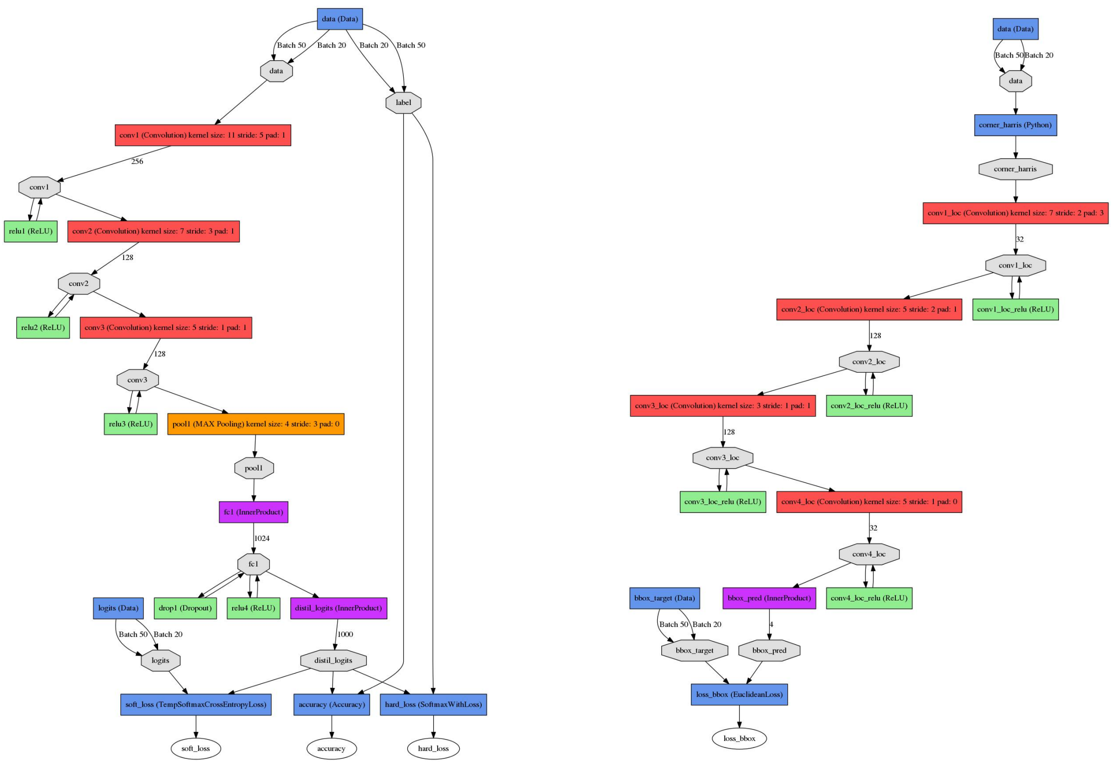
CHALMERS
UNIVERSITY OF TECHNOLOGY



Object classification and localization using machine learning techniques

Designing and training models for use in limited hardware-applications

Master's thesis in Complex Adaptive Systems

CARL ASPLUND

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2016

Master's thesis 2016

Object classification and localization using machine learning techniques

Designing and training models for use in limited hardware-applications

CARL ASPLUND

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2016

Object classification and localization using machine learning techniques
Designing and training models for use in limited hardware-applications
CARL ASPLUND

Supervisor: PER-LAGE GÖTVALL, Volvo Group Trucks Technology
Examiner: MATS GRANATH, Department of Physics, University of Gothenburg

Master's Thesis 2016
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The graphical representation of the two models developed in this thesis.

Typeset in LaTeX
Gothenburg, Sweden 2016

Object classification and localization using machine learning techniques
Designing and training models for use in limited hardware-applications
CARL ASPLUND
Department of Physics
Chalmers University of Technology

Abstract

When working with object classification and localization in image data, the development of traditional rule-based solutions has stagnated in recent years. In its place, machine learning has become a major field of research in order to handle more and more complex image recognition problems. With machine learning, new state-of-the-art models can be developed by training a model instead of implementing an explicitly programmed feature detector.
In this thesis, a literature study covering the field of machine learning has been carried out on behalf of Volvo Advanced Technology and Research. Furthermore, with an autonomous garbage handling project initiated by Volvo in mind, two machine learning models meant for limited hardware-deployment have been designed and trained. The classification model is based on knowledge distillation, where a compact model learns to generalize from a more complex state-of-the-art model, and a localization model, where a typical machine learning implementation is combined with computer vision solutions from the OpenCV framework.
Both models, that were trained on images from the ImageNet database, produced poor results in their respective tasks. The process of knowledge distillation, used to train the classifier, was not achievable due to unfortunate choice of cumbersome model combined with hardware limitations during training. The hardware was also an issue for the localization model, which due to this and unwanted performance from the OpenCV corner detector converged early during training and ended up producing unchanged results for different input. However, the thesis as a whole came to important conclusions regarding a proper next step in order to stay competitive within the field of machine learning.

Keywords: machine learning, computer vision, classification, localization, neural networks, convolutional neural networks.

Acknowledgements

Firstly, I would like to thank my supervisor Per-Lage Götvall for taking on my project and providing valuable feedback. I also thank Daniel Lexén, my initial supervisor, who was very helpful and engaged in the project during the time he was involved.
Furthermore, I would like to thank Erik Ylipää and Abubakrelsedik Karali at SICS Swedish ICT research institute for their input and advice regarding the technical details of the project.
Finally, I thank my family and friends for all their support and patience throughout my entire education, but also for advice and proofreading during this thesis.

Carl Asplund, Gothenburg, June 2016

Contents

List of Figures

List of Tables

1 Introduction

This chapter introduces the topic of the thesis and clarifies the purpose of this work and what it aims to accomplish.

1.1 Background

With their products sold and serviced in more than 140 countries and a workforce of 17000 employees, Volvo Trucks is the second largest producer of heavy-duty trucks in the world. About 95 % of their production capacity is located in Sweden, Belgium, Brazil and the USA, so they are without a doubt a major industrial actor with heavy local representation [35].

Volvo Trucks is a company that is used to being at the forefront of technology and is always trying to keep up with new emerging fields of research. An example of a technological innovation in recent years that drew a lot of public attention is the Dynamic Steering system, promoted by the actor Jean-Claude Van Damme [36]. The ATR division (Advanced Technology and Research), a part of Volvo Group Trucks Technology, invests a lot of time and effort in evaluating emerging technologies and trying to assess their compatibility with the Volvo Trucks business model. Apart from investigating interesting fields of technology, Volvo ATR also takes part in different product development and research projects.

1.1.1 The ROAR project

The Robot-based Autonomous Refuse handling project, ROAR, is based on a collaboration between Volvo Group, Chalmers University of Technology, Mälardalen University, Penn State University and Renova. The projects aims to present a robot that automatically collects and empties refuse bins. Another important objective of the project is to demonstrate the usefulness of smart machines [15]. Volvo foresees a world with more automation, and a system like this would result in garbage handling that is less noisy and involves no heavy lifting for the driver [16].

A drone located at the roof of the truck starts simultaneously with the robot and scans the area to locate bins. The drone then communicates the position to the robot which then handles the emptying of the bin. The whole process as well as the position of the robot can be monitored by the driver inside the truck. Mälardalen University developed the actual robot platform, Chalmers University of Technology developed the task management system, and Penn State University has developed the graphical interface handling overview and control of the process [15].

In order to determine its own position, the robot already knows a map of the maneuverable area as well as likely bin locations. It uses several tools to keep itself positioned inside the map, such as GPS, LiDAR (a system similar to RADAR that uses infra-red light instead of radio waves), cameras and an IMU (inertial measurement unit). The IMU is based on accelerometers, gyros and odometry (measure position over time). The cameras are also used to, for example, detect people coming to close to the robot which will abort the emptying process [15].

1.1.2 Machine learning

In applications such as object classification, detection and localization, hand engineered solutions to address these task have plateaued in recent years [21]. In its place, a new approach that appeared decades ago but has boomed during the last few years has become an important option in order to develop new systems. It is called machine learning, and it aims to encompass automatic computing and can, based on logical or binary operations, learn a task from a series of examples. Given sufficient data, machine learning can in theory represent any problem of any complexity [26]. With the two key prerequisites, processing power and sufficient data, a machine learning system can produce highly generalized non-linear mappings without using explicit programming. Since its rise in popularity, the research around machine learning has increased rapidly and has already resulted in the technology being tried on multiple industrial applications within different fields [7, 24, 30].

1.2 Purpose

The main purpose of this master's thesis is to provide the Volvo ATR division a first step into the up and coming-world of machine learning and its applications. This will be carried out through a literature study, looking into the underlying structures, tools, necessary hardware and the state-of-the-art models that exist today.

Furthermore, the techniques included in the machine learning field will be applied to an existing Volvo project. The ROAR project is at this point reliant on an extensive mapping of the usage area in order for the robot and drone to function properly. Considering that the robot is equipped with cameras, a machine learning model made for classification and localization of garbage bins would simplify the work of this system to a large extent. The second main goal of this thesis is therefore, with the limited hardware access of the robot platform in mind, to investigate the possibility to produce such a model using modern machine learning solutions.

1.2.1 Specification of thesis aims

- Map the field of machine learning through a literature study, highlighting the state-of-the-art.
- Investigate the development of models for object classification and localization tailored for limited hardware-applications.

1.3 Delimitations

In order to adapt the project to the master's thesis time frame, which is relatively limited, some limitations have to be set. Based on the nature of machine learning, the project will mainly be limited by two aspects: data and computational power. When it comes to data, the availability of free-to-use image databases makes it possible to cope with the need of large amounts of labeled (annotated) data. However, in order to achieve task-relevant results there is a large need for task-specific data, which depending on the prior data collecting approach might be limited for the project at hand.

Should one manage to obtain the data needed, the next problem that occurs is the computational power needed to perform the training. An ill-designed or too weak computation setup may likely jeopardize important factors such as model complexity, the amount of training data being handled at once and computation time. Within the time frame of the project, learning the necessary theory, setting up the hardware and preparing data has to be successfully executed before any training can be carried out. Hence, the training time may be quite limited, and should it be that the model can not be large or complex enough, the results and models produced may lack relevance. However, performing any training and testing new approaches may still provide valuable conclusions for the company.

The final major limitation for the prototype to be produced within this thesis project is the actual industrial implementation. The goal is to achieve something of a prototype that is as relevant as possible to Volvo in the evaluation of this technological field. However, actually producing a functioning product that can be implemented in the real garbage handling system proved to be very difficult to achieve during the course of the thesis, mainly because of the extent of the project. The results will hopefully be of great interest for the company, but its impact on actual products might probably come at a later stage in this process.

1.4 Outline

In chapter 2, the necessary theory along with tools and state-of-the-art models will be presented in a literature study. In chapter 3, the data used, its preprocessing, the models designed and the training procedures will be presented. In chapter 4, the results of the training will be presented. Chapter 5 will provide a discussion on the results produced as well as the different techniques used, and look at what could be a suitable next step within this field. Finally, chapter 6 sums up the conclusions drawn in this thesis.

2
Theory

The theory for this project will include the fundamentals of machine learning based on artificial neural networks and a walk through of the main principles and tools applied to this particular project. It also includes descriptions of state-of-the-art-models within the field.

2.1 Artificial neural networks

Even though computers are designed by and for humans, it is clear that the concept of a computer is very different from a human brain. The human brain is a complex and non-linear system, and on top of that its way of processing information is highly parallelized. It is based on structural components known as neurons, which are all designed to perform certain types of computations. It can be applied to a huge amount of recognition tasks, and usually performs these within 100 to 200 ms. Tasks of this kind are still very difficult to process, and just a few years ago, performing these computations on a CPU could take days [18].

