LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK
LEHR- UND FORSCHUNGSEINHEIT
FÜR DATENBANKSYSTEME

Masterarbeit
in Informatik

Semisupervized Object Detection in UAV Images

Michael Fromm

Aufgabensteller:	Prof. Dr. Matthias Schubert
Betreuer:	Evgeniy Faerman
Abgabedatum:	01.05.2018

Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 01.05.2018

.....
Michael Fromm

Abstract

This thesis investigates the effectiveness of machine learning algorithms for automatic detection of coniferous seedling data along Boreal seismic lines. In order to obtain a survival assessment and survey of the restoration process along these seismic lines, ground crews must undertake expeditions which are expensive, potentially hazardous and difficult to scale. Since the seismic lines cover a length of more than 10,000 km, an automated solution is necessary. The literature describes several machine learning applications using satellite data to extract information on topics such as forestation and expanse of deserts. In contrast, we conduct experiments on small sites on the basis of drone imagery. To this end we use algorithms from computer vision and apply them to the drone image data. We use convolutional neural networks as a feature extractor on the images. Subsequently, we train an object detector to spot the seedlings and annotate them with bounding boxes. In this work we evaluate the accuracy of modern object detectors such as Faster R-CNN with regard to remote sensing capacity of conifer seedlings. We further investigate the problem by doing several experiments which focus on the special environmental variables in nature, including seasons and flight height of the drone. These are necessary to understand what conditions are beneficial to support the machine learning process. Modern convolutional object detectors require huge amounts of data. Meanwhile, we conduct experiments on the amount of data needed to achieve high accuracy and we also investigate the influence of pretrained networks on the object detector. We further employ error analysis to understand how the object detector performs depending on the seedling size, to determine where further improvements are possible. Finally, we suggest further research options based on our findings.

Abstract

In dieser Arbeit untersuchen wir die Wirksamkeit von Machine Learning Algorithmen zur automatischen Erkennung von Nadelbäumsetzlingen entlang borealer seismischer Linien. Um eine Überlebensbeurteilung und eine Bestandsaufnahme des Sanierungsprozesses in den seismischen Linien zu erhalten, müssen Bodenmannschaften Expeditionen durchführen, die teuer, potentiell gefährlich und kaum skalierbar sind. Da alle seismischen Linien zusammen eine Länge von mehr als 10.000 km besitzen, ist eine automatisierte Lösung notwendig. Die Literatur beschreibt verschiedene Machine-Learning Lösungen bezüglich Satellitendaten, um Informationen wie etwa den Waldbestand oder Veränderungen in der Wüstenbildung festzustellen. Unsere Arbeit hingegen untersucht kleine Objekte auf der Basis von Drohnendaten. Um das zu erreichen, verwenden wir Algorithmen aus dem Computer Vision Bereich und wenden sie auf Bilddaten von Drohnen an. Wir verwenden neuronale Netze um die Attribute aus den Bilder zu extrahieren. Anschließend trainieren wir einen Objektdetektor, um die Setzlinge zu erkennen und mit Begrenzungskästchen zu versehen. In dieser Arbeit bewerten wir die Genauigkeit moderner Objektdetektoren wie Faster R-CNN auf der remote sensic Kapazität von Nadelbäumen. Wir untersuchen das Problem weiter, indem wir mehrere Experimente durchführen, die auf die speziellen Umgebungsvariablen in der Natur, wie Jahreszeiten und die Auflösungsabhängigkeit eingehen. Diese sind notwendig, um zu verstehen, welche Bedingungen notwendig sind, um den maschinellen Lernprozess zu unterstützen. Moderne neuronale Netze benötigen eine große Datenmenge, wir führen Experimente über die benötigte Anzahl an Daten durch, die für eine hohe Genauigkeit notwendig ist, und wir untersuchen den Einfluss welches das vortrainieren der neuronalen Netze auf den Objektdetektor hat. Weiterhin führen wir eine Fehleranalyse durch, um zu verstehen, wie sich der Objektdetektor in Abhängigkeit von der Setzlingsgröe verhält, um zu sehen, wo weitere Verbesserungen möglich sind. Am Ende erötern wir weitere zukünftige Forschungsmöglichkeiten.

Contents

Chapter 1

Introduction

Human intervention in natural habitats often has far-reaching consequences. The global energy industry and its land use has a huge impact. Canada's boreal forests face rapid land use change across the globe compared to other environments. Western Canada's boreal forest soil contains at least one-third of the worlds proven oil reserves [Heb17]. The oil and gas industry has cut more timber than the forest industry in order to develop energy infrastructure. The widespread land use changes have transformed Alberta's forests into an industrialized landscape with a large network of energy-related infrastructure sites, including roads, transmission lines, pipelines, seismic exploration lines and well sites. The woodland caribou, which is an umbrella species for the Boreal forest, depends on old growth coniferous forests and boreal peatland complexes. The huge network of roads and seismic exploration lines has increased encounter rates between wolves and caribous [BMF+13]. This has led to a situation in which 28 of 57 populations of boreal woodland caribou and 20 of 25 southern mountain populations are in decline. Scientists have recognized the land use changes as the reason for the decline of the caribou and the government is now trying to mitigate the damage and preserve the populations. The local authorities have released a restoration framework called LIRA, which aims to restore the seismic lines. The total length of old seismic lines to be restored exceeds 10,000 km. This massive distance is nearly impossible to monitor manually. In recent years, new technology such as unmanned aerial vehicles and improvements in camera sensors have made it possible to take high quality images of the seismic line restoration process. This thesis aims to show that it is possible to extract information from these images with machine learning algorithms. Information about the number, growth and location of planted seedlings is needed in order to guarantee the success of the restoration effort.

1.1 Structure of the thesis

In the second chapter of the thesis we review the concepts of convolutional neural networks (CNNs), including their different layers and characteristics. In particular, we examine two modern architectures of CNNs: Inception Net and ResNet. We then discuss how CNNs can be used as a feature extractor for object detectors. The fourth chapter puts forth the problem statement and introduces the dataset we used and problems that occurred during the work. In the fifth chapter we explain the experiment environment and provide the technical details of the software and hardware we used. The sixth chapter includes multiple experiments with different object detectors and techniques such as data augmentation. We also carried out experiments to gain an understanding of how the flight height of UAV and boreal seasons influence the results of the object detection task. In the penultimate chapter we perform an error analysis to obtain an impression of how the architectures we used can be further optimized for better results. In the final chapter we discuss our results and present ideas for future research.

Chapter 2

Background

This chapter introduces convolutional neural networks, as modern object detectors use them as feature extractors. In the following section we explain which operations convolutional neural networks perform and of the layers that they consist of. The second and third sections introduce two modern convolutional architectures, the Inception Net [SLJ+14] and then the ResNet [HZRS15].

2.1 Convolutional neural networks

Convolutional neural networks [LBD+89] (CNNs) are specialized neural networks for grid data such as images or time-series. Time-series data can be thought of as a 1-D grid with values at each time step, and images as 2-D grids of pixels. CNNs have been incredibly successful in practical applications. They are based on a linear operation called convolution. In machine learning applications the input is often a multidimensional array of data and the kernel, also called the filter, is a multidimensional array of parameters which we want to learn. In the event that we have a two-dimensional image I with indexes i and j as the input, we also use a two-dimensional kernel K: The **convolution operation** S is then defined by the following:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

The convolution operation is commutative, so we can also write.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n).$$

Neural network libraries implement a related function called cross-correlation,

which is the same as convolution without flipping the kernel.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n).$$

Figure 2.1 shows an example of cross-correlation. This technique has the advantage that we do not need to calculate the kernel flipping operation, since the neural network is able to learn the flipped kernel K by itself. The downside is that the operation is no longer commutative. However, the neural network often consists of other functions that do not commute regardless of whether or not the convolution operation is commutative.

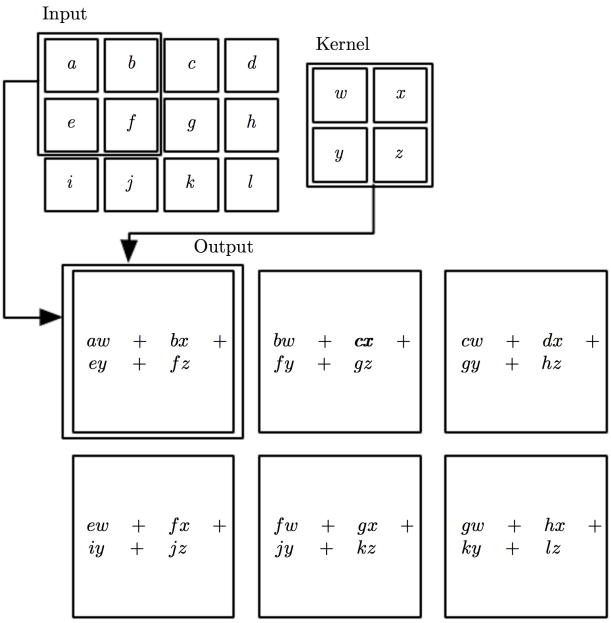

Figure 2.1: Example of cross-correlation convolution from [GBC]. The kernel moves in a sliding window approach over the input and generates the output. This configuration is called **valid** because there is no padding at the edges.

