that the width should decrease with row index no matter what, it was considered superfluous to use the width derivative.

Gradient loss

In Figure 5.4 the gradient loss is evaluated on some frames. Though, since cases where training without the gradient loss term result in a smaller total error exist (in Figure 5.4c and 5.4e), there should be possible improvements how to use the gradient error term. It is not certain that the weighting between the error terms in L is optimal, since the weights were determined with a brute force method. Also, it might be a good idea to weight the gradient error with a weighting function taking the relative importance of keypoints into consideration, as in the two other error terms.

Origo translation

Late in the thesis work, it was realized that thousands of iterations were spent on moving the bottom estimates towards the horizontal middle of the image. Since the rail generally is located around the horizontal middle of the image, it was concluded that it would be a good idea to move origo accordingly. As evident in Table 5.5, the best rail detector in the random sequence uses a centered origo.

6.2.7 Dataset splitting

The noise augmentation datasets were used separately after the original and mirrored datasets in the training phases. This was due to problems with long computation times converting the images to a memory efficient database. This was probably not the best way to use augmentation, since overfitting might already have occurred before the noise augmentation sets were used.

6.3 Future work

In order to improve the system, a number of actions should be evaluated. The proposed actions are mentioned below.

6.3.1 Usability estimation

In a future system, it would probably be a good idea to estimate the certainty (or usability) of each keypoint. Since the parameters in the network are shared, bad keypoint estimates will affect other keypoints as well. The keypoint located at $y_{\min}$ is usually not usable in the labeled data, so its corresponding weight will not be as adjusted as the weights of the keypoints with higher row index. This is clearly visible in the results, where the $y_{\min}$ keypoint estimates usually are located just above the second top most keypoint.

6.3.2 Unsupervised learning

Unsupervised learning could be used to preprocess the rail images fed to the CNN, in order to remove redundant information in the images, information which might mislead the network.

Stacked autoencoder

A lot of recorded rail data is available. This data could be used in an unsupervised manner, and use a stacked autoencoder to remove redundant image information from the images. These new images could then be used as input to the CNN, and it is very likely that is is easier to train a good net with these images. The amount of needed labeled data is possibly also reduced.

6.3.3 Multiple railways

In a real system it is very important to be able to detect multiple railways at once. The representation need to be modified in order to handle multiple railways in order to be more useful.

6.3.4 Recurrent neural network

Since the rail frames are recorded sequentially in a video sequence, much information is relevant and connected between frames. This time-dependent information is probably very useful, and could be taken into consideration with a recurrent neural network (RNN).

An interesting property of the estimates in Figure 5.8a-5.8c is that it seems as if the bad estimates somehow depends on that the third double pair of keypoints (from the bottom of the image) is misjudged between the right rail and its neighbouring edge in the image (to the right). Since parameters is shared in a CNN net, the mentioned misjudgment should affect other estimates. This misjudgment

was only evident in some consecutive frames. Right before and after this misjudgment effect occurred, the estimate was much better for a longer series of frames. By using temporal information between consecutive frames over a longer period of time, and using the fact that the rail would never suddenly move sideways in a couple of frames and then back again, this misjudgment could be handled properly.

Bibliography

[1] Flir camera specification. URL http://www.flir.co.uk/automation/display/?id=56345. Accessed 2017-03-20. Cited on page 9.

[2] Keras documentation. URL https://keras.io/applications/. Accessed 2017-05-11. Cited on pages 29 and 39.

[3] Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. From generic to specific deep representations for visual recognition. *CoRR*, abs/1406.5774, 2014. URL http://arxiv.org/abs/1406.5774. Cited on pages 13 and 29.

[4] A Berg. *Detection and Tracking in Thermal Infrared Imagery*. Linkoping University Electronic Press, apr 2016. doi: 10.3384/lic.diva-126955. URL https://doi.org/10.3384%2Flic.diva-126955. Cited on pages 7, 8, and 9.

[5] A. Berg, K. Öfjäll, J. Ahlberg, and M. Felsberg. Detecting Rails and Obstacles Using a Train-Mounted Thermal Camera. In *Image Analysis*, volume 9127 of *Lecture Notes in Computer Science*, pages 492–503. Springer, 2015. Cited on pages 1, 2, 17, 18, and 22.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738. Cited on pages 7, 9, and 10.

[7] BVLC. Bair/bvlc caffenet model, . URL https://github.com/BVLC/caffe/tree/master/models/bvlc_reference_caffenet. Cited on pages 27 and 37.

[8] BVLC. Brewing imagenet, . URL http://caffe.berkeleyvision.org/gathered/examples/imagenet.html. Cited on page 38.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. Cited on page 37.

[10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010. Cited on pages 14, 15, 27, and 39.

[11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey J. Gordon and David B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011. URL http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf. Cited on page 14.