Inspired by this amazing system, in order to make computers more suitable for these kinds of task a new way of handling these problems arose. It is called an artificial neural network (ANN). An ANN is a model based on a potentially massive interconnected network of processing units, suitably called neurons. In order for the network and its neurons to know how to handle incoming information, the model has to acquire knowledge. This is done through a learning process. The connections between the neurons in the network are represented by weights, and these weights store the knowledge learned by the model. This kind of structure results in high generalization, and the fact that the way the neurons handle data can be non-linear is beneficial for a whole range of different applications. This opens up completely new approaches for input-output mapping and enables the creation of highly adaptive models for computation [18].

The learning process itself generally becomes a case of what is called supervised learning, which is described in the next segment.

2.1.1 Supervised learning

The concept of supervised learning, applied to an artificial neural network, is about tuning its weights so that it performs a desired mapping of input to output activations. The mapping itself is given by a so called pattern set, containing input activation vectors and target activation vectors. The output vector produced by the network when given a certain input vector should equal the corresponding target vector. The fitness of the weights, in other words how well the network is doing, is measured using a so called loss (alt. energy or cost) function that produces a total loss value for the network. Based on this total loss, the weights are now shifted along a search direction in weight space. This step in weight space is scaled by a learning parameter, usually known as the learning rate. The direction of the step is most commonly given by looking at first order derivative information, known as the gradient. This error adjustment then succesively propagates through the network from output to input layer, a process known as backpropagation [32].

2.2 Convolutional neural networks

Convolutional neural networks came along much later than the regular neural networks, but still share many of their characteristics. They consist of neurons with learnable weights and biases that process data and possibly pass it through non-linear functions. They also use a loss function to compute an error that can be propagated through the network. However, as the input data of a regular neural network is scaled up into, for example, an image, the very extensive network connecting all neurons tends to grow unmanageably large. The structure of a convolutional neural network is explicitly designed to handle this issue, and it is all based on a set of layer types specialized for this task [22].

The main feature is the so called convolutional layer. This layer accepts a three dimensional input, typically an image with height H_1, width W_1 and a number of channels (or depth) D_1. It consists of a set of K three dimensional filters with sides according to the set filter size (or kernel size) F and the same depth as the input, D_1. The filters are basically sets of neurons with weights and a bias that handle data just like in a regular neural network. When producing output, each filter will sweep the input image, and for each step produce a value to the output. How the filter steps through the image is determined by the stride S, which is the distance between two points in the filter's image covering grid. In order to approach the edges of the image, another parameter P determines the amount of zero-padding, meaning the number of layers of zero values added around the image that will also be included in the sweep. This way, the output size can equal the input size regardless of the kernel size. After all filters in the layer have gone through the image input, the two-dimensional outputs of all filters are stacked up to create the output volume with dimensions:

- width: $W_2 = (W_1 - F + 2P)/S + 1$
- height: $H_2 = (H_1 - F + 2P)/S + 1$
- depth: $D_2 = K$

This structure have multiple perks when processing large input data such as images. Since a relatively small filter can be used over the entire image, the amount of parameters decreases heavily. Still, by having many filters working the image, different filters can be trained to look for different features. Also, by having different filter sizes in different layers, one can easily build a model that detects features at different scales in the image [22].

Once a convolutional layer has processed and restructured the data from the input or previous layer, the output volume is often sent through a ReLU-layer (Rectified-Linear).

It applies an elementwise activation function, most commonly a $max(0,x)$-function that sets all negative values to zero. This becomes the activation function of the layer that introduces non-linearity, which is crucial for the networks ability to map different types of functions. It is important to note that this layer leaves the size of the output intact [22].

In order to progressively reduce the spatial size of the output volume and decrease the amount of parameters to be computed inside the network, it is common to periodically insert what is called a Pooling layer in-between successive convolution layers. The down-sampling operation performed by the layer is also important in the role of preventing overfitting, where the network becomes too accustomed to the images it is being trained on and performs worse on other images. The layer resizes the data spatially by sweeping each depth slice in the input volume and applying a max-operation. Just like the convolutional layer, the pooling layer has a spatial extent (kernel size) F, where the local max-operation is applied, and a stride parameter S. It is important to note that as the kernels sweep the different depth slices, the total depth of the volume is unchanged. If a volume with width W_1, height H_1 and depth D_1 is fed into a pooling layer, the produced output has dimensions:

- width: $W_2 = (W_1 - F)/S + 1$
- height: $H_2 = (H_1 - F)/S + 1$
- depth: $D_2 = D_1$

This changes if zero-padding is introduced, but this is not common for pooling layers. Other operations than the max can also be used, such as *average* or $L2 - norm$. The max has however been shown to work best for most situations. Since the function that this layer applies is fixed, it introduces no new parameters to the model [22].

A convolutional neural network is often finished with regular feed forward layers, which recently have adopted the name fully-connected layers. These are, as mentioned, due to all neurons in the layer connecting to all neurons in the previous, expensive to train. Being at the end of the network, the input is usually manageable and they are used to compute the final output. This would, for example, mean the class scores for an image classifier [22].

2.3 Stochastic gradient descent

When training neural networks the adjustment of the defining weights is usually handled with backpropagation based on gradient descent. In recent years, however, as neural networks have grown larger, a new variation on this algorithm has emerged and has become a frequent option for large training processes: Stochastic gradient descent [3]. In order to describe this algorithm, take a simple example of supervised learning. There are n patterns (x,y) that are supposed to be mapped using a function f. This results in the equation:

$$E_n(f) = \frac{1}{n}\sum_{i=1}^{n} l(f(x_i), y_i) \tag{2.1}$$

Here, $l(f(x),y)$ is the loss functions that compares the output $f(x)$ to the target y, and E_n is the so called Empirical risk (basically average loss) based on the n patterns. In order to look at the actual adjustment of the weights, let z represent a pattern pair (x,y)

and $Q(z,w) = l(f_w(x),y)$, where w represent the weights of the network and f_w is the network function using that set of weights. The weight update equation based on the original gradient descent algorithm at time t can then be written as:

$$w_{t+1} = w_t - \gamma_t \frac{1}{n} \sum_{i=1}^{n} \nabla_w Q(z_i, w_t) \tag{2.2}$$

Here, γ_t is the learning rate at time t and ∇ is the gradient operator. The variable t is the current time step or iteration. This algorithm will perform the training in the sought way. However, as the network grows larger and the amount of weights increases, this algorithm becomes computationally heavy. To address this issue, Stochastic gradient descent handles this by not calculating the gradients based on all the patterns z in the set. Instead, in each iteration, a random subset from $\{z_1,...,z_n\}$, S, is chosen. With the amount of pattern pairs in S being n_S, this results in the new weight update equation:

$$w_{t+1} = w_t - \gamma_t \frac{1}{n_S} \sum_{z_i \in S} \nabla_w Q(z_i, w_t) \tag{2.3}$$

Even though (2.3) does not converge as fast as (2.2), the computational gains when working with larger models are so big that is has become a common choice. It is hence used primarily when training time is the bottleneck [3].

Another common addition to the training process is momentum. By adding a momentum term, the weight update step becomes:

$$\Delta w_{t+1} = -\gamma_t \frac{1}{n_S} \sum_{z_i \in S} \nabla_w Q(z_i, w_t) + \mu \Delta w_t \tag{2.4}$$

Here, Δw_{t+1} is the new weight update step, Δw_t is the old update step and μ is the momentum parameter. This parameter scales how much influence the previous weight-step should have on the next one. This has been proven to be a good approach for a lot of learning applications. It is however recommended to decrease the learning rate regularly when using momentum in order to keep the training stable [32].

Just like the momentum, another common option when training a neural network is weight decay. It means that all the weights decrease in value at a rate λ, which in practice means multiplying them with $(1 - \lambda)$. This has been proven to improve generalization and suppress the effects of static noise in the data [28]. Adding the weight decay rate to the weight update step results in:

$$\Delta w_{t+1} = -\gamma_t \frac{1}{n_S} \sum_{z_i \in S} \nabla_w Q(z_i, w_t) + \mu \Delta w_t - \lambda w_t \tag{2.5}$$

Concluding all these techniques, the final weight update equation now becomes:

$$w_{t+1} = w_t - \gamma_t \frac{1}{n_S} \sum_{z_i \in S} \nabla_w Q(z_i, w_t) + \mu \Delta w_t - \lambda w_t \tag{2.6}$$

This equation is applied to each layer in the network in order to update its weights, regardless of if it is a regular network or a convolutional one. Since the input to an intermediate layer depends on the output (and thereby the weights) of the previous layer, determining $\nabla_w Q(z,w)$ is for most layers an instance of the chain rule. This way, the error propagates back through the network, which is what is known as backpropagation.

2.4 CUDA

The rapid development within the field of computational science results in a never-ending demand for more processing power and computational efficiency. Everything from finance to physics is to a large extent relying on a modern hardware architecture in order to produce interesting results in a reasonable amount of time. In recent years, looks have turned towards the graphical processing unit, or GPU. These devices, that until just recently were used solemnly to render graphical content, have become the future hope of fast parallelized computing. An important step towards making full use of these devices was in 2007 when Nvidia first showed their CUDA (Compute Unified Device Architecture) platform [23]. Now, a large range of Nvidia GPU products could relatively easily be made into specialized computation devices for use in both industry and academia.

The GPU is a massively parallel processor and supports thousands of active so called threads. To make use of GPU computing however, one needs a programming model that can express this level of parallelism in an efficient way. That is what CUDA provides. When using CUDA, an application is portioned into so called kernels. Each kernel is then executed by a grid of thread blocks on the GPU device. One of the things that makes CUDA fast is that the threads in these blocks can cooperate and use a shared memory. All of this is carefully orchestrated by the CPU, which communicates with the GPU through CUDA [23].

2.5 Caffe

In order for the development of machine learning, deep neural networks and CNN:s to really flourish, the steep learning curve of the hard-to-grasp world of GPU computing has to be managed. The key to making this technology available and get academia to really dig in to the science behind it is to offer an off-the-shelf, relatively easy to use, development tool that makes use of CUDA as its backbone. That is what the Berkeley Vision and Learning Center (BVLC) had in mind when they in 2014 released the platform that they call Caffe, which stands for *Convolutional architecture for fast feature embedding* [21]. Caffe is a C++ based open-source platform that offers all the tools you need to set up your own deep machine learning process. It also features well-supported bindings to both Python and Matlab, which makes getting started with it even easier. Caffe comes with a whole library of functions and computational layers to design and train a machine learning model. In recent years, most state-of-the-art performing models within this field have been developed and trained in this environment [19, 31].

One of the key features of Caffe's success is the separation between representation and implementation. The design and setting of all parameters in the model is done in a separate description in the Google Protocol Buffer format. In this file, all layers (such as fully connected, convolutional or pooling) are set up with all their parameters, the data to be used in training is defined, and the loss layer saying what type error will be propagated through the net is determined. Once the model is defined, it is loaded in the Caffe solver (which is also defined in a Protocol Buffer file). The solver then communicates and distributes the computations throughout the C++ code that is the foundation of the platform. Caffe is also well integrated with CUDA, and switching between CPU and GPU computing is as simple as one line in a file or one function call in Matlab or Python. The representations remains the same, but the solver will then use the CUDA-versions of all layers and functions. This seamless integration removes the need of specialized hardware, since a model can be efficiently trained on a GPU system and then easily implemented in a CPU-system without having to change the model definition [21].