In the next section we motivate the usage of CNNs instead of multi-layer-perceptrons(MLPs) for grid data. We then provide a short introduction to the different layers that a convolution neural net consists of.

2.1.1 Motivation

A black-and-white image from our seedling dataset has the resolution of 512×512 pixels. A useful operation for object detection is an edge detector. To

build an edge detector one can subtract for each pixel the value of its neighboring pixel (either horizontal or vertical). This transformation can be described by a convolution kernel containing two elements. It would require $511 \cdot 512 \cdot 3 = 786,432$ floating point operations (one addition and two multiplication per output pixel) to compute it. If we used a feed forward network with matrix multiplication instead, it would save $511 \cdot 512 \cdot 511 \cdot 512$ entries in a matrix. The standard matrix multiplication algorithm with a runtime from $\mathcal{O}(n^3)$ would perform $512^4 \cdot 2$ floating point operations (one addition and one multiplication). The convolution operation would therefore be 174,762 times more computationally efficient than matrix multiplication. For linear transformations that are applied to small local regions across the entire input image, convolution is an extremely efficient method.

In the following, we will present three central ideas from convolutional neural networks that support them to process large inputs efficiently.
Parameter sharing refers to the concept that we can use one kernel multiple times over different regions in a model. Contrary to a traditional neural net where each entry in the weight matrix is used only once in the forward propagation step, in convolutional neural networks the filters are applied multiple times. For low level features like edge detection, it is beneficial not to have to store large weight matrices to detect edges.
Parameter sharing gives the convolution layer a property called **equivariance** to translation. A function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$. In the context of convolution, g can be any function that translates the input, then the convolution (f(x)) is equivariant to g. This property is important for the feature extraction process because convolution creates a 2-D map where features appear in the input. The feature representation should move the same amount if the object in the input moves. This is a useful trait for edge detection, as the same edges appear approximately everywhere in the image. Therefore, if we want to share parameters across the image, it should not depend on the order of functions. Figure 2.2 shows an example of parameter sharing.

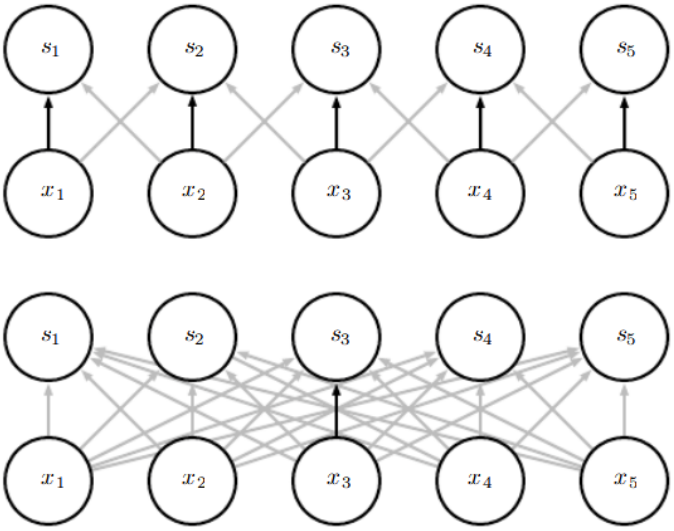

Figure 2.2: Example of parameter sharing from [GBC]. Black arrows indicate that a parameter is computed and saved to connect multiple nodes. In convolutional neural networks the parameter is shared over all connections (Top). In the example below, one parameter is used only for a single connection, no parameter sharing occurs and multiple parameters for each layer are necessary. This is the case in traditional neural nets.

Another important concept with regard to CNNs is **sparse interactions**. Sparse interactions (also called sparse connectivity or sparse weights) are accomplished by having a kernel smaller than the input. For example, one image in the seedling dataset has the resolution $512 \times 512 \times 3$, but we can detect meaningful features such as edges with filters only tens or hundreds of pixels in size. The consequence is that we have to store fewer parameters which reduces both the memory requirements and also the computation effort. A further consequence is that we have to learn fewer parameters and therefore don't need very much data. Sparse connectivity also leads to an effect called receptive field(see figure 2.3).

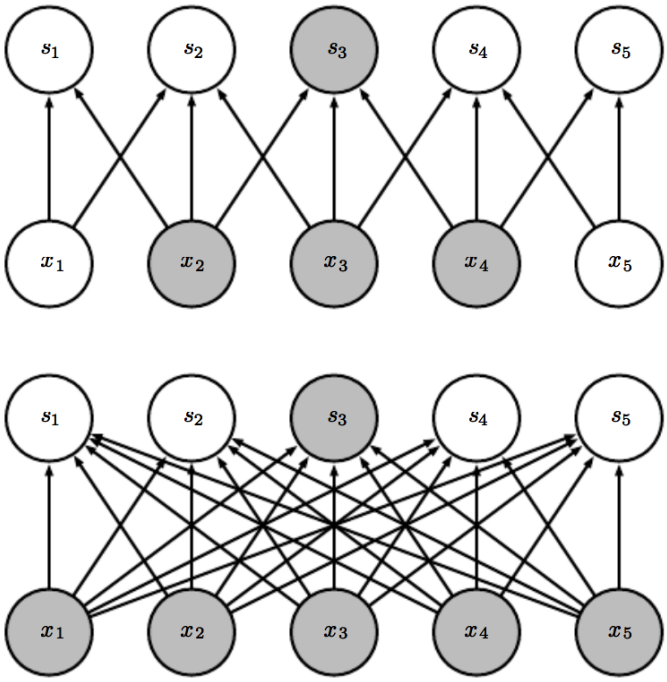

Figure 2.3: Example of sparse interactions from [GBC]. The upper example demonstrates how a given output unit s_3 is influenced by the input units x_2, x_3 and x_4 (kernel with a size of 3) in a convolutional neural net. The lower example shows the connections in a traditional fully connected neural net. Here each unit depends on all previous units.

Sparse connections enable the network to build low-level features like edge detectors in early layers. These units are affected by only a few pixels. Convolutional neural networks are also able to generate high-level features in later layers through the receptive field effect.

2.1.2 Convolution Layer

This section explains the hyperparameters of the convolutional layer in CNNs. As one can see in figure 2.1 the convolution operation shrinks the input by k+1, where k is the size of the kernel. If we have an image with the resolution 26×24, the output of the convolution with a 3×3 kernel would be $26 - 3 + 1$ and $24 - 3 + 1 = 24 \times 22$. Generally, the output dimensions for a two dimensional input (n, m) with a two-dimensional kernel (i, j) is calculated as follows:

$$(n - i + 1, m - j + 1)$$

This is a problem for deep convolutional neural networks with dozens of convolutional layers, because after a few steps the input will shrink more and more until only multiple small feature maps remain. **Zero padding** is a method to avoid this shrinking. It adds zero values at the edges in order to make it possible to retain the same dimensions. This configuration is called **'SAME'** since the convolution operation does not change the number of dimensions. The calculation of the output dimension changes to the following:

$$(n - i + 2 * p + 1, m - j + 2 * p + 1)$$

where p is number of zeros added to the input edges.
Another method to influence the output dimension besides changing the kernel size or using padding is by changing the **stride**. A stride size of one can be understood to mean that the filter has a step size of one while convolving over the input data. One can calculate the output dimensions involving a stride with the formula:

$$(\frac{n - i + 2 * p + 1}{s}, \frac{m - j + 2 * p + 1}{s})$$

The padding configuration, the filter size and the stride are important hyperparameters for the convolution layer in CNNs.

2.1.3 Detector Layer

The detector layer consists of non-linear functions such as sigmoid or tangent hyperbolic functions. The most commonly used function in convolutional nets is the rectified linear unit (ReLu). The ReLu function is defined as follows:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ \text{x} & \text{else} \end{cases}$$

Detector layers introduce non-linearities in neural networks and enable them to learn functions so that the output cannot be reproduced from a linear combination of the inputs. If we use only linear functions, the effect of the network would just be a linear transformation of the input. To generate a universal function approximator it is necessary to use non-linear functions such as ReLu [Hor91].