[12] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006. ISBN 013168728X. Cited on pages 7, 8, 10, and 11.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. Cited on pages 2, 7, 10, 12, 13, and 14.

[14] A. Gurghian, T. Koduri, S. V. Bailur, K. J. Carey, and V. N. Murali. Deeplanes : End-to-end lane position estimation using deep neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2016. Cited on page 2.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL http://arxiv.org/abs/1502.01852. Cited on page 15.

[16] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014. URL http://arxiv.org/abs/1408.5093. Cited on pages 15 and 40.

[17] Eric Kauderer-Abrams. Quantifying translation-invariance in convolutional neural networks. 2016. Cited on page 52.

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980. Cited on page 53.

[19] Philipp Krähenbühl, Carl Doersch, Jeff Donahue, and Trevor Darrell. Data-dependent initializations of convolutional neural networks. *CoRR*, abs/1511.06856, 2015. URL http://arxiv.org/abs/1511.06856. Cited on page 27.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C.

Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. Cited on page 27.

[21] H. Lai, S. Xiao, Y. Pan, Z. Cui, J. Feng, C. Xu, J. Yin, and S. Yan. Deep Recurrent Regression for Facial Landmark Detection. *ArXiv e-prints*, October 2015. Cited on page 20.

[22] C. L. Liao and H. H. Huang. Study on Shadow Effects of Various Features on Close Range Thermal Images. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 357–361, July 2012. doi: 10.5194/isprsarchives-XXXIX-B5-357-2012. Cited on page 9.

[23] Bogdan Tomoyuki Nassu and Masato Ukai. A vision-based approach for rail extraction and its application in a camera pan-tilt control system. *IEEE Trans. Intelligent Transportation Systems*, 13(4):1763–1771, 2012. Cited on page 2.

[24] Michael Opitz, Georg Waltner, Georg Poier, Horst Possegger, and Horst Bischof. Grid loss: Detecting occluded faces. *CoRR*, abs/1609.00129, 2016. URL `http://arxiv.org/abs/1609.00129`. Cited on page 2.

[25] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, abs/1403.6382, 2014. URL `http://arxiv.org/abs/1403.6382`. Cited on page 2.

[26] W.G. Rees and W.G. Rees. *Physical Principles of Remote Sensing*. Topics in remote sensing. Cambridge University Press, 2001. ISBN 9780521669481. URL `https://books.google.se/books?id=u17Zv45DENoC`. Cited on pages 7 and 8.

[27] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958. Cited on page 10.

[28] Guido Rossum. Python reference manual. Technical report, Amsterdam, The Netherlands, The Netherlands, 1995. Cited on page 40.

[29] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL `http://arxiv.org/abs/1609.04747`. Cited on pages 37 and 53.

[30] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/j.neunet.2014.09.003. URL `//www.sciencedirect.com/science/article/pii/S0893608014002135`. Cited on page 2.

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL http://arxiv.org/abs/1409.1556. Cited on page 53.

[32] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2670313. Cited on page 14.

[33] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep convolutional network cascade for facial point detection. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '13, pages 3476–3483, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-0-7695-4989-7. doi: 10.1109/CVPR.2013.446. URL http://dx.doi.org/10.1109/CVPR.2013.446. Cited on page 20.

[34] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. URL http://arxiv.org/abs/1409.4842. Cited on page 53.

[35] Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neural networks. *CoRR*, abs/1312.4659, 2013. URL http://arxiv.org/abs/1312.4659. Cited on page 13.

[36] Stefan van der Walt, S. Chris Colbert, and Gael Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science and Engg.*, 13(2):22–30, March 2011. ISSN 1521-9615. doi: 10.1109/MCSE.2011.37. URL http://dx.doi.org/10.1109/MCSE.2011.37. Cited on page 40.

[37] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014. URL http://arxiv.org/abs/1411.1792. Cited on pages 29 and 53.

[38] Jun Yuan, Bingbing Ni, and Ashraf A. Kassim. Half-cnn: A general framework for whole-image regression. *CoRR*, abs/1412.6885, 2014. URL http://arxiv.org/abs/1412.6885. Cited on page 20.

[39] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. URL http://arxiv.org/abs/1212.5701. Cited on pages 37 and 53.

[40] Gao Zhu, Fatih Porikli, and Hongdong Li. Robust visual tracking with deep convolutional neural network based object proposals on pets. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2016. Cited on page 2.

[41] Matthijs H. Zwemer, Dennis W. J. M. van de Wouw, Egbert Jaspers, Sveta Zinger, and Peter H. N. de With. A vision-based approach for tramway rail extraction. In Robert P. Loce and Eli Saber, editors, *Video Surveillance and Transportation Imaging Applications 2015*. SPIE-Intl Soc Optical Eng, mar 2015. doi: 10.1117/12.2075641. URL http://dx.doi.org/10.1117/12.2075641. Cited on page 2.