Once the training is actually performed, the Caffe solver module offers multiple well-tested learning algorithms, a common choice being the stochastic gradient descent described in section 2.3. In order to make training as quick as possible, Caffe supports a number of different well-established data formats that result in more efficient data handling. Today, the first hand choice is the so called LMDB (Lightning Memory-mapped Data Base) format that offers the greatest IO-performance to this date [8]. Once the needed databases are prepared, the Caffe model handles the loaded training data in what is called Blobs. These are four-dimensional vectors that are integrated with the computational layer design. All of this combined results in groundbreakingly fast training processes, which is key since the future of deep machine learning depends on the possibility to perform heavy training processes within a reasonable amount of time.

2.6 ILSVRC

The ImageNet Large Scale Visual Recognition Challenge, or ILSVRC, is the benchmark and state-of-the-art determining competition focusing on object category classification and detection. Starting in 2010, the competition has been held annually, and the amount of participants has been rising steadily. It resembles the older competition PASCAL VOC that was first held back in 2005 [10]. Both PASCAL VOC and ILSVRC consist of two main parts: the publically available dataset that is supposed to enable researchers to deepen their knowledge in the field of object recognition and detection, and the annual competition where solutions are discussed, developed, but first and foremost tested with a standardized evaluation procedure. This way, each year, the number one categorical object recognition algorithm can be determined [33].

The images used for the ILSVRC dataset are taken from the ImageNet database [9], which is a large open database with subsets structured according to the WordNet architecture [27]. The database is today maintained by the Stanford Vision Lab, Stanford University and Princeton University [2]. The images are manually annotated, but there are also images meant for testing purposes that are not annotated. The annotations are of two different kinds: image-level annotation, which holds for example class information about the entire image, and object-level annotation, which usually means position information about objects in the image. Manually annotating images became a major challenge when moving from the PASCAL VOC challenge, which had about 20000 images in 20 different classes [10], to ILSVRC which includes more than 1.4 million images in 1000 classes. The solution to provide accurate large scale annotations was to apply different crowdsourcing solutions, such as the Amazon Mechanical Turk [33].

ILSVRC is divided into several subchallenges, where two are of certain interest for this thesis. The first one is Image Classification, where the algorithms competing create a list of objects categories that they conclude are present in the image. The dataset for this challenge are photographs carefully collected from Flickr and other search engines (like all images in ImageNet) and are manually labelled with 1000 categories. Each photograph has an image-level annotation in the form of a ground truth class label. The training set consists of 1281167 images (732 to 1300 for each class), the validation set has 50000 images (50 for each class), and finally the test set with 100000 unlabeled images [33].

The second subchallenge focuses on single-object localization. Here, the algorithm should produce a list of object categories presumably present in the image, but also a bounding box for each present category that is axis-aligned and properly scaled. The dataset used is a subset of the one used in the Image Classification, limited by the amount images that include object-level annotations in the form of bounding boxes. Each image present in the subset includes one bounding box. The training set here consists of 523966 images (91 to 1268 for each class), but the validation set and test set are the same as for the classification task [33].

2.6.1 Licensing

Regarding legal issues when making use of the ImageNet database, it is important to note that ImageNet does not own the copyright to these images. They only provide access to them for non-commercial and educational use [2]. Being internet photos collected from search engines, they might very well be under copyright. The ImageNet metadata, such as labels, bounding boxes etc., is declared to be freely available [2]. Uncertainty arises when training a model using this data, is it then permitted to, for example, release this model for unrestricted use? The research team behind the platform Caffe have faced this particular problem, and their understanding is that no restrictions are placed on the open release of the learned model weights. This holds since none of the original images, in whole or in part, are distributed. Hence, the model itself is generally not considered derivative work of the copyrighted images [1].

2.7 Knowledge distillation

When trying to achieve the highest accuracy possible with, for example, classification models based on neural networks, a common option is to create an ensemble of large models. These models can be specialized towards different tasks and by averaging their results, usually with some kind of weighted sum, one can achieve very high accuracy. It is no secret, however, that these big ensembles are very cumbersome and computationally expensive, which makes them unsuitable for real time applications [20]. Several approaches have been tried to address this problem, and it has been shown that large models or ensembles like these can in many cases, using new training structures, be compressed into smaller models [6]. Using for example a highly generalized but cumbersome model, one can use what is called distillation to transfer knowledge to a small model that is more suitable for deployment.

When performing knowledge distillation, the goal is to make use of the entire probability distribution computed by the cumbersome model, not just train against a single label. The relative probabilities of all the incorrect classes contain a lot of information on how the cumbersome model tends to generalize. The goal is to, instead of trying to optimize the small model on training data, use the pre-trained cumbersome model to make the small model generalized and perform well on test data [20].

The class probability distribution from a classifying model is usually computed by ending the model with what is called a Softmax-layer:

$$q_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}} \tag{2.7}$$

Here, q_i is the probability of class i, and T is the temperature of the Softmax implementation. A higher temperature results in a softer distribution of probabilities. The last variable, z_i, is the logit corresponding to class i, meaning its corresponding value from the second last layer (the usually fully-connected layer before the Softmax that has the same number of outputs). In order to make the distributions computed by both models more easily and efficiently comparable, a higher temperature Softmax function is used during training. Raising the temperature parameter smoothens out the distributions. Using a high-temperature distribution as target is called using soft targets. Having high entropy among these targets means that more information is included per training case and results in less variance in the model gradients. On the other hand, in this context training with regular labels and low temperature is called hard targets [20].

In order to describe the distribution matching model, the logits of both networks must be used. Let z_i be the logits of the distilled model (with probabilities q_i) that is to be trained, and v_i the logits of the cumbersome model (with probabilities p_i) which have been produced in advance. The models are compared using a cross-entropy loss. Cross-entropy is an information-theoretic distance between two probability distributions, basically a measure of their similarity. A smaller cross-entropy corresponds to the distributions being more similar. With the temperature of the Softmax used being T, the cross-entropy loss C for the two probability distributions becomes [25]:

$$C(p,q) = \sum_i p_i \cdot log\left(\frac{p_i}{q_i}\right) \tag{2.8}$$

The resulting cross-entropy gradient $\partial C/\partial z_i$ that controls the backpropagation becomes:

$$\frac{\partial C}{\partial z_i} = \frac{1}{T}(q_i - p_i) = \frac{1}{T}\left(\frac{e^{z_i/T}}{\sum_j e^{z_j/T}} - \frac{e^{v_i/T}}{\sum_j e^{v_j/T}}\right) \tag{2.9}$$

The derivation of this gradient is described in Appendix A. If the temperature T is high compared to the magnitude of the logits, this can be rewritten as:

$$\frac{\partial C}{\partial z_i} \approx \frac{1}{T}\left(\frac{1 + z_i/T}{N + \sum_j z_j/T} - \frac{1 + v_i/T}{N + \sum_j v_j/T}\right) \tag{2.10}$$

Here, N is the number of classes. Assume that the logits are zero-meaned separately for each transfer case, meaning that $\sum_j z_j = 0$ and $\sum_j v_j = 0$. Then (2.10) simplifies to:

$$\frac{\partial C}{\partial z_i} \approx \frac{1}{NT^2}(z_i - v_i) \tag{2.11}$$

This concept has proven effective and has also been implemented for Caffe [38]. Regarding the choice of temperature, Hinton *et. al* have shown that when the smaller model has large intermediate layers $T = 8$ and above produce similar results. When these layers are smaller, $T = 2.5 - 4$ shows better results than both below and above. Using this concept, it has been shown that distilling works well for transferring knowledge from a highly regularized model into a small and light model [20].

An important question to address is how to combine soft and hard targets when training. Combinations of the two have shown promising results, for example using a weighted average. Here, it is important that the training is more heavily based on the soft targets to achieve good results. It must also be taken into account that the magnitudes of the gradients for the soft targets scale as $1/T^2$ and therefore has to be multiplied by T^2. This way, the relative contribution stays the same when the temperature is changed for the soft targets [20]. Another approach is to use the soft-targets as pre-training. The hard targets are then used to perform fine-tuning on the model to make it more precise. This has also proven to be successful [39].

2.8 OpenCV

Computer vision can be defined as the transformation of data from a still or video camera into an actual decision or possibly a new representation. A new representation could mean extracting and presenting contextual information, and a decision could be determing whether certain information is contained in the data or not. As visual creatures, it is easy to imagine this as a relatively simple task, but a computer is very different from the human brain. The brain has an advanced prioritizing attention system that makes decision based on reference. A computer on the other hand has to find the right information in a noisy numeric grid without reference, which of course has proven to be a difficult task.

In order to hopefully improve the results within this field, in 2000, OpenCV was released [4]. OpenCV is an open source computer vision library that aims to provide basic tools for solving computer vision problems. It is written in C and C++ and runs on all major platforms. Today it also features a Python binding. It is designed for multi-core usage and includes hundreds of functions covering fields such as medical imaging, security, stereo vision and robotics. Its high-level features may in many cases be enough to handle the task at hand, but mostly it is about creating code using the basic features and create a solution tailored for your task. Since computer vision and machine learning go hand in hand, OpenCV also features a full-fledged machine learning library [5].

2.9 Harris corner detector

In the early days of image processing, state-of-the-art edge-filtering models were not able to handle junctions and corners. A small change in edge strength or in pixellation would cause large changes in the edge topology, which was a big issue. The solution was to create a combined corner- and edge detector, and the result was what is called the Harris corner detector [17].

The model considers a local window in the image. In order to avoid noise, it uses a smooth and circular gaussian to describe the window. The point (u,v) in the window region w around the point (x,y) in the image can hence be written as $w_{u,v} =$

$e^{-((u-x)^2+(v-y)^2)/2\sigma^2}$, where σ is the variance. The model aims to determine the average changes of image intensity when shifting the window in different directions. If the image intensity is written as I, the change E based on a shift (x,y) can be written as:

$$E_{x,y} = \sum_{u,v} w_{u,v}[I_{x+u,y+v} - I_{u,v}]^2 \tag{2.12}$$

By performing a taylor expansion, this can be rewritten as:

$$E_{x,y} = \sum_{u,v} w_{u,v}[xX + yY + O(x^2,y^2)]^2 \tag{2.13}$$

Here, X and Y are defined as:

$$\begin{aligned} X &= I \otimes (-1,0,1) \approx \frac{\partial I}{\partial x} \\ Y &= I \otimes (-1,0,1)^T \approx \frac{\partial I}{\partial y} \end{aligned} \tag{2.14}$$

The operator $\otimes$ is defined as the vector to the right applied (using multiplication) to each position in the matrix to the left, then summed over to create a new matrix. For small shifts, E can be written as:

$$E(x,y) = Ax^2 + 2Cxy + By^2 \tag{2.15}$$

Where A, B and C are defined as:

$$\begin{aligned} A &= X^2 \otimes w \\ B &= Y^2 \otimes w \\ C &= (XY) \otimes w \end{aligned} \tag{2.16}$$

Define the 2 $\times$ 2 symmetric matrix M:

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} \tag{2.17}$$

For small shifts, E can now be written as the matrix multiplication:

$$E(x,y) = (x,y)M(x,y)^T \tag{2.18}$$

The eigenvalues of M, α and β, are proportional to the principal curvatures. Using them, it is possible to form a rotationally invariant representation of M. The result is one of three cases:

- If both eigenvalues are small, the intensity is approximately constant
- If one eigenvalue is high and one is low, the functions is ridge-shaped in this region. This indicates and edge.
- If both eigenvalues are high, the function has a peak. This indicates a corner.

With the determinant of M being $\alpha\beta$ and the trace $\alpha+\beta$, a final score R used to evaluate the result of the algorithm is given by:

$$R = det(M) - k(trace(M))^2 \tag{2.19}$$

Here, k is a free parameter. When the magnitude of R is small, the region is flat, when R is negative the region includes an edge, and when R is large the region is a corner. The balance between these states is given by the parameter k.