2.1.4 Pooling Layer

Pooling is a frequently used operation in convolutional neural networks. The pooling layer is special in that it constitutes a layer without parameters. A pooling function replaces the output of the previous layer with a summary statistic. For example, the average pooling [ZC88] operation calculates the average of a rectangular region. Pooling helps to make the features invariant to small translations of the input. One can translate an image of a cat image by a small amount and it still remains an image of a cat. This is often used for images that have varying dimensions because it is possible to scale any dimensions to a fixed representation by taking the maximum of the four quadrants in an image. This is especially important in the classification layer where the input must be of a fixed size.

The convolution layer, the detector layer and the pooling layer are the most common layers in modern convolutional neural networks. Figure 2.4 shows the stacking of these layers.

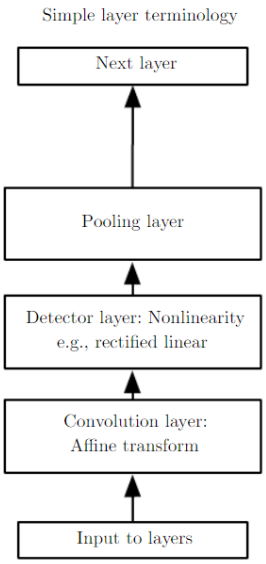

Figure 2.4: The components of a typical convolutional neural network according to [GBC]. In the simple layer terminology the convolutional net is viewed as a larger number of simple layers. In this terminology, not every layer has parameters.

2.1.5 Fully-connected layer

The hidden layers on top of a CNN are typically fully-connected layers. This is often between a flattened output from a convolutional layer and a soft max layer. However a fully-connected layer needs a small data volume because otherwise it will have too many parameters and slow down the training. Stride and pooling are often used in convolutional layers to reduce the data volume before the fully-connected layers. A convolutional neural network which has no such layers is called a fully convolutional network (FCN). If the network is used for object detection, it usually does not need any fully-connected layers. The convolutional network is used as a feature extractor in such cases.

The next two sections introduce two case studies, the Inception net [SLJ+14] and the residual net [HZRS15]. Both of these are later used individually and combined as feature extractors for object detection.

2.2 Inception Net / GoogLeNet

The first version of the Inception net, also called GoogLeNet, introduced the concept of **Inception modules** in 2014. The most straightforward way to improve the feature extraction process is by adding additional layers. This can be done in two ways, through the depth of the network or the width of the layer. Given the availability of a large amount of labeled training data, this is a promising means of increasing the quality of the network. However deeper/broader networks come with a computational burden. Increasing the number of layers or their width introduces new parameters which increase the use of computational resources. The creators of the Inception net describe the motivation behind their architecture as follows:

"The main idea of the Inception architecture is based on finding out how an optimal local sparse structure in a convolutional vision network can be approximated and covered by readily available dense components."

We first introduce the naive Inception module and then explain how the addition of 1×1 convolutions reduces the computational costs of the module.

2.2.1 Inception Module

The naive Inception module consists of 1×1, 3×3 and 5×5 filters, as well as a 3×3 max pooling layer. Figure 2.5 shows the naive approach. All layers use the 'SAME' configuration so they keep their dimensions. The original Inception net had seven Inception modules stacked on top of each other. As justification

for the use of 1×1 convolutions, let us consider the following example. Given an input($28 \times 28 \times 192$) to a 5×5 convolution with the same configuration ($s = 1, p = 1$) and 32 filters we would need $28 \cdot 28 \cdot 32 \cdot 5 \cdot 5 \cdot 192 \approx 120$ million multiplications. Modern computer architectures are capable of doing 120 million multiplications, but we must consider that this is only one module of seven and that there are additional layers outside of the Inception modules. The next section shows how we can reduce this computational burden.

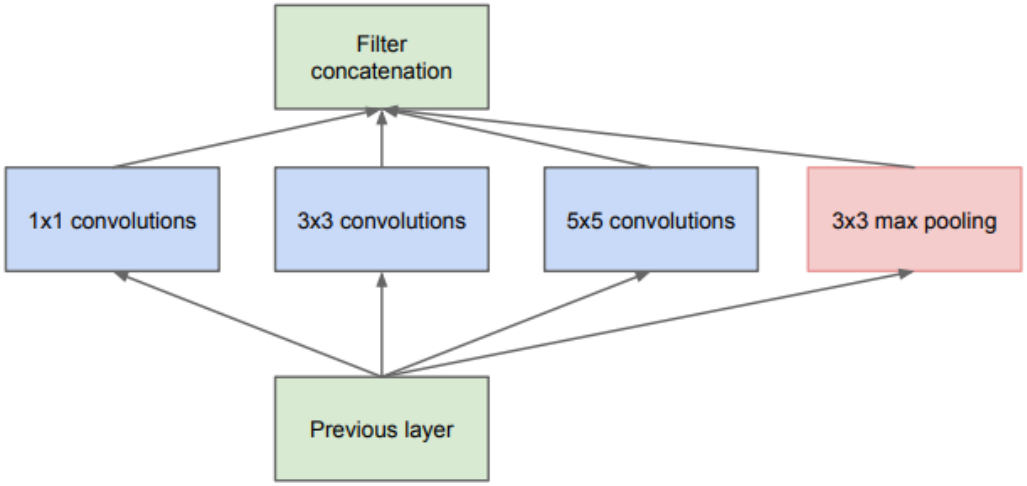

Figure 2.5: Naive version of the Inception module [SLJ+14].

2.2.2 1×1 Convolutions

Figure 2.6 shows the actual Inception module. As one can see, the 1×1 convolutions happen before the 3×3 and the 5×5 convolutions. Following the example from the previous section, we now need $28 \cdot 28 \cdot 16 \cdot 1 \cdot 1 \cdot 192 \approx 2.4$ million multiplications for the 1×1 convolution with 16 filters and $28 \cdot 28 \cdot 32 \cdot 5 \cdot 5 \cdot 16 \approx 10$ million multiplications for the 5×5 convolution with 32 filters. This approach now uses only 10% of the multiplications used in the naive approach. One can view the 1×1 convolutions as low-dimensional embeddings comparable to word embeddings from natural language processing.

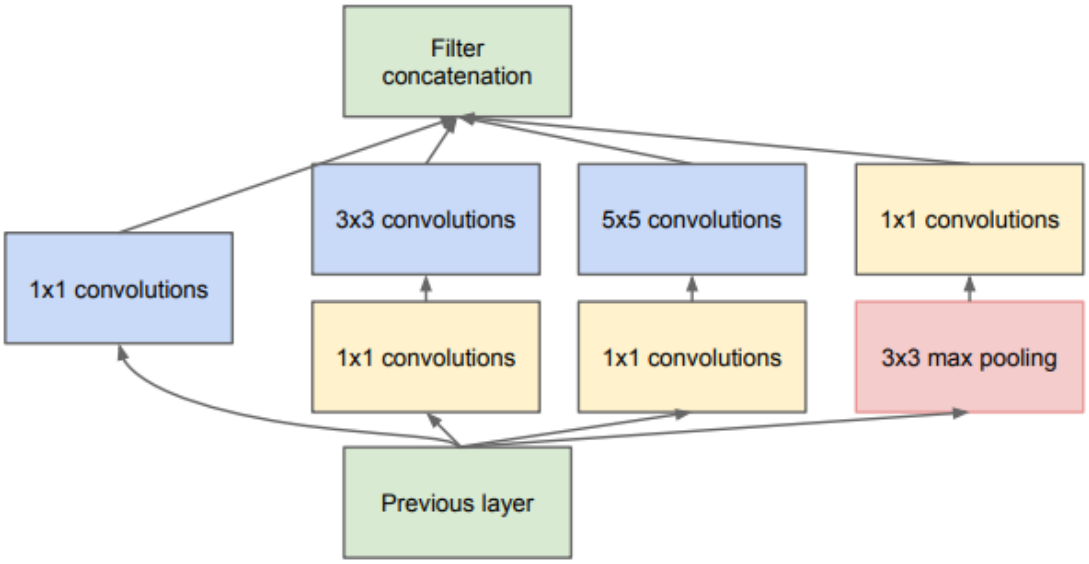

Figure 2.6: Inception module with dimension reduction [SLJ+14].

2.3 ResNet

Empirical results have shown that if you make neural networks deeper they should produce better results [GBI+13]. New practical experiments, however [HZRS15], have indicated a contrary behavior with an increasing depth of the networks. The authors of the Residual Net have seen a phenomenon called the **degradation problem** that occurs if convolutional networks with more layers are given worse training and test errors than a shallow one (Figure 2.7). Consider a shallow architecture and a deeper equivalent that adds more layers to the shallow one. When we copy the weights from the shallow one to the deeper net and the added layers are identity mappings, both networks should produce the same results. The experiments, however, showed that current solvers are unable to find these identity mappings.