This concludes the Harris corner detection algorithm, which can in fact detect both edges and corners [17]. This algorithm also has a highly efficient implementation included in the OpenCV library.

2.10 State of the art

In this section, today's best performing models in different specialties are presented.

2.10.1 Classification

Classification is a typical machine learning task that has been steadily developed for decades. It has in recent years however taken large steps because of the rapid development of deep machine learning. The best model structure to date is the so called residual network, which is presented below.

2.10.1.1 Residual Networks

The depth of a neural network has been shown to be of crucial importance. As larger and deeper models are being developed and also have started converging, a so called degradation problem has been exposed. As the network becomes deeper, its accuracy tends to saturate and degrade rapidly. Surprisingly, this is not caused by overfitting, and adding more layers to a suitably deep model actually increases the training error in many cases. Since these layers could be modelled as identity mappings, the resulting training error should be no higher than that of the original model. This phenomena suggests that solvers could be struggling when approximating identity mapping using multiple non-linear layers [19].

In order to tackle this issue, scientists from Microsoft research have developed a so called deep residual learning framework [19]. Instead of hoping that a few stacked layers will successfully get the underlying mapping, this framework forces them to explicitly fit a residual mapping. This approach is supposed to be easier than matching the original unreferenced mapping. This concept is realized by using shortcuts in the network that represent an identity mapping. These shortcuts are made around network building blocks. The input to the segment of stacked layers is added to its output, thereby creating a residual mapping [19]. An illustration of this can be seen in Figure 2.1. The authors have then created an ensemble of models making use of this framework. They are all trained end to end on the ILSVRC dataset using stochastic gradient descent, backpropagation and Caffe [19].

During the training of these models, the developers made three important observations [19] :

- Models based on deeper networks tend to produce lower training errors, which also is generalizable to validation. This insinuates that the degradation problem has been addressed.

- The top-1 classification error (ratio of cases where the correct class does not get the highest score) is reduced compared to state of the art models, which shows that residual learning is effective on deep systems.
- For smaller depth, the results of the models trained show results comparable to those from a plain model (without residual learning). However, the convergence of the residual network is significantly faster!

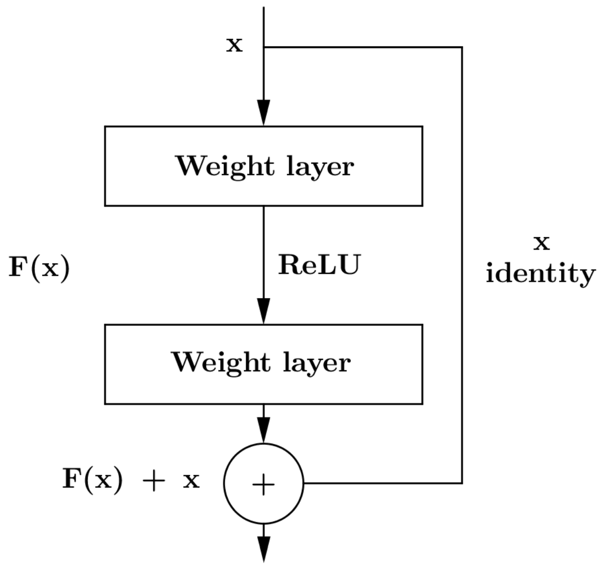


Figure 2.1: An example of a residual mapping. The input x is fed into the network building block, in this case consisting of two layers equipped with weights (e.g. convolutional or fully connected) with an intermediate ReLU-layer. The mapping performed by this block is denoted $F(x)$. The input is also fed via a shortcut around the block, resulting in an identity mapping. The output of of the block and the shortcut are added up to $F(x) + x$, creating the sought residual mapping.

The 152 layer residual net developed in [19] is the deepest model ever to be trained on and applied to ImageNet. Still, due to the new framework its complexity is lower than for the well-established VGG16/19 networks [34]. It performs a 4.49% top-5 error (ratio of cases where the score of the correct class is not in the top 5) on the ImageNet classfication set, which outperforms all previous models. An ensemble of six residual models, including two 152 layer deep ones, has managed to perform a top-5 error of 3.57%. This ensemble won first place in the ILSVRC 15 classification competition. Due to its great generalization, the Microsoft Research team also ended up winning ImageNet detection and ImageNet localization using models based on residual nets [19].

2.10.2 Localization

The object localization task has been considered very difficult throughout the years, but recently even this field has seen groundbreaking development. Today, even real-time applications are considered and developed. The latest and best performing concept so far is the Faster R-CNN framework, presented below.

2.10.2.1 Faster R-CNN

Region-based convolutional neural networks (R-CNN) is today the number one choice in object detection models. They were originally computationally very expensive, but due to efficiency improvements such as convolutional implementations being shared among object region proposals, this has recently changed. A late and established implementation of this concept is the Fast R-CNN framework [14]. This can be used for near real-time object localization using deep neural networks, but this is only valid if the time spent on producing region proposals is ignored. Producing region proposals is a tedious and inefficient process that usually is based on a CPU implementation. This causes problems and eliminates the benefits of a synchronized GPU architecture. A GPU implementation is the obvious solution, but even though a separate GPU implementation for detection and for producing region proposals would be an improvement it would still not make use of shared computation, which is more efficient [31].

In the new framework Faster R-CNN, developed by Microsoft Research, what is called Region Proposal Networks (RPN) is introduced. An RPN shares convolutional layers with the object detection network, thereby making computations much more efficient. It looks at both map positions and objectness scores to create multiple region proposals that are of both varying scales and aspect ratios. In order to train this integrated all-GPU-model efficiently, a new training scheme that alternates between fine-tuning the region proposal part of the model and the object detection part has been developed. The result is a unified network that converges nicely. Faster R-CNN, which consists of RPN combined with Fast R-CNN, is considerably faster than for example the benchmark method Selective search [37] combined with Fast R-CNN. Its groundbreaking speed and high accuracy makes it the state of the art of object localization [31]. Another show of this is that RPN was used together with residual nets to win ImageNet localization challenge in ILSVRC 15 [19]. Fast R-CNN object detection, however, was not used, and the authors believe that using this could improve results even further.

3
Methods

In this chapter, the methodology and materials used in this thesis are presented. Also described are the structures of the produced models for classification and localization.

3.1 Hardware and software

The setup used to perform the training sessions included in this thesis was based on a Dell Precision Workstation T7500. When purchased, this was a power house made for simulations and graphics, and it still delivers the ability to perform many types of high-performance computing. Its specifications are presented in Table 3.1.

Name	Dell Precision Workstation T7500
CPU	Double Intel Xeon X5660 (12 cores) @ 2.80 GHz
GPU	Nvidia Quadro 4000
Memory	12 GB DRR3 RAM 1333 MHz
Storage	600 GB 15000 RPM
OS	Ubuntu 14.04 LTS 64-bit

Table 3.1: Specifications for the hardware setup used in this thesis.

All programming done to perform preprocessing of the images and control the training sessions was done in Python using the Anaconda Python distribution[1] including a large range of packages and modules. The used IDE (integrated development environment) was PyCharm[2]. The GPU was using the Nvidia 352.21 graphics driver and handled computations via CUDA release 7.5-18. Setting up, designing and training the neural network models was done with Caffe through its Python binding module: pycaffe. The graphs shown in chapter 4 were made using Matlab.

3.1.1 GPU

Because of the development of these models being based on GPU-computing for faster and more efficient training, the GPU played a very important role for the results of this thesis. The specifications of the GPU used in this work is presented in Table 3.2.

[1]`https://www.continuum.io/why-anaconda`
[2]`https://www.jetbrains.com/pycharm/`

Name	Nvidia Quadro 4000
Release year	2010
GPU clock	475 MHz
Memory	2 GB GDDR5
Memory clock	702 MHz
CUDA cores	256
CUDA CC*	2.0

Table 3.2: Specifications for the GPU used for performing computations in this thesis.[3]
*: CUDA Compute Capability, an index to measure the GPU:s computational performance.

It should be mentioned that the Quadro 4000 is not a GPU specialized for computation, but rather for CAD and 3D graphics. As shown in Table 3.2, it was released several years ago and it also has a CUDA Compute Capability (index used to grade computational performance) of 2.0, which today is relatively low.

3.2 Models

This section will explain the design and structure of the models designed for the different tasks. For each task, the actual Caffe model was specified in a Protocol Buffer file. This file determined which LMDB databases to load, the batch sizes (which are used to create the random subsets for stochastic gradient descent), all the layers and their parameters, and the different outputs. Since the same file was used for both training and validation, layers and features exclusive to one of the phases were specified as well (validation is called TEST in Caffe).

3.2.1 Classification

For the classification task, the model was designed to be a compact convolutional network that would be trained using knowledge distillation. By making use of the knowledge of a larger network, one could produce a compact model that then could be fine-tuned towards a certain task. The logits representing the cumbersome model were produced in advance by the ResNet-50 model, a large but not very complex residual network developed by He *et. al* [19]. This choice as the cumbersome model was made based on residual learning having the role of state-of-the-art, but also because its size offered reasonable computation time to produce the logits on the current hardware. An illustration of the knowledge distillation setup is shown in Figure 3.1.

[3]`http://www.nvidia.com/object/product-quadro-4000-us.html`

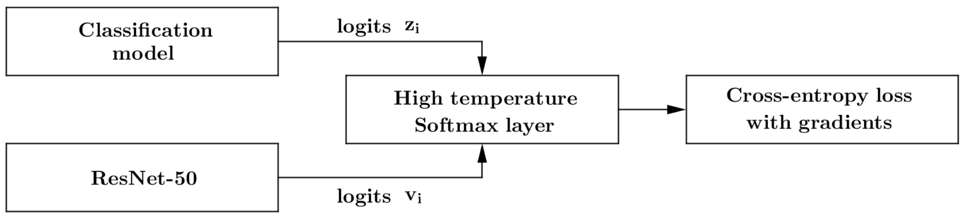

Classification model
logits z_i
High temperature Softmax layer
Cross-entropy loss with gradients
ResNet-50
logits v_i

Figure 3.1: The knowledge distillation setup. The ResNet-50 model is used to produce the logits v_i in advance. During the training process, logits z_i are produced by the designed classification model and are fed together with logits v_i into a high temperature Softmax layer. This layer then produces the cross-entropy loss with associated gradients used to train the model.

The images and labels, for both training and validation, were imported using a data layer. The logits were handled by a separate data layer.

During training, the batch size (for all data types) was 50, and for validation it was 20. The model was designed as a small standard convolutional network using convolutional, pooling, ReLU and fully connected layers to process the data. The number of outputs and parameter values were based on the model being of a sought after size but still have filters covering different scales and aspect ratios. The resulting structure of the model is given in Table 3.3.

Name	Type	Nbr. of outputs	Kernel size	Stride	Padding
conv1	Convolutional	256	11	5	1
relu1	ReLU	-	-	-	-
conv2	Convolutional	128	7	3	1
relu2	ReLU	-	-	-	-
conv3	Convolutional	128	5	1	1
relu3	ReLU	-	-	-	-
pool1	MAX Pooling	-	4	3	0
fc1	InnerProduct	1024	-	-	-
relu4	ReLU	-	-	-	-
drop1	Dropout	-	-	-	-
distil_logits	InnerProduct	1000	-	-	-

Table 3.3: Table showing all the intermediate layers and their parameters for the classification model.

The dropout layer is a simple function that, during training, sets each input to zero with a probability based on a user set parameter. Here, that parameter was 0.5. For the convolutional layers, the weights were initialized using a Gaussian distribution with mean zero and standard deviation of 0.01. The biases were initiated as zero. The same went for the final fully connected layer. The first fully connected layer, however, had its bias set to one and weights initiated with a Gaussian with standard deviation 0.005. For all layers included in the backpropagation, the bias experienced a double learning rate and no weight decay.

After creating the distilled logits, three different layers were used to create the output of the model:

- **hard_loss**: A Softmax layer that produces a multinomial logistic loss based on the image labels.
- **soft_loss**: A high temperature ($T = 6$) Softmax that takes both the distilled logits and the logits from ResNet-50, performs mean-value normalization on both, and then computes a cross-entropy loss according to equation (2.8). Based on the Caffe implementation by Noord [38].
- **accuracy**: Only included in validation, computes a ratio of correct predictions on the validation set.

This concludes the classification model. A graphical representation produced in Caffe can be seen in Figure B.1.

3.2.2 Localization

To take on the localization task, the model designed was something of a hybrid. To extract relevant features from an image, it made use of the Harris corner detector in OpenCV. This was implemented using the Python layer in Caffe, where custom Python code can be run as a layer in a Caffe model. The output from this layer was then sent through a small convolutional network in order to try and determine the bounding box. The size of the model was very much determined by the limited memory of the GPU.

Just as in the classification case, the images for both training and validation were imported into a data layer. Just like the logits, the bounding box values were imported into a separate data layer. The batch sizes were also the same: 50 for training and 20 for validation. The design of the network was, just like for the classification model, a standard convolutional network. A difference was that the localization model did not include a pooling layer. Since the model tries to work on singular pixel precision, downsampling the data is not an option. The final structure of the network processing the data is shown in Table 3.4.

For all the layers involved in the backpropagation, the weights were initialized according to a Gaussian distribution with mean zero and standard deviation 0.01. The biases were initialized with the value one, experienced double learning rate and no weight decay.

After producing the bounding box predictions, these were compared to the ground truth values. This was done using a Euclidean loss layer, that then produced the final output of the training and validation network: **loss_bbox**. The euclidean loss between the prediction- and target vector is computed using the expression:

$$\frac{1}{2N}\sum_{i=1}^{N}||x_i^1 - x_i^2||_2^2 \tag{3.1}$$

Here, N is the number of parameters defining the bounding box, x^1 is the prediction vector and x^2 is the target vector. This concludes the localization model. A graphical representation produced in Caffe can be seen in Figure B.2.

Name	Type	Nbr. of outputs	Kernel size	Stride	Padding
corner_harris	Python	-	-	-	-
conv1_loc	Convolutional	32	7	2	3
conv1_loc_relu	ReLU	-	-	-	-
conv2_loc	Convolutional	128	5	2	1
conv2_loc_relu	ReLU	-	-	-	-
conv3_loc	Convolutional	128	3	1	1
conv3_loc_relu	ReLU	-	-	-	-
conv4_loc	Convolutional	32	5	1	0
conv4_loc_relu	ReLU	-	-	-	-
bbox_pred	InnerProduct	4	-	-	-

Table 3.4: Table showing all the intermediate layers and their parameters for the localization model.

3.2.2.1 Harris corner detector from OpenCV

As mentioned above, the localization model used the Harris corner detector from OpenCV as a layer in the network to perform the first step of feature extraction. The Caffe Python layer used to implement this function was written as a regular Python file that imported the Caffe layer structure. It was specified in the file that this layer would not handle any backpropagation. The layer would thereby only be used in the forward computation and be excluded in the learning process.

When an image was provided to the layer, it was converted to gray scale by averaging over all channels. The resulting image was then sent to the OpenCV function, that would take in three other parameters: $blocksize$, which is the size of the neighborhood considered for detection, $ksize$, which is a parameter for the spatial derivative computation, and k, which is the free parameter in the score equation (2.19). These parameters were set to 2, 3 and 0.04 respectively. Once the scores were calculated for all positions in the image, the output of the layer (which was of the same dimensions as the gray scale image) was set to one for all scores above 1% of the maximum score, and zero otherwise. This way, corners in the original image were highlighted for the remaining parts of the model. An example showing the method applied to an image from the training set can be seen in Figure 3.2.

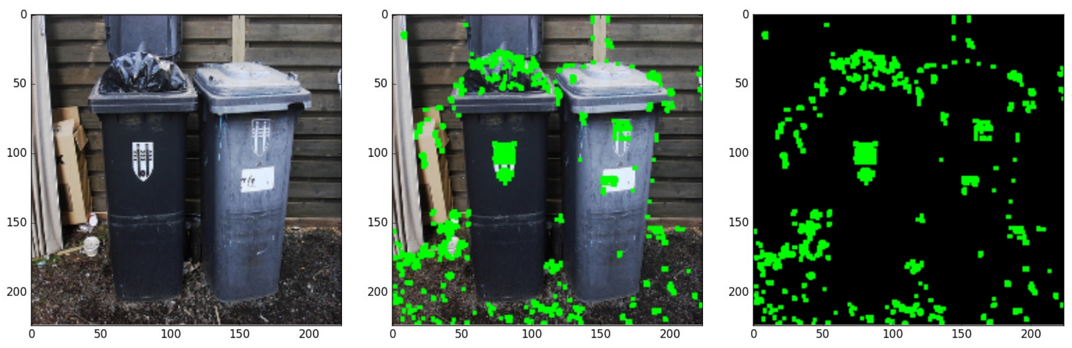


Figure 3.2: The implemented Harris corner detector applied to an image [11] from the training set. The left figure is the original image resized to 224×224. In the middle figure, the image has been preprocessed according to Section 3.3.1.1 and then fed through the corner detector. The output is then presented on top the original image, with green indicating found corners. The right figure shows the actual output, where one-values are shown in green and zero-values shown in black.

3.3 Dataset

The ImageNet database includes a synset named *Ashcan, trash can, garbage can, wastebin, ash bin, ash-bin, ashbin, dustbin, trash barrel, trash bin* that contains a considerable amount of objects interesting to the purpose of this thesis amongst its 1315 images. This synset is also included in the ILSVRC data set for classification and localization (CLS-LOC), which resulted in the ILSVRC data being used as training and validation data for the models in this thesis. The number of images from the data set used are shown in Table 3.5.

ILSVRC	**Training**	**Validation**
Classification	1281167	21980
Localization	544546	21980

Table 3.5: The number of images from the ILSVRC dataset used. For both classification and localization, there was a training set and a validation set. The images were of varying resolutions, and the ones in the localization set all included one bounding box.

There were several reasons to why the numbers differed from the ones presented in Section 2.6. First of all, the training set for the localization task had new images added to it recently, hence the increased number. Furthermore, the size of the validation set (which was the same for both tasks) was unfortunately decreased because of a faulty download resulting in corrupt files (from 50000 to 21980 images). However, the remaining amount was in the same order of size, close to evenly divided between classes, and its size could be beneficial (from a training time perspective) when working with limited hardware performance.

All images involved were JPEG color images (RGB) with varying resolutions. The constellation of each image focuses on one specific object and was only labeled with one class. All the images in the localization training set were also included in the classification training set, but they also included one bounding box. The validation set was the same for both tasks and all images in the set included a bounding box. For the training sets, the label was given in the file name. For the validation set and the localization task, each image came with a XML-file containing the label and the bounding box given as the four values x_{min}, y_{min}, x_{max} and y_{max}.

3.3.1 Preprocessing

In order for Caffe to achieve the best possible IO performance when going through large batches of images (or other data), a series of preprocessing actions had to be applied to the datasets used. Most of the adjustments were made using the transformer module included in Caffe, and the data was then packaged using the LMDB format.

3.3.1.1 Classification

For the classification task, images, their labels and their logits produced by the ResNet-50 model had to be prepared. Starting with the images, they were first resized to a resolution of 224 $\times$ 224. Next step was to transpose them, so that the first dimension of the array representing the image became the number of channels. Regarding colors, the channels were switched to BGR (Blue-Green-Red) instead of the standard RGB. This was because of Caffe preferring to handle images with this arrangement. The color channels were then rescaled from the zero to one scale to zero to 255. The ImageNet-mean for each channel (included in the Caffe distribution) was then subtracted in order to make the color channels zero meaned. Once all of the training images had been randomly shuffled (the validation images already were) to make sure different classes are not segregated, the images were now ready to be entered into the LMDB database.

The database was built around objects called datums, which have their definition included in Caffe. Data was entered into a key-value system, where each datum was given an index. For each datum, the image data, the number of channels, the height, the width and the label were entered separately. Once all datum objects were saved into the database, the LMDB environment was closed and saved. The same procedure was carried out for both the training set and the validation set.

For the logits, a special version of the ResNet-50 model was created where the final smoothening Softmax-layer was removed. The new model output was then the logits coming directly from a fully connected network. In order to create these, the model was iterated through the newly created LMDB databases containing all the training and validation images. Each image was passed through the network to create a vector with 1000 logits. Once the data was created, the logits were stored in the same kind of datum objects as for the images, but with some modifications. The number of channels was set to 1000, the height and width both set to one, and instead of using the standard image data field the float data field was used to store the logits. The datum was then inserted in a new LMDB database created specifically for the logits, where it was given the same index as its corresponding image in the main LMDB. The same procedure was carried out for both the training images and the validation images.

3.3.1.2 Localization

The images from the complete (shuffled) ILSVRC CLS-LOC training set that should also be included in the localization training set were determined by checking the existence of a corresponding XML-file to each image. Once the set of images to be used was known, the image preprocessing happened in the same way as for the classification training set described above. The validation set is the same for both tasks, hence the LMDB database created for the classification task could be reused.

What separated the data preparation for the localization task were the bounding boxes. As mentioned they were stored in XML-files as four variables: x_{min}, y_{min}, x_{max} and y_{max}. For each image, these four values had to be rescaled according to the new size of the images: 224×224. Since the XML-file also included values describing the original image format, this action could be easily performed using Python. Once these values were computed, they were inserted into separate LMDB databases, one for training and one for validation. The data storage was formatted in the same way as for the logits in the classification task using the float data field in the datum object. The indices used for keys in this database of course matched the ones used for the image LMDB:s.

3.4 Training process

The training of the models described above was carried out using the Caffe SGDSolver module (SGD stands for stochastic gradient descent). All the parameters of the training were specified in a Protocol Buffer file. Once the training was started, the solver loaded both the training- and validation network and calculated the amount of memory needed for the computations. It then started training the model according to the specifications. The solver was set to create a snapshot of both the weights and the solver state every 5000 iterations (one iteration being the processing of one batch). All the parameter settings for the solver can be seen in Table 3.6.

Almost all of these parameter values were standard for the Caffe SGDSolver. The learning rate, however, was decreased from 0.01 to 0.001 in order to offer better convergence. Both when training the classification and the localization model, when to stop the training was given by when the validation error started increasing (which indicates overfitting) or when the loss value stagnated. The computation time was also a limiting factor.

Test iterations	1099
Test interval	1000
Base learning rate	0.001
Learning rate policy	step
γ	0.1
Stepsize	100000
Momentum	0.9
Weight decay	0.0005

Table 3.6: Table showing the used solver parameters. Test iterations is the number of batches used during validation ($1099 * 20 = 21980$). The Test interval is how often (measured in iterations) the solver performs validation. The learning rate policy being set to step means that every Stepsize iterations, the learning rate (which from the start is set to Base learning rate) is multiplied by γ. The momentum and weight decay parameters are the same as μ and λ in equation (2.6) and are meant to stabilize the training procedure.

4 Results

In this chapter, the results of the training processes executed using the models in the previous chapter are presented.

4.1 Classification

The training of the classification model was first carried out using a combination of the hard loss and the soft loss, with Caffe computing gradients based on both. Unfortunately, the results were far from satisfactory, with the soft loss diverging into non-numerical values after just a few iterations. Different learning rates were tested, all resulting in divergence.