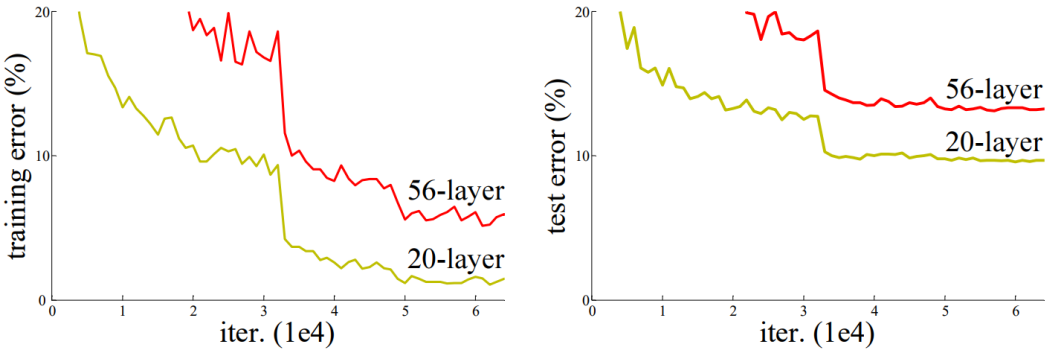

Figure 2.7: Comparison of 20-layer and 56-layer networks are compared. The left shows there the training error on CIFAR-10 and the right shows the test error. Contrary to the theory, the shallow network has a lower training and test error. The author experienced a similar phenomenon on the ImageNet benchmark [RDS+15].

Residual nets therefore introduced short cuts, also called skip connections, to convolutional neural nets. They enable the network to learn identity mappings and skip one or more layers. They add neither extra parameters nor computational complexity. Residual networks show that the usage of skip connections enables them to produce better results with deeper nets.

Inception-ResNet [SIV16] combines the Inception module from the Inception Net and the skip connections from the Residual Net. The authors found that residual connections accelerate the training of Inceptions networks significantly.

Chapter 3

Convolutional Object Detectors

This chapter discusses different convolutional object detectors based on convolutional neural networks (CNNs). In contrast to the classification task, the object detectors work on small regions of the input image, also called region proposals. We further discuss the possibilities for generating these region proposals, also called regions of interest (RoI). We introduce the three object detectors which are also used in the Experiments and Results chapter. There also exist several other object detectors, such as YOLO [RDGF15].

3.1 Regional CNN (R-CNN)

The Regional convolutional neural network (R-CNN) [GDDM13] was the first object detector to use convolutional neural nets as a feature detector for object detection. In 2012 it defeated the system that was the best at the time by 30% in the VOC2012 challenge. The approach splits the object detection task into two simpler ones:

1. **Region proposal generation**
   Given an image I, generate a list of region proposals which likely contain an object of interest.

2. **Region classification**
   Given a region proposal, classify it as an object of class C.

In R-CNN the first task uses an approach called **selective search** [UvdSGS13]. Selective search uses a bottom-up segmentation, merging regions of different sizes, and tries to group adjacent regions by texture, color and intensity. It runs relatively fast on a CPU: for 1000 region proposals it needs only a few seconds. However, as we will see in the Faster R-CNN section, this computation is the bottleneck for the inference step in the neural network.

For the second task R-CNN takes the region proposals from the selective search as input. The recall of the region proposal method must be high, because otherwise the network will not be able to classify them if they are not detected. To achieve a high recall the region proposal algorithm often outputs a huge number of regions (around 2000 for a standard image). The task of the R-CNN is to test whether these regions are significant and to classify the objects they represent. The last fully connected layer is often modified according to the number of classes in the specific dataset. The convolution net is frequently pretrained on ImageNet to increase the quality of the low-level features such as edge detectors. After the pretraining, the net is fine-tuned on the region proposals and outputs an embedding of the features learned. On top of this, multiple support-vector-machines (SVMs) which have been trained separately for each class, classify the fixed size embedding. Additionally, a bounding-box regression is performed to tighten the bounding-box positions around objects. Figure 3.1 shows the architecture of the R-CNN.

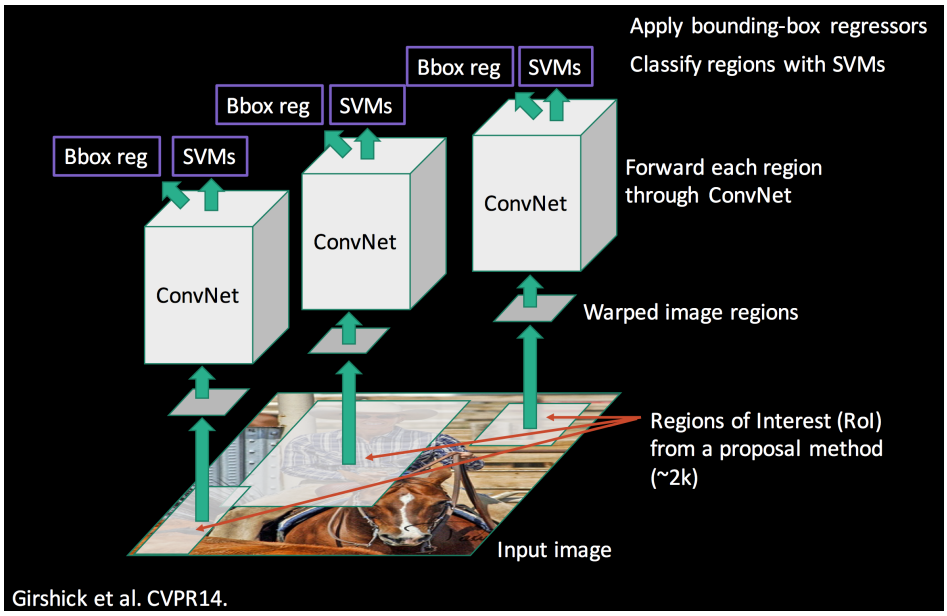

Figure 3.1: The architecture of R-CNN [Hui].

3.1.1 Drawbacks

Using and training the R-CNN network is a complex process. It requires the pretraining of the convolutional net on a huge dataset such as ImageNet. In addition, it must be fine-tuned with the new object classes and the background classes. For each image the proposals have to be saved to the disk and warped

to the CNN input size. For each class there is also binary SVM which has be to trained to distinguish between object and background. Finally a regression was performed to tighten the bounding boxes around the objects. The training of such a model took multiple days because the network was trained on each proposal. The test time per image was extremely slow, up to one minute per image, even on a GPU. The next model, an improvement to R-CNN, shows how we can achieve the same accuracy without training the CNN for each proposal.

3.2 Fast R-CNN

Fast R-CNN[Gir15] is an object detector based on R-CNN. It still uses as an input region proposals from methods such as selective search. Instead of training the network for each region, it is trained now for each image and shares the computation across ≈ 2000 proposals. Each region does not need the whole extracted feature map from the CNN: only the features at the spatial local of the region are important. The region of interest pooling layer (RoIPool) is responsible for the mapping. RoIPool shares the forward pass of a CNN for the original image across its regions. The RoIPool layer is located after the last convolutional layer (see figure 3.2).

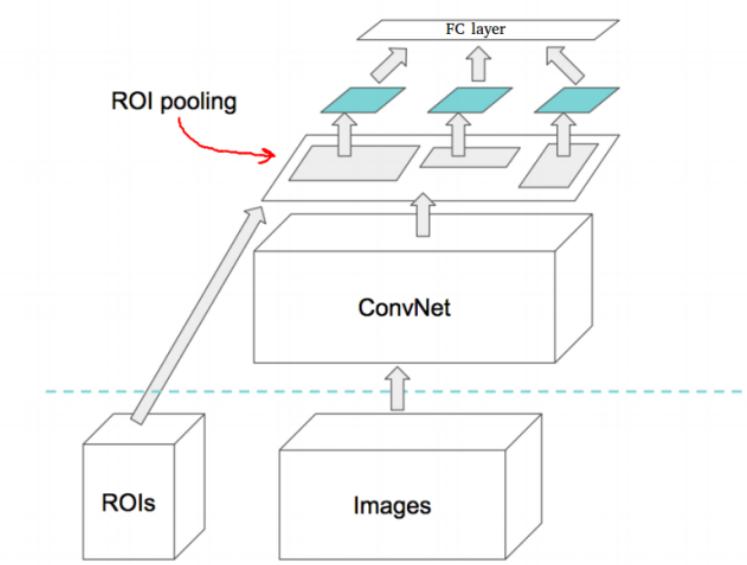

Figure 3.2: Location of the RoI pooling layer in Fast R-CNN. Image from [San]

The fixed-length vectors from the RoIPool layer are then fed into fully-connected layers that are connected to two output layers: a real-valued layer

that outputs bounding box coordinates computed during regression and a softmax layer that produces probability estimates for the different object classes. In Fast R-CNN, binary SVM and linear regression are not used. Compared to R-CNN, Fast R-CNN reduces the training time by up to a factor of 18, and the test time by a factor of up to 100.