The second approach tried was to use the soft loss only (pure knowledge distillation) as pre-training in order to smoothen out the network output, before fine-tuning it using the hard loss training. Even though this approach is supported by previous work, again the soft loss diverged heavily and early. Also with this approach, the loss ended up taking on non-numerical values, even when the learning rate was decreased.

The final approach was to start the training using only hard targets, basically a typical machine learning training session using images and labels. This would move the model closer to a proper classifier in weight space (at least closer than the randomly initiated), and thereby allow the soft loss to generalize the results without divergence. The hard loss training for the validation set can be seen in Figure 4.1. Its accuracy on the validation set can be seen in Figure 4.2. The reason to why only the validation values are shown is because they are based on the entire validation set. The training values in each iteration are based only on the current batch of data (goes for both the classification and the localization task).

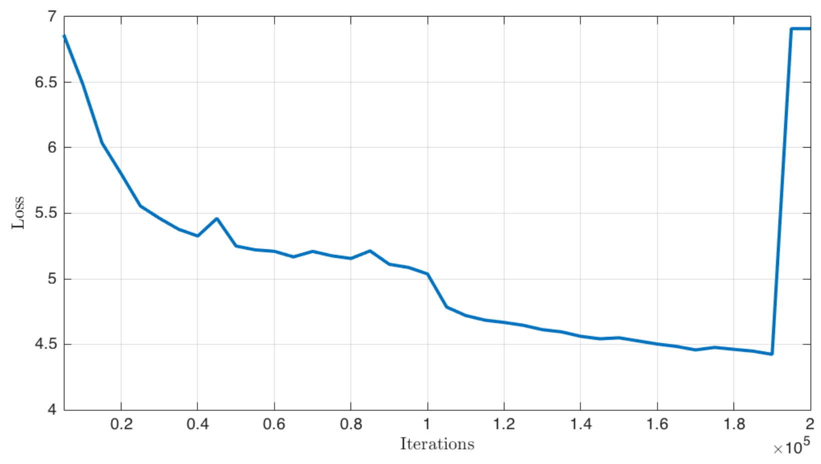


Figure 4.1: The label-based loss for the validation set during the training of the classification model as a function of the number of iterations. The loss can be seen to decrease for about 190000 iterations, before it rapidly returns to the initial level. The plot was produced using Matlab.

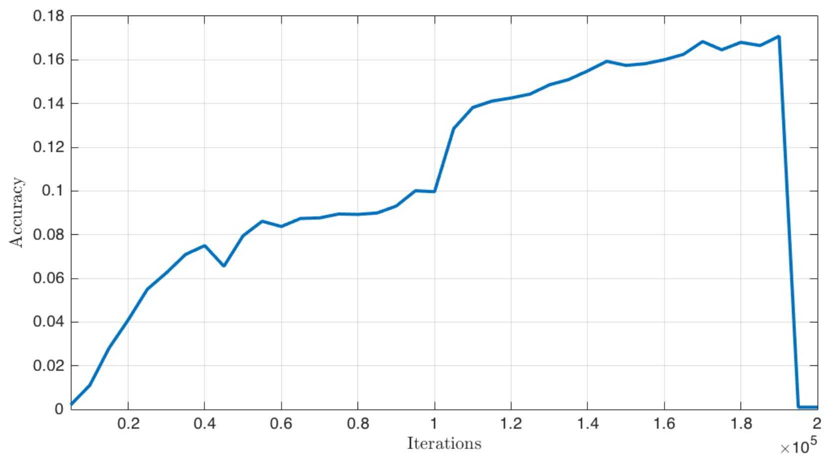


Figure 4.2: The accuracy for the validation set during the training of the classification model as a function of the number of iterations. The accuracy can be seen to increase for about 190000 iterations, before it rapidly returns to the initial level. The plot was produced using Matlab.

The training was allowed to run for 200000 iterations, which with a batch size of 50 corresponds to about 7.8 epochs of the complete training set. This process took about four days to run. As can be seen in Figure 4.1, after about 190000 iterations, the loss jumped back to a value close to the initial one. This can be the result of for example overfitting. Either way, the last 10000 iterations are not interesting when moving on to the next stage in the training process.

The next step was to keep training the model (based on the snapshot from 190000 iterations), now using soft targets to smoothen out its mapping. Unfortunately, again, training with the soft loss resulted in almost immediate divergence, even for lower learning rates. Throughout the try to train this compact model to perform above the level proportionate to its size, the concept of knowledge distillation has not worked at all.

4.2 Localization

The results from the training of the OpenCV-hybrid based localization model are shown in Figure 4.3, which is a plot of the validation loss against the number of iterations.

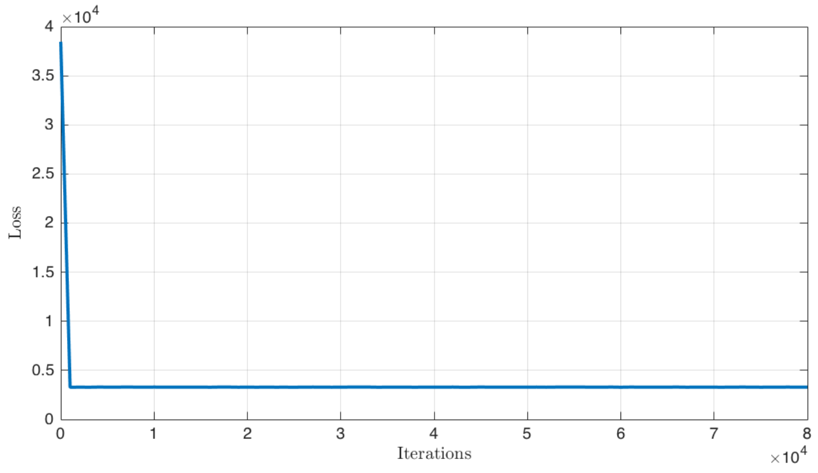


Figure 4.3: The bounding box-based Euclidean loss for the validation set during the training of the localization model as a function of the number of iterations. The loss can be seen to start at a higher value before suddenly dropping. The loss then stabilizes and remains about the same for the entire training session.

As can be seen in the figure, the model exhibits rapid convergence to a steady value that was held throughout the training. The training is allowed to run for 80000 iterations, which with a batch size of 50 images corresponds to about 7.3 epochs of the localization training set. This took about three days to run. Despite the fast convergence, the loss value is still rather high considering the nature of the Euclidean loss. When applied to images from the training set, the model seems to produce the same bounding box for each image (after being cut to fit inside the image): positioned at (41,43) with width 182 and height 180. Lowering the learning rate during training to 0.0001 does unfortunately result in the same behavior. How the model produces bounding boxes on the actual training set can be seen for a few example images in Figure 4.4.

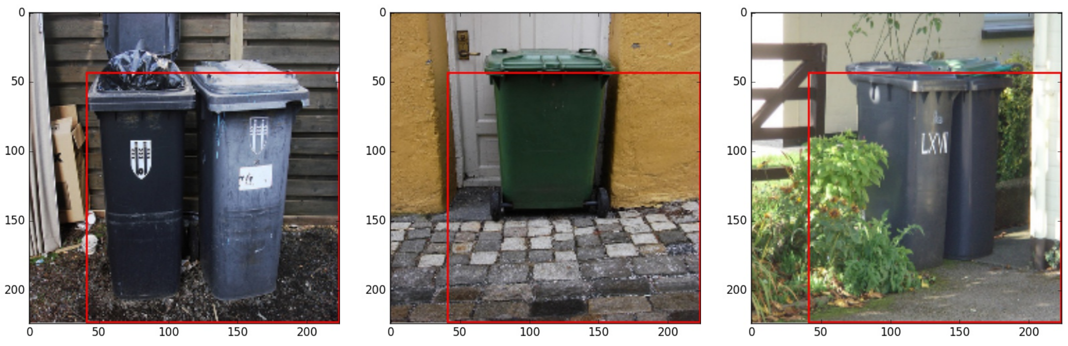


Figure 4.4: Examples of the bounding box-producing localization model applied to images [11–13] from the training set. In the figure, the images are resized to 224 × 224. The model can be seen to produce the same red box for each image (at (41,43) with width 182 and height 180), which indicates that the model performance stagnated during training.

5
Discussion

This chapter reflects on the results from this thesis, discusses its relevance and suggests future course of action.

5.1 General considerations

Apart from the models developed, there are some aspects regarding the project as a whole that are worth mentioning before moving on to the results.

5.1.1 Steep learning curve

As this thesis has developed, it has become clear that machine learning and its applications is a field that has a steep learning curve with a high threshold before actually achieving relevant progress. The highly theoretical foundation of the concept makes it some what unattainable, and it is easy to fall into a false sense of understanding. Machine learning is by nature a black box-technique with very little insight, and in order to make the most use of it, it is important to at least know what the different possibilities and strategies are.

With high-level science comes high-level tools, and considering the amount of research and follow up needed to successfully set up the hardware and software needed to perform this kind of research, the original time frame of this thesis was some what skewed. GPU computing is still in heavy development and the community around it is still of a very academic nature. Hence, full-scaled out-of-the-box solutions are few. Taking the time to research these tools, however, deepens the understanding of machine learning itself heavily.

5.1.2 Hardware

The amount of research going into the field of machine learning in recent years have been substantial, and with that comes a steadily increasing demand on powerful and reliable hardware. This goes especially for GPU:s, which today are very important for fast parallelized computing. Going into this thesis, the impression was that a under-dimensioned GPU would mainly slow down, for example, a training process. As it turns out, it results in even larger set backs than that. The computation time is definitely increased, but an aged hardware architecture or, most importantly, an undersized memory unit severely limits the opportunities of producing relevant results. For example, the GPU used in this project had a CUDA Compute Capability of 2.0, where as Caffe is developed on systems with 3.0 or higher. The result is the older GPU showing flaws not even known to BVLC (developers of Caffe).

Even so, for a thesis like this where the GPU is something of a weak spot, it still performs many times better than the high-level CPU available. That is an important fact that emphasizes even more the importance of GPU computing. This also hits both ways, since in most cases the hardware is not powerful enough to support sophisticated models in real-time applications, but neither light enough to be used in low-scale hardware applications.

5.2 Evaluation of the models

In this section, the models for classification and localization produced are discussed. Their design, results and academic relevance are covered.

5.2.1 Classification

When designing the classification model, it was known that the model itself was not supposed to be a standalone model to handle the 1000 class ILSVRC dataset. The idea was that the concept of knowledge distillation would produce a model that performed beyond the expectations of a model that size, all to create something applicable to low-scale performance setups. Unfortunately, the distilling process ended up producing no relevant results due to heavy divergence. Exactly what this was caused by is hard to pin-point. Usually, the training time provided to the system is a matter to discuss, since there is often a possibility that the used loss function may experience another drop within a reasonable amount of iterations. However, since the soft target loss diverged into non-numerical values, there was never any chance that the tide would turn and the results would come around. The training was lost at an early stage, and there must have been a structural failure that did not allow the system to adapt to the provided logits.

A probable reason for the poor performance of the knowledge distillation would be the choice of the cumbersome model providing the knowledge: the residual model ResNet-50. In the literature that was studied regarding knowledge distillation, any explicit requirements on the relationship between the structure of the cumbersome model and the distilled model were never provided. Hence, the residual structure was considered merely because of it being considered state of the art. It is clear that the architecture of a residual model, with its typical identity mappings, is very distinct. It is likely that the model to be distilled basically could not pick up on the mapping from the cumbersome model, and therefore never learned from the logits. Structures with a stronger resemblance with residual networks were tried for the distilled model during development, but did not perform differently.