3.3 Faster R-CNN

Faster R-CNN [RHGS15] removes the inference bottleneck selective search from Fast R-CNN by using a region proposal network (RPN) which outputs regional proposals for each image. It is nearly computation free because Faster R-CNN uses a single network for the region proposals and classification of the regions. In the next section we explain region proposal networks and how they work.

3.3.1 Region Proposal Network

The RPN takes an image as input and outputs several rectangular region proposals. It uses a fully convolutional network [SLD17] to generate the proposals. A sliding window is moved across the features map of the last shared convolutional layer. This window has a size of $n \times n$ (the authors used $n = 3$) and is fed into two fully connected layers, a box regression layer *reg* and a box classification layer *cls*. The authors state that for the VGG features map the receptive field is 228 pixels large. For every sliding window, they predict multiple region proposals depending on the number of anchors. Anchors are centered at a specific sliding window and are associated with a scale and aspect ratio. These anchors are translation invariant, and the RPN guarantees that if an object in an image is translated the network will predict the proposal in either location. The performance of the architecture is sensitive to the number of anchors and their different scales and aspect ratios [RHGS15]. The RPN loss function is defined as:

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

.

Here i is the index of an anchor and p_i is the probability of anchor i being an object. p_i^* is the ground-truth label and is 1 if the anchor is positive and 0 if the anchor is negative. t_i^* and t_i are vectors representing the four coordinates of the ground truth and the predicted bounding box. L_{cls} is the classification loss over two classes and L_{reg} is the L_1 norm for the regression. N_{cls} is the batch size and N_{reg} is the number of bounding boxes. Both are needed to

normalize the function. λ is used as a weighing factor to either increase or decrease the influence of the classification or the regression.

The RPN is trained end-to-end by backpropagation and stochastic gradient descent [LBD+89]. There are two general approaches for training the networks: alternating training where the RPN and the CNN are trained in turns, or joint training where both networks are merged into one network during training. Each iteration performs region proposal generation, the network classifies the regions and computes the loss functions, the backpropagation step combines the RPN loss and the convolutional loss and signals it back through the network.

3.4 Region-based Fully Convolutional Networks

Region-based Fully Convolutional Networks [DLHS16] (R-FCN) are comparable in terms of architecture with the Faster R-CNN network. They follow the popular two-stage object detection strategy, which consists of generating and subsequently classifying region proposals. In Faster R-CNN there is a dilemma of increasing translation invariance for image classification vs. respecting translation variance for object detection[DLHS16]. The image classification task requires translation invariance, and small changes of an object in an image should not affect the classification result. The object detection task however requires translation variance, a translated object must be recognized at a different location. In Faster R-CNN this issue is resolved by implementing a deep RoI subnetwork at the end of the architecture that improves accuracy but increases computational requirements due to the unshared per-RoI computation for each region. R-FCN avoids this by introducing position-sensitive score maps and position-sensitive RoI pooling to achieve translation variance without the use of a fully connected layer at the end of the network. On top of the newly introduced layer, no weight layers following (see figure 3.3) for an overview). In the aforementioned publication, the authors achieve the same accuracy on PASCAL VOC datasets [EEG+15] as Faster R-CNN with an test time reduction of 2.5× to 20×.

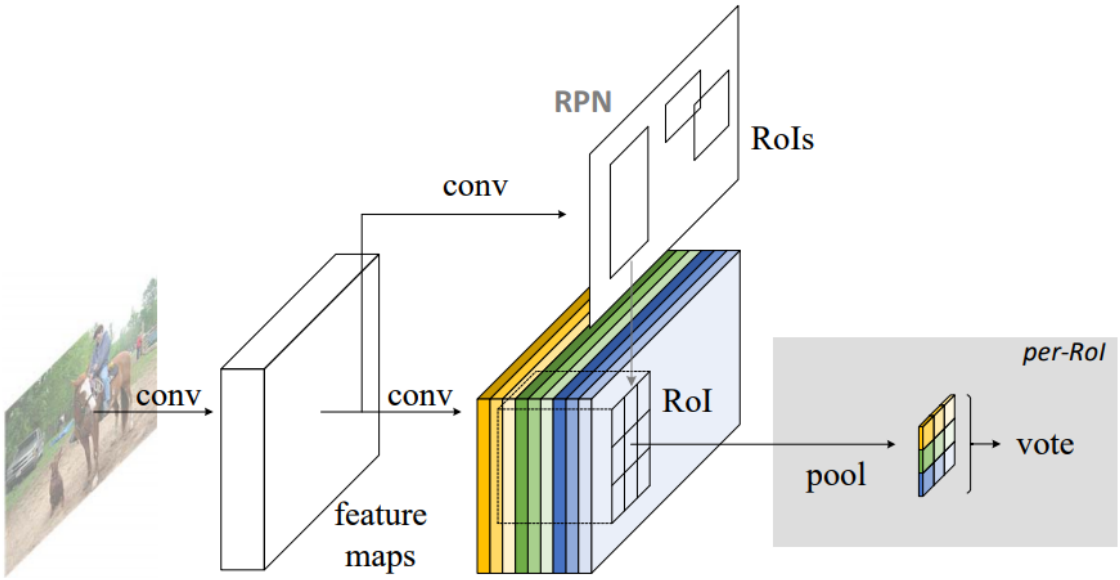

Figure 3.3: Overview [DLHS16] of the R-FCN architecture. An RPN suggests RoIs, which are then applied on the score maps.

3.5 Single Shot MultiBox Detector

Single Shot MultiBox Detector (SSD)[LAE+15] uses a complete different approach from that of Faster R-CNN and R-FCN, which completely eliminates the proposal generation step. SSD generates object detections using a single feedforward pass of the network. They use a set of default bounding boxes for each feature map cell at the end of the network. The default boxes are similar to the anchors in Faster R-CNN, but they are applied to several features map of different resolutions. This allows them to discretize the space of possible output shapes. The method generates a huge number of dense bounding boxes, which are mostly eliminated through a non-maximum suppression stage [LAE+15]. Most region proposals contain no interesting objects they are discarded after the object detection phase if they fall below a predefined confidence threshold.

Chapter 4

Object Detection in UAV Images

We start this chapter with the problem statement. We then discuss problems which occurred during the practical work and present possible solutions. The solutions are then tested in the following chapters through multiple experiments.

4.1 Problem Statement

Bird's-eye view images from different seismic lines are captured from an unmanned aerial vehicle. We look for a function which has an image as input and outputs bounding boxes covering conifer seedlings. This task is known as object detection and is a standard problem of computer vision. We show that current state-of-the-art algorithms, particularly convolutional neural networks (CNNs) are able to extract meaningful information from the aerial images. The main task of the thesis is to review these techniques and adapt them to the special images we have. We also perform experiments to understand how variables such as flight height and boreal forest seasons influence the object detection task.

4.2 Dataset

The main task of the experimental part is to test how well object detectors can perform on images taken from unmanned aerial vehicles (UAVs). We used a collection of 859 images from five different seismic lines (see table 4.1). The objective was to detect small conifer seedlings in the images and draw bounding boxes around them (See figure 4.1).

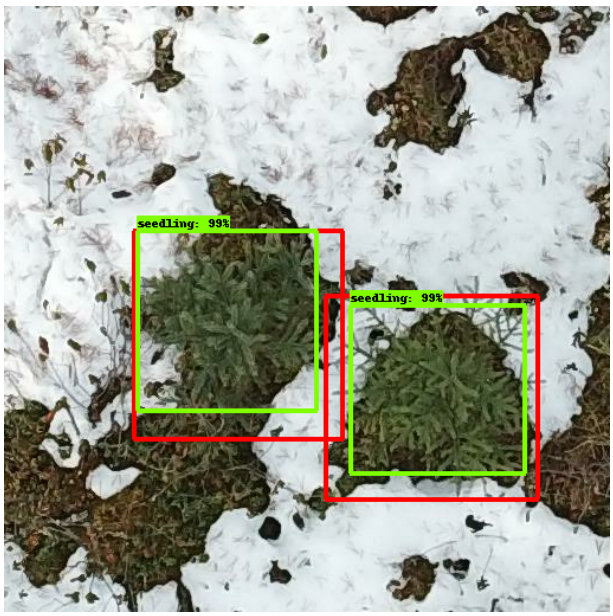

Figure 4.1: The goal of the object detector is to predict the location of the conifer seedlings (green bounding box) after training on annotated (red bounding box) conifer seedlings.