Once it was clear that training the model on a combination of the two losses or pre-training it using the logits did not work, there were few options. The available data was not task-specific enough to fine-tune an existing but generalized model, which could have been a good way to proceed. The ILSVRC set was chosen because it contained one class out of 1000 that was at least task-relevant, and it would hopefully produce a generalized model that could be fine-tuned at a later stage. Furthermore, the available hardware was not powerful enough to be able to train a model from scratch (on the entire ILSVRC set) that would be academically relevant and better than any freely available state-of-the-art model. The choice was therefore made to move on to pure hard loss training of the distilled model. Again, the model was never expected to perform well on this training alone, and when the soft target training still would not pick up even after the model has been further specialized, it was clear that no relevant results would come out of it. However, it is in itself quite extraordinary that a model this small would be able to (on a data set with almost 22000 images) predict the correct class (out of 1000) one sixth of the time.

5.2.2 Localization

The localization model showed great convergence at the early stages of training as its training loss dropped several orders of magnitude. The loss did however stabilize at a rather high value, and with a batch-average loss over 3000, the average error (according to equation (3.1)) for one value in the bounding box prediction would be about 80 pixels. Considering that the side of the resized image used is 224, this value is very large and results in poor predictions. It was also shown when applied to images from the training set that the model was producing the same output regardless of input. It is clear that the training early converged into a local minimum resulting in very poor accuracy, which surely has several reasons. One design choice that may have caused the convergence to stagnate would be the use of a Euclidean loss. Due to its quadratic properties, the loss minimization is done on the wrong scale. It is unable to perform correct adjustments on the scale of singular pixels, which is what is needed to compute correct bounding boxes. There is also the fact that Euclidean loss is mirrored in sign, meaning that negative pixel positions could be accepted, and worse case, preferred during training.

Another important factor for the performance of the localization model is of course the use of OpenCV and the Harris corner detector. The idea is that information would be extracted and presented in an efficient manner, so that the machine learning part can focus on the positioning of the bounding box. An apparent problem is that the OpenCV function could be removing information useful to the model. Is its net contribution really positive to the outcome? This is highly dependent on the used parameters. The free variable, k, determining the score has a major impact on how the model treats the images. The same goes for the parameters determining how to compute derivatives and the spatial distribution. Also, when setting the output of the custom Python layer, it is all based on the score corresponding to a certain ratio of the max score. This ratio is also a crucial parameter, and this structure means that only corners are considered, not edges. In order to determine the impact of all these parameters and design choices, a proper mapping of the parameter space would have been necessary. With a larger time frame, this kind of investigation would have been prioritized.

An important aspect of the use of OpenCV is that the Python layer reduces the data amount, from input to output, to only a third. The Python layer therefore takes up a lot of the available GPU memory, heavily reducing the amount of memory available to the machine learning model. It is very likely that the main convergence problem for the model is its very limited size. It should be noted that is was not given that many iterations to train, but considering how quickly the initial convergence stagnates, it is likely that the model has reached its full potential. A larger graphics memory would have allowed a larger machine learning implementation, and this kind of hybrid solution may not have been optimal for this hardware setup.

5.3 Future work

This section addresses a proper course of action when developing the knowledge within the topics included in this thesis. This section is one of the main contributions from this thesis to the work at Volvo ATR, since it approaches the topic with a developing actor in mind.

5.3.1 Data collection

In order to make machine learning a more easily applicable technique in the future, extensive data collection has to be integrated into the daily work of both industry and academia. To efficiently design and train models made for certain tasks, large amounts of task-specific but yet varied data has to be provided. One of the major issues is still, however, that as much of the data as possible has to be annotated. Supervised learning is still a very common approach when training for example a neural network, and then labeling data properly is key. Manually labeling data as it is collected is tedious, and a proper solution to this issue is yet to be found. A first step are the crowdsourcing solutions offered, that aim to streamline large annotation projects for the benefit of machine learning development. The Amazon Mechanical Turk service has been shown to be a viable alternative for data collection [29].

Public data sets of different kinds, such as the ILSVRC data set used in this thesis, are still popping up and can be used to create stable generalized models. These are still of great importance for the development of the field. When looking at designing models for industrial deployment, however, the need for data tailored for the application at hand is obvious.

5.3.2 Addressing the hardware issue

To be competitive and produce valuable results using machine learning, investing in hardware is key. As this field has developed, GPU:s tailored for computation have emerged, and their prices are steadily dropping. Still, the more graphics oriented products that are much cheaper can perform rather well for most applications. Depending on what is being developed, different hardware might be needed during training and deployment. GPU:s still need a lot of power and ventilation to function, so it is still complicated to make use of these in, for example, embedded systems applications.

If equipping with state-of-the-art GPU:s for in-house training is not an option, there is a growing industry around third-party actors offering GPU computing for rent. Companies like Amazon engage in these kinds of services, which for many developers might be the key to put idea into practice. This way, the opportunity to create and train highly sophisticated models reaches out to more actors which is of benefit for the entire field.

5.3.3 Model development

Regarding the models produced in this report, there are major flaws that need to be adjusted before getting even close to actual deployment. Even though these models perform quite poorly, this thesis makes out the first step for establishment in the world of machine learning. Starting from zero, the entire set up of both hardware and software has been configured and tested, and actual training processes have been carried out successfully using the state-of-the-art tool Caffe. Knowledge distillation has been shown to work very well under the right circumstances in previous work, and a larger investigation of the parameter space and different structures has to be carried out. OpenCV offers well-designed and efficient models for feature detection, and with the right hardware, these could prove valuable in a compact machine learning model.

One of the keys to successfully develop new models is balance. Balance between using existing material and starting from zero. Balance between fine-tuning pre-trained nets and creating a generalized large-scale model. Balance between outsourcing computation and investing in the right hardware. Machine learning has created an open community spanning both academia and industry, and by finding the right balance, many actors have the opportunity to create accurate and fast models ready for deployment.

6 Conclusion

After establishing a solid theoretical foundation and setting up the hardware available, two models were designed and trained. When applied, both models performed poorly and did not produce academically relevant results for different reasons.

For the classification model, the knowledge distillation concept did not perform as expected. A combination of a possibly poor choice of cumbersome model combined with limited hardware and training time resulted in the model only being trained on traditional labels. Knowledge distillation has, however, been proven to be effective, and could under different circumstances be a good alternative for producing a model for limited hardware-applications.

In the case of localization, the hybrid concept making use of the Harris corner detector from OpenCV did not perform as hoped. The choice of loss function combined with the machine learning part of the model being of limited size resulted in early convergence into a local minimum, which in practice meant producing the same output for all input. However, the hardware was an obvious issue, and it is not possible to rule out the concept based on the results in this thesis.

Regarding future work, finding the balance within the two key factors of machine learning, data and processing power, is crucial in order to be competitive and produce models ready for deployment in the future.

Bibliography

[1] Caffe model zoo. `http://caffe.berkeleyvision.org/model_zoo.html`, 2016.

[2] Imagenet. `http://image-net.org/index`, 2016.

[3] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.

[4] Gary Bradski et al. The opencv library. *Doctor Dobbs Journal*, 25(11):120–126, 2000.

[5] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[6] Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541. ACM, 2006.

[7] Real Carbonneau, Kevin Laframboise, and Rustam Vahidov. Application of machine learning techniques for supply chain demand forecasting. *European Journal of Operational Research*, 184(3):1140–1154, 2008.

[8] Howard Chu. Mdb: A memory-mapped database and backend for openldap. *LDAP'11*, 2011.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[10] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[11] Flickr. `http://farm2.static.flickr.com/1281/908219757_008953b095.jpg`, 2016.

[12] Flickr. `http://farm1.static.flickr.com/154/409264194_c8baa6ecf4.jpg`, 2016.

[13] Flickr. `http://farm3.static.flickr.com/2240/1616297688_78c5ed9be4.jpg`, 2016.

[14] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

[15] Volvo Group. Drone to help refuse-collecting robot find refuse bins.

[16] Volvo Group. Refuse truck driver is supported by robot.

[17] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Citeseer, 1988.

[18] Simon Haykin and Neural Network. A comprehensive foundation. *Neural Networks*, 2(2004), 2004.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[20] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

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

[22] Andrej Karpathy. Convolutional neural networks for visual recognition. Unversity Lecture, 2016.

[23] David Kirk et al. Nvidia cuda software and gpu parallel computing architecture. In *ISMM*, volume 7, pages 103–104, 2007.

[24] Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 34(4-5):705–724, 2015.

[25] Chun Hung Li and CK Lee. Minimum cross entropy thresholding. *Pattern Recognition*, 26(4):617–625, 1993.

[26] Donald Michie, David J Spiegelhalter, and Charles C Taylor. Machine learning, neural and statistical classification. 1994.

[27] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[28] J Moody, S Hanson, Anders Krogh, and John A Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4:950–957, 1995.

[29] Gabriele Paolacci, Jesse Chandler, and Panagiotis G Ipeirotis. Running experiments on amazon mechanical turk. *Judgment and Decision making*, 5(5):411–419, 2010.

[30] Andry Maykol Pinto, Luís F Rocha, and A Paulo Moreira. Object recognition using laser range finder and machine learning techniques. *Robotics and Computer-Integrated Manufacturing*, 29(1):12–22, 2013.

[31] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[32] Martin Riedmiller. Advanced supervised learning in multi-layer perceptrons—from backpropagation to adaptive learning algorithms. *Computer Standards & Interfaces*, 16(3):265–278, 1994.

[33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[34] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[35] Volvo Trucks. About us. `http://www.volvotrucks.com/trucks/global/en-gb/aboutus/Pages/about_volvo_trucks.aspx`, 2012.

[36] Volvo Trucks. How volvo dynamic steering made world-first stunt possible. `http://www.volvotrucks.com/trucks/global/en-gb/newsmedia/pressreleases/Pages/pressreleases.aspx?pubId=16768`, 2013.

[37] Jasper RR Uijlings, Koen EA van de Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[38] Nanne van Noord. Caffe-kdistil. `https://github.com/Nanne/caffe.git`, 2015.

[39] Dong Wang, Chao Liu, Zhiyuan Tang, Zhiyong Zhang, and Mengyuan Zhao. Recurrent neural network training with dark knowledge transfer. *arXiv preprint arXiv:1505.04630*, 2015.

A

Cross-entropy gradient derivation

In the process of knowledge distillation, the cross-entropy loss is computed based on two probability distributions: probability q_i for class i for the distilled model (based on logits z_i), and probability p_i for class i for the cumbersome model (based on logits v_i). The probabilities are produced using the Softmax function on the logits:

$$q_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}, \quad p_i = \frac{e^{v_i/T}}{\sum_j e^{v_j/T}} \tag{A.1}$$

Here, T is the temperature of the Softmax function. Since p and q are probability distributions, it is given that $\sum_i p_i = \sum_i q_i = 1$. The cross-entropy loss, measuring how well the distribution q corresponds to p is calculated as:

$$C(p,q) = \sum_{i=1}^{N} p_i \cdot log\left(\frac{p_i}{q_i}\right) \tag{A.2}$$

In order to perform knowledge distillation, the gradient of the cross-entropy loss with respect to logit z_i of the distilled model. This can be derived by first re-writing the cross-entropy expression:

$$C(p,q) = \sum_i p_i \cdot log\left(\frac{p_i}{q_i}\right) = \sum_i p_i\left(log(p_i) - log(q_i)\right) = \sum_i p_i log(p_i) - \sum_i p_i log(q_i) \tag{A.3}$$

Since p_i does not depend on z_i, the gradient with respect to z_i becomes:

$$\frac{\partial C}{\partial z_i} = \frac{\partial}{\partial z_i}\left(-\sum_j p_j log(q_j)\right) \tag{A.4}$$

Using the expression for q_j from (A.1), this can be written as:

$$\frac{\partial C}{\partial z_i} = \frac{\partial}{\partial z_i}\left(-\sum_j p_j log\left(\frac{e^{z_j/T}}{\sum_k e^{z_k/T}}\right)\right) \tag{A.5}$$

With the derivative being a linear operation, the term where $j = i$ can be treated separately. The terms $j \neq i$ will be handled first:

$$\begin{gathered}\frac{\partial}{\partial z_i}\left(-p_j log\left(\frac{e^{z_j/T}}{\sum_k e^{z_k/T}}\right)\right) = -p_j\frac{\sum_k e^{z_k/T}}{e^{z_j/T}}\frac{\partial}{\partial z_i}\left(\frac{e^{z_j/T}}{\sum_k e^{z_k/T}}\right) = \\ -p_j\left(\sum_k e^{z_k/T}\right)\frac{\partial}{\partial z_i}\left(\frac{1}{\sum_k e^{z_k/T}}\right) = -p_j\left(\sum_k e^{z_k/T}\right)\left(-\frac{e^{z_i/T}}{T\left(\sum_k e^{z_k/T}\right)^2}\right) = \\ \frac{p_j}{T}\left(\frac{e^{z_i/T}}{\sum_k e^{z_k/T}}\right) = \frac{p_j q_i}{T}\end{gathered} \tag{A.6}$$

The next step is to look at the term where $j = i$:

$$\begin{gathered}\frac{\partial}{\partial z_i}\left(-p_i log\left(\frac{e^{z_i/T}}{\sum_k e^{z_k/T}}\right)\right) = -p_i \frac{\sum_k e^{z_k/T}}{e^{z_i/T}} \frac{\partial}{\partial z_i}\left(\frac{e^{z_i/T}}{\sum_k e^{z_k/T}}\right) = \\ -p_j \frac{\sum_k e^{z_k/T}}{e^{z_i/T}}\left(\frac{e^{z_i/T}(\sum_{k\neq i} e^{z_k/T})}{T\left(\sum_k e^{z_k/T}\right)^2}\right) = -\frac{p_i}{T}\left(\frac{\sum_{k\neq i} e^{z_k/T}}{\sum_k e^{z_k/T}}\right) = -\frac{p_i}{T}\left(\sum_{k\neq i} q_k\right) = \\ -\frac{p_i}{T}(1-q_i)\end{gathered} \tag{A.7}$$

Now, the final gradient can be obtained by adding all these terms together:

$$\begin{aligned}\frac{\partial C}{\partial z_i} &= -\frac{p_i}{T}(1-q_i) + \sum_{k\neq i}\frac{p_k q_i}{T} = \frac{1}{T}(q_i p_i + q_i \sum_{k\neq i} p_k - p_i) = \\ &\frac{1}{T}(q_i(p_i + \sum_{k\neq i} p_k) - p_i) = \frac{1}{T}(q_i - p_i)\end{aligned} \tag{A.8}$$

Hence, the sought expression is derived.

B
Network graphs

This appendix includes the Caffe-generated graphical representations of the two models produced in this thesis.

B.1 Classification model

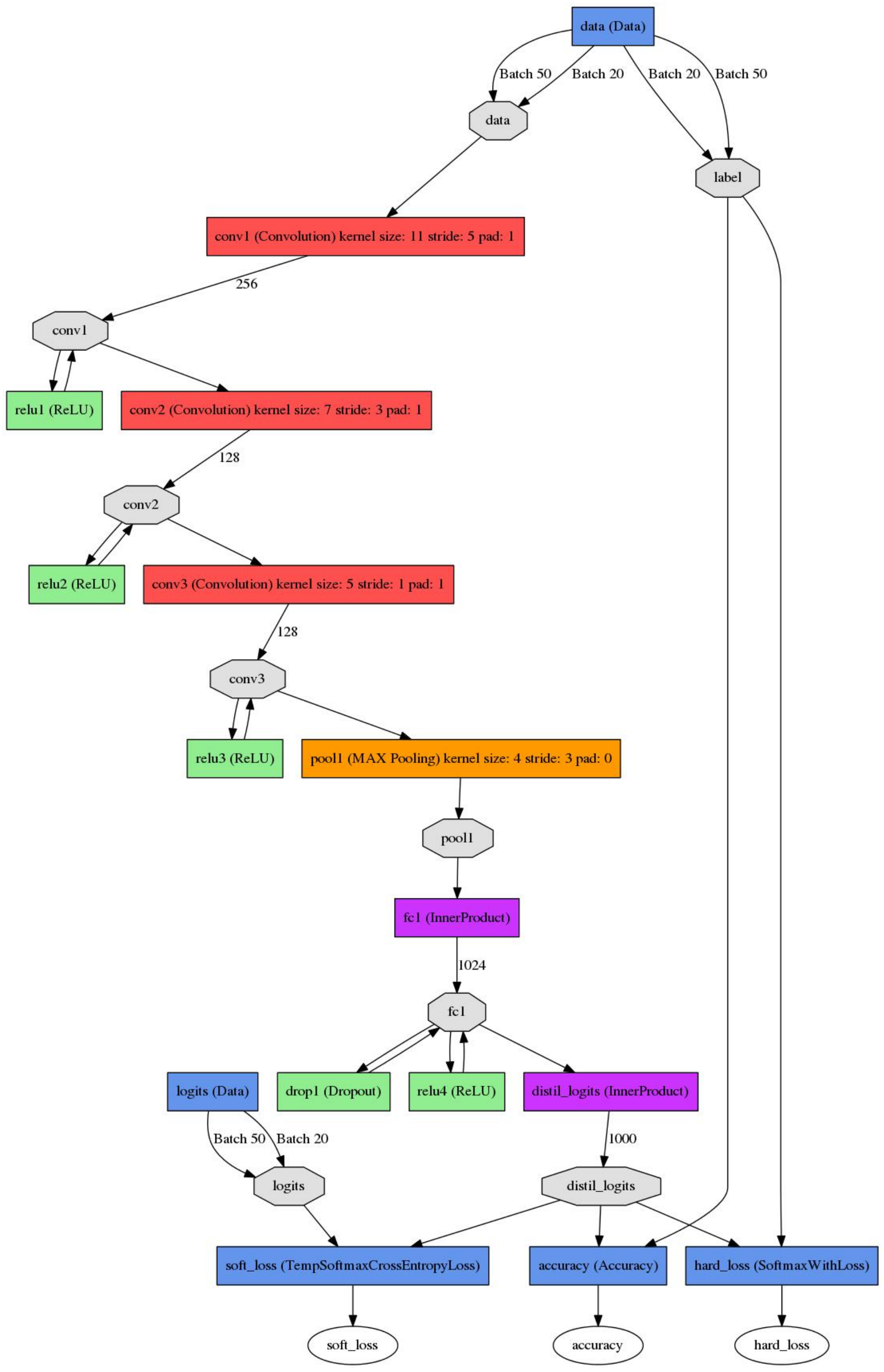

data (Data)
Batch 50
Batch 20
Batch 20
Batch 50
data
label
conv1 (Convolution) kernel size: 11 stride: 5 pad: 1
256
conv1
relu1 (ReLU)
conv2 (Convolution) kernel size: 7 stride: 3 pad: 1
128
conv2
relu2 (ReLU)
conv3 (Convolution) kernel size: 5 stride: 1 pad: 1
128
conv3
relu3 (ReLU)
pool1 (MAX Pooling) kernel size: 4 stride: 3 pad: 0
pool1
fc1 (InnerProduct)
1024
fc1
logits (Data)
drop1 (Dropout)
relu4 (ReLU)
distil_logits (InnerProduct)
Batch 50
Batch 20
1000
logits
distil_logits
soft_loss (TempSoftmaxCrossEntropyLoss)
accuracy (Accuracy)
hard_loss (SoftmaxWithLoss)
soft_loss
accuracy
hard_loss

Figure B.1: A graphical representation of the classification model. Created using Caffe.

B.2 Localization model

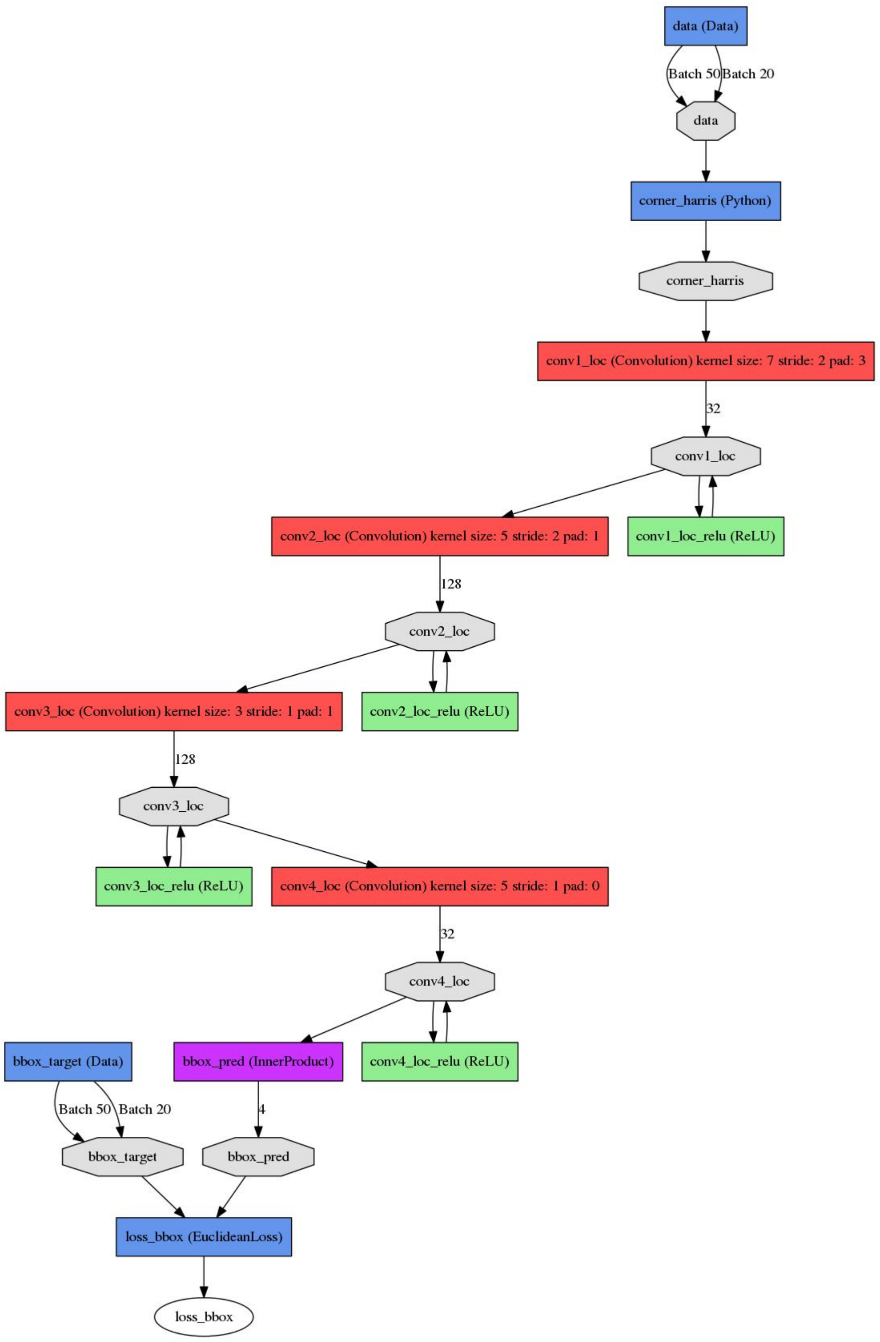


Figure B.2: A graphical representation of the localization model. Created using Caffe.