Site	Flight Height	Month of the Year	Resolution	GSD per pixel	Images
441	5 m	08	4000 × 3000	3.5 mm	39
441	30 m	08	4864 × 3648	7.5 mm	97
449	5 m	08	4000 × 3000	3.5 mm	51
449	30 m	08	4864 × 3648	7.5 mm	93
450	5 m	08	4000 × 3000	3.5 mm	34
450	5 m	10	4000 × 3000	3.5 mm	94
450	30 m	08	4864 × 3648	7.5 mm	91
450	30 m	10	4864 × 3648	7.5 mm	91
464	5 m	08	4000 × 3000	3.5 mm	37
464	5 m	10	4000 × 3000	3.5 mm	115
466	5 m	08	4000 × 3000	3.5 mm	35
466	5 m	10	4000 × 3000	3.5 mm	82

Table 4.1: Overview of the seismic line dataset

The ground sample distance (GSD) in an image is the distance between two pixels in the image measured on the ground. For example, in an image with a 1 mm GSD, adjacent pixels are 1 mm apart on the ground. The ground sample distance is a measure of the image quality. Later we present experiments to

determine the influence of the GSD on the object detection accuracy. The GSD in cm/pixel is defined as follows:

$$GSD = \frac{S_w * H * 100}{F_r * I_w}$$

where S_w is the sensor width(mm), H is the flight height(m), F_r is the focal length of the sensor(mm) and I_w is the image width in pixels.

Deep convolutional networks require a large amount of training data. Since collecting and annotating a dataset requires a significant effort, we labeled only some of the available images. We focused our labeling on seismic lines 464 and 466 because they have the best lightning conditions and the most conifer seedlings. Training modern ConvNets can take 2-3 weeks across multiple GPUs on ImageNet. We therefore used pretrained models from the TensorFlow Object Detection API (see section 5.1 for more information). To understand the influence of the pretraining on performance, we conducted experiments whose results are presented in the following chapters. Since low-level features such as edge detectors are easily transferable to new datasets and are especially important for object detection, we anticipate a major improvement in accuracy compared to networks without transfer learning. Initializing a network with transferred features independent of the layer can also result in an improvement in the generalization performance compared to random weights after fine-tuning to a new dataset [YCBL14].

4.3 Cropping

This section explains how we preprocessed the images from the seismic line dataset. We decided to slice one image into several smaller ones because of the high resolution of the images(4000×3000). Object detectors with convolutional neural networks as feature extractor often require a huge amount of GPU memory. The requirements for the graphical processing unit memory increases in proportion with the image resolution. We initially opted for a resolution of 512×512 because we thought the resolution was low enough to be efficiently fed into a neural network, but not too small to avoid truncated seedlings. Since the images had a resolution of 4000×3000 and 512 is not divisible without remainder, we could only fit seven rows and five columns of slices in an image (see figure 4.2).

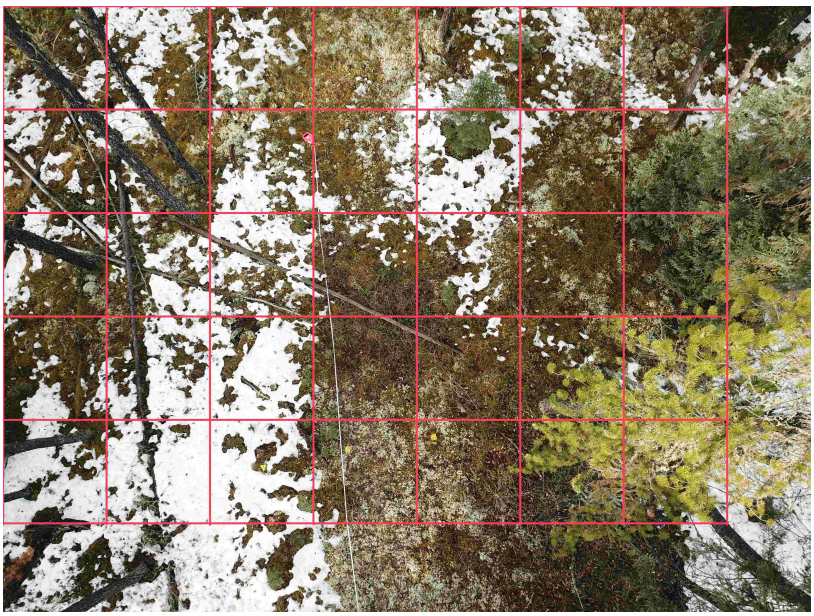

Figure 4.2: Slices of original image

We lose parts of the picture on the right and lower sides of the image. Since the images shot from the UAV overlap vertically, we do not lose much information. The conifer seedlings are usually located centrally on the image, so the missing information on the right side also does not affect the number of seedlings in the image.

A statistical dataset analysis showed that the average seedling in the images has a size of about 110×110. We also extracted the smallest seedling (23×19) and the largest (471×453) one (see figures 4.3 and 4.4 for more information). With a ground sample distance of 3.5 mm, the average seedling can be fit into a 30×30 cm bounding box, the smallest seedling within a bounding box of 8×6 cm, and the largest one within a bounding box of 1.6×1.7 meters.

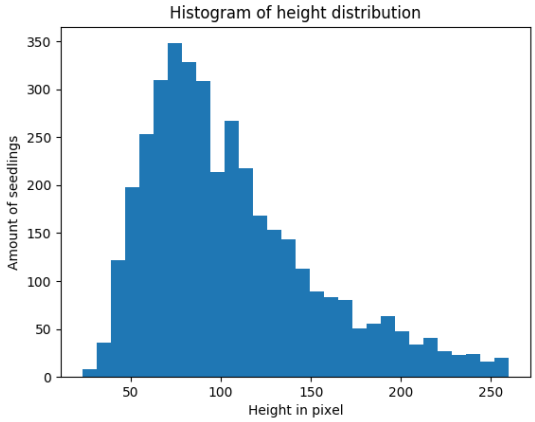

Figure 4.3: Histogram of seedling height

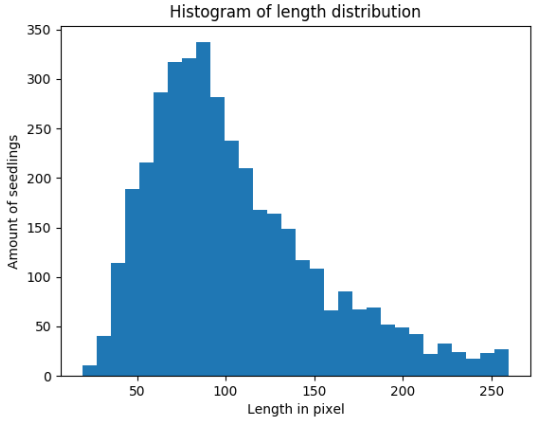

Figure 4.4: Histogram of seedling length

The cropping size of 512×512 turned out to be suitable. All models could be trained with this resolution and even the largest conifer trees that can still be classified as seedlings fit into a single slice.

4.4 Labeling process

After we sliced the image into several smaller ones, we started to label the seedlings on the seismic line dataset. We used the graphical image annotation tool LabelImg [Tzu15] to annotate the seedlings in the images (see Figure 4.5). LabelImg saves the annotations in the PASCAL VOC format.

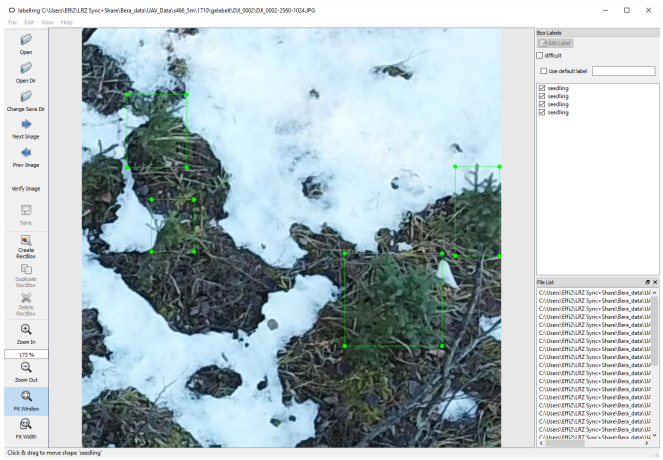

Figure 4.5: Labeling of the seedlings with labelImg

In total we sliced 269 images in 9415 cutouts and labeled the seedlings in all the cutouts. We annotated 3940 conifer seedlings in the 9415 picture slices from the seismic line 464 and 466. First, we labeled some images from site 464 from summer and winter and trained the first model on the data. We used the model to annotate the remaining images from sites 464 and 466 in order to have more data available. We manually fixed errors found to be contained in the annotations. However, manually moving the bounding boxes to the right spot is much faster than drawing them, so the labeling speed increased dramatically when using the annotations from the first model. We labeled all seedlings at the border of the slice if more than 50% of the seedling was visible. Since the pretrained network on PASCAL VOC does not have classes for conifer seedlings, we had to finetune the networks to detect them. In the following chapters we present the results from experiments regarding how many conifer seedlings annotations we need to reliably detect them in the test set. Since the first model was already quite good for annotating unseen images, we hypothesize that a small number of annotations (≈ 200) will be enough for good object detection accuracy ($mAP > 0.5$).

4.5 Training and Test Set

To generate a training and test set, we could not use the standard practice of sampling because images in the dataset often overlap, and it can not be ruled out that slices from different images overlap. This would mean that the same seedlings could appear in both the train and test set. We decided to use all image slices from site 466, all the summer images from site 464 and half the winter images from site 464 to build the training set (1863 image slices). The

test set consists of half the winter images from site 464 and consists of 661 image slices. Since the drone captured seismic lines often bidirectionally and sometimes four times, we had to assemble the training and test set manually to avoid overlapping. For the reduced training set with fewer annotations (see Section 6.8) we randomly deleted images from the standard training set until we had the correct number of annotations.

4.6 Lighting conditions

The summer images from seismic line 464 and the winter images from site 466 suffer from lighting problems (see Figure 4.6).

Figure 4.6: Example of an image with lighting problems. The seedling inside the blue bounding box has a different color than the one with the red bounding box.

During the labeling process it was often difficult to distinguish the conifer seedlings from the other vegetation. The lighting conditions further complicated this process. Because of the sun and the resulting shadows, conifers often appear as different colors. In the following chapter we present results from experiments regarding whether the object detector can be improved by using black/white images. We hypothesize that the colors are important to distinguish the conifer seedlings from other plants with similar shapes.

4.7 Seasonal effects

To understand the seasonal influence on the object detection accuracy we labeled both summer images (August) and winter images (October) at both sites (see figure 4.7 for a comparison).

(a) Summer image example (b) Winter image example

Figure 4.7: Comparison of boreal seasons

During the labeling stage we had more problems with summer images because of the dense vegetation. In some images it was difficult to distinguish the conifer seedlings from other plants. We believe that the object detector will have a significantly lower accuracy in these images compared to those from winter. Furthermore, we hypothesize that the features from the object detector trained on the summer image are more helpful for detecting seedlings in the winter image than vice versa. We review these hypotheses in the Chapter 6. To verify this, we constructed a dataset with all the summer images ($n = 453$) and a dataset with all winter images ($n = 2073$) to train a model on the summer/winter set and check the object detection accuracy on the other one. We also split the summer and winter dataset in training and test sets to determine which season presents more problems for the object detector. We decided to have a similar number of annotations for the winter and summer training set, so that the amount of data does not influence the results. For an overview see table 4.2.

Dataset	sites	images
Summer(total)	464, 466	453
Winter(total)	464, 4668	2073
Summer(train)	464, 466	519
Summer (eval)	466	215
Winter (train)	s466	761
Winter (test)	s464, s466	869

Table 4.2: Overview seasonal datasets.

4.8 Flight height

The seismic line dataset contains images from 5 and 25 m flight height. Since labeling additional images at 25 m requires significant effort, we down-sample the 5 m images to a ground sampling distance comparable to images captured at 25 m. We hypothesize that with increasing flight height, and therefore lower ground sampling distance, the object detection accuracy will deteriorate. We also hypothesize that an increase in flight height by a factor of 6 (30 m) significantly impairs the object detection performance, because the seedlings in the 25 m images were difficult to identify. The authors of the paper **Super-Resolution of Sentinel-2 Images: Learning a Globally Applicable Deep Neural Network**[LBG+18] describe a process of down-sampling images by first blurring them using a Gaussian filter with a standard deviation of $\sigma = \frac{1}{s}$ and then averaging over $s \times s$ windows where s is the desired scale ratio. This process is an approximation of actual images shot at this flight height. It does not include the alteration of angles. We conduct experiments with s $\in \{3, 5, 7, 9, 21, 31\}$ which results in images shot from 15, 25, 35, 45, 105 and 145 m height. See figure 4.8 for an example.

(a) Original flight height (5m) (b) Down-sampled to 25m

(c) Down-sampled to 45m (d) Down-sampled to 105m

(e) Down-sampled to 145m

Figure 4.8: Down-sampling examples

Chapter 5

Experiment setup

All models were trained on a computer with an Nvidia GEFORCE GTX 1080 Ti as GPU, 128GB Ram and an Intel i7-5930K CPU. For training on the GPU we used CUDA 9.0 and TensorFlow 1.6. We trained the models up to 60,000 steps, because in most cases the fine training of the networks reached a low training error around this point. The Faster R-CNN and R-FCN with ResNet-101 CNN had a step/sec of 0.15, which resulted in a training time of ≈ 3 hours. In the next section we introduce the TensorFlow Object Detection API and describe the available methods and their parameters. The end of the chapter discusses the evaluation of the experiments.

5.1 TensorFlow Object Detection API

We decided to use the TensorFlow Object Detection API because deep convolutional neural networks require a large amount of labeled training data. The framework offers a huge number of pretrained detection models which lowers the requirement of labeled training data. See table 2.1 for an overview of available pretrained models on the COCO dataset [LMB+14]. To train a model with the API, one first has to clone the repository and download the frozen models. Furthermore, the API requires a label map with all the classes which exist in one's own dataset. We also had to construct a persistent data structure, called TensorFlow Records. TensorFlow Records are the combination of the images and the annotations in XML. The TensorFlow API has multiple configuration files for the different object detectors and convolutional neural net combinations. These configuration files must be adapted to the custom training and evaluation set.

Object Detector	Convolutional Net	Speed (ms)	COCO MAP
SSD	MobileNet	30	21
SSD	InceptionV2	42	24
R-FCN	ResNet101	92	30
Faster R-CNN	InceptionV2	58	28
Faster R-CNN	ResNet50	89	30
Faster R-CNN	ResNet101	106	32
Faster R-CNN	InceptionResNetV2	620	37
Faster R-CNN	NAS	1833	43

Table 5.1: List of available pretrained object detectors in the TensorFlow Object Detection API and their performance and speed parameters

The Object Detection Training Pipeline can be configured by modifying the pipeline.proto file. At a high level the configuration file consists of five parts:

1. **The model configuration**
   This defines the object detector and the feature extractor.

2. **The training configuration**
   This section is for specifying the hyper parameters, such as the number of steps and learning rate. One can also define preprocessing techniques and define checkpoints of pretrained networks.

3. **The training input**
   Here the user can specify the training set and give further instruction concerning whether the data should be shuffled.

4. **The evaluation configuration**
   Here one can generate object detection visualizations and define multiple metrics for the evaluation.

5. **The evaluation input**
   This is where one defines the evaluation set. Typically this should be different from the training input dataset but one can also use the training input here to test whether the model suffers from overfitting.

The configuration file used a format with the name **protocol buffers** for serializing the data. The API also provides the user with predefined training and evaluation starting methods for the network.

5.2 Evaluation

For the evaluation of object detectors the most frequently used metric is mean average precision (MAP) [SM86]. In order to calculate the MAP we first explain the calculation of intersection over union (IoU). The IoU is a ratio between the union and the intersection of predicted and ground truth boxes. It is also known as the Jaccard Index in reference to Paul Jaccard. Given two bounding boxes, A and B, the IoU is defined by:

$$IoU(A, B) = \frac{A \cap B}{A \cup B}$$

where the intersection operation is defined as the pixels that rectangles A and B share. The union operation is defined as the combined pixels from rectangles A and B. The MAP values are calculated for a fixed IoU threshold(usually $IoU = 0.5$, so that the ground truth and bounding boxes overlap in at least 50% of the area).

In order to calculate the MAP in the area of object detection [EEG+15], one needs to calculate the AP for each class (in this work there is only one seedling class), and then compute the mean over all classes. To calculate the AP for a class, one needs to compute the precision (P) and recall (R):

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

True positives (TP) are the detections with $IoU > 0.5$. False positives (FP) are the detections with $IoU \leq 0.5$ or that have already been detected. False negatives (FN) are the number of objects were the method failed to produce bounding-boxes. In the context of object detection there are no true negatives (TN) because the images are expected to contain at least one object. Each bounding box also has a confidence value for a given class. A scoring method sorts the predictions in order and computes the precision and recall for each possible rank. The precision-recall curve is obtained by taking the recall values in the interval $[0, 1]$ and maximizing the precision for each value. Taking the mean of all values yields the AP for the class.

This metric can easily be used to compare different approaches to object detection, as seen in the next chapter. Since we have only one class in our experiments the mean average precision is equal to the average precision of the seedling class. Generally, when recall increases, precision decreases and vice versa: a sensitive model is able to capture a large percentage of objects

in an image, but also generates a high number of false positives. A model with a high threshold for detection yields few false positives but also leaves a higher percentage of objects undetected. The best balance between these two depends on the application area.

5.3 Hyperparameters of the Models

This section describes the hyperparameters we used for training Faster R-CNN, R-FCN and SSD.

Starting with Faster R-CNN and R-FCN the pretrained models had 12 anchors (4 different scales and 3 aspect ratios), used a maximum of 300 proposals in the first stage and weighted the localization loss by a factor of two. We used a batch size of one since we only had one GPU available and the memory was a limiting factor. We used stochastic gradient descent with momentum with a static learning rate of 0.0003 to optimize the loss function.

For the object detector SSD we used anchors with five different aspect ratios and multiple scales ranging from 0.2 to 0.95. The detector used 100 proposals and weighted classification and localization with the same weight. Since SSD uses a smaller network (Inception v2) and has a simpler architecture, we were able to increase the batch size to 24. We used the RmsProp optimizer [TH12] and an exponential decay learning rate beginning at 0.004 and having a decay factor of 0.95. In none of the models did we use regularization methods other than batch normalization. We followed the advice from the TensorFlow Object Detection API where they used the hyperparameters according to the publications ones. In the thesis we focused on comparisons of different models and CNN architectures. The next step would be to finetune the hyperparameters of the best models.

Chapter 6

Experiments and Results

In this chapter we present several experiments to evaluate the questions and hypotheses from the Chapter 4. We do this by training the SSD, R-FCN and Faster R-CNN models on the specified training and test sets and evaluating their object detection performance through the MAP evaluation metric and the precision-recall curve.

6.1 Comparison of Object Detectors

We first trained several models to compare the different object detectors without transfer learning. Faster R-CNN and R-FCN used ResNet101 as convolutional neural net, while SSD used a Inception v2 net. The plotted precision-recall curve and the MAP are presented in figure 6.1 and table 6.1.

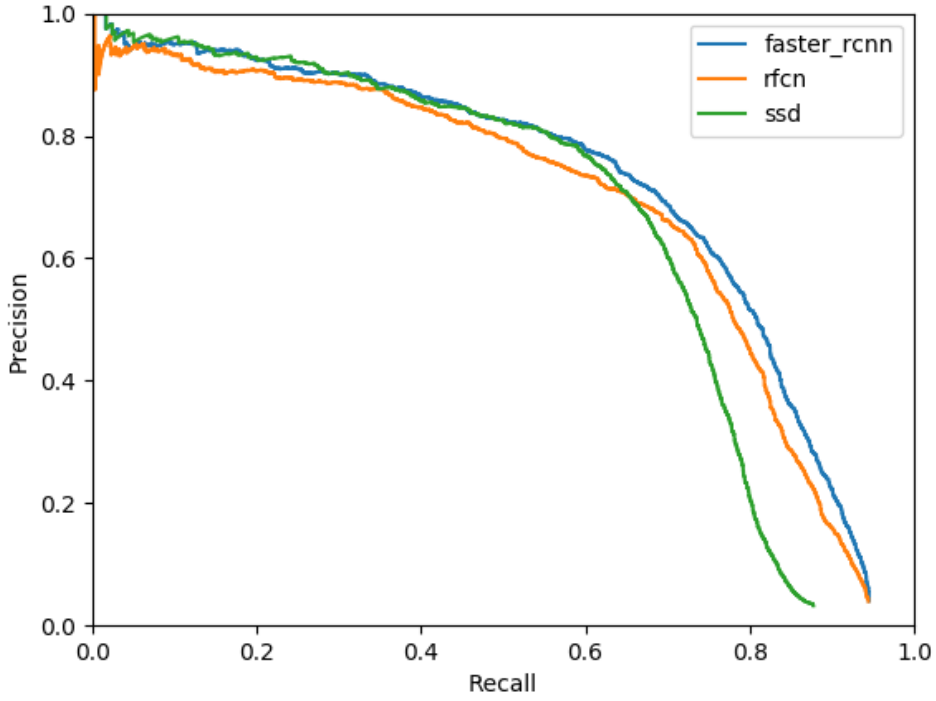

Figure 6.1: Precision-recall curves for Faster R-CNN, R-FCN and SSD without transfer learning.

Object Detector	MAP	COCO MAP
SSD	0.65	0.24
R-FCN	0.68	0.30
Faster R-CNN	0.71	0.32

Table 6.1: Comparison of Faster R-CNN, R-FCN and SSD on the conifer seedling dataset and the COCO dataset. The evaluation used MAP@[0.5]IoU as the metric.

Faster R-CNN had the best performance on the test set, followed by R-FCN and SSD. All three detectors achieve high recall values which are important for remote sensing of the seismic line. Compared to the detection quality from the detectors on COCO even the worst model, SSD performs far better on our custom dataset. The reason for this is that the COCO dataset has 80 different object categories and a large amount of different images compared to ours. If we added additional classes (see Chapter 8) the performance would be likely to decrease.

6.2 Transfer Learning

In this section we benchmark SSD, R-FCN and Faster R-CNN against each other with pretrained models on the COCO dataset. Training on a huge dataset like ImageNet or COCO should enable the network to learn low-dimensional features such as edge detectors which are transferable to our custom dataset. R-FCN and Faster R-CNN use pretrained weights from ResNet-101 whereas SSD uses Inception Net v2.

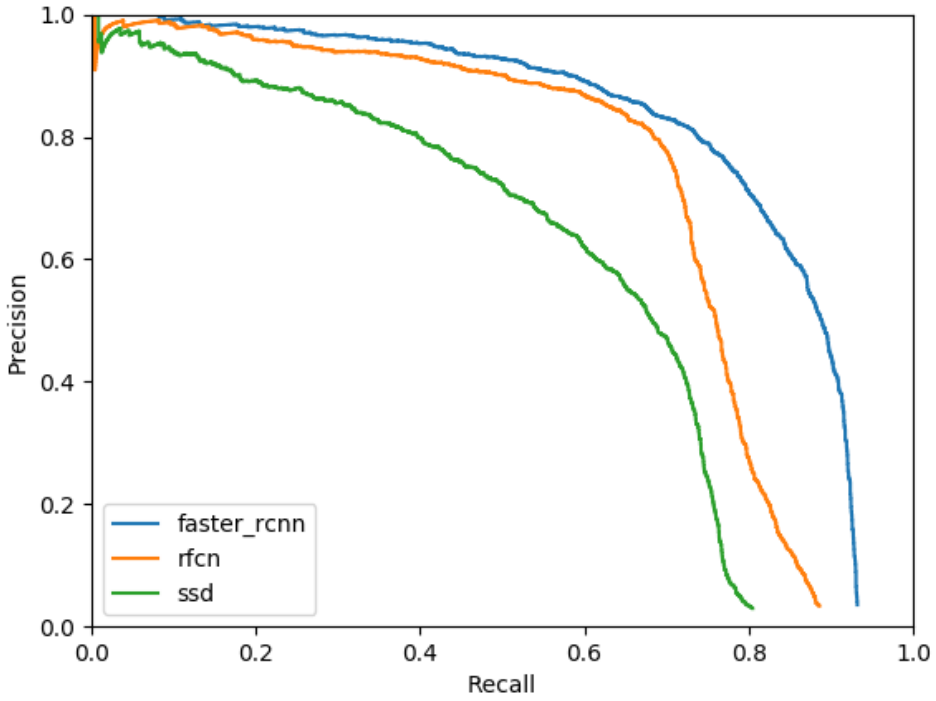

Figure 6.2: Precision-recall curves for Faster R-CNN, R-FCN and SSD with transfer learning.

Object Detector	MAP without TF	MAP with TF	Increase
SSD	0.65	0.58	-0.07
R-FCN	0.68	0.71	0.03
Faster R-CNN	0.71	0.81	0.10

Table 6.2: Comparison of Faster R-CNN, R-FCN and SSD on the conifer seedling dataset with and without transfer learning (TF). The evaluation used MAP@[0.5]IoU as the metric.

Faster R-CNN and R-FCN benefit from the pretrained ResNet-101 models by

increasing the MAP by 14% and 4%, respectively. The precision at low recall values increases compared to the non-pretrained models, whereas the recall for low precision values remains constant. The usage of the pretrained Inception v2 net decreases the performance of the SSD object detector by 11%. Further tests would be necessary to understand whether the decrease is caused by the underlying convolutional neural net or the object detector.

6.3 Preprocessing results

To understand the influence of the color channels on the object detectors we trained several models on black/white images. Since no black/white pretrained models are available in the TensorFlow Object Detection API, we started the training from randomized weights. Figure 6.3 and table 6.3 show the results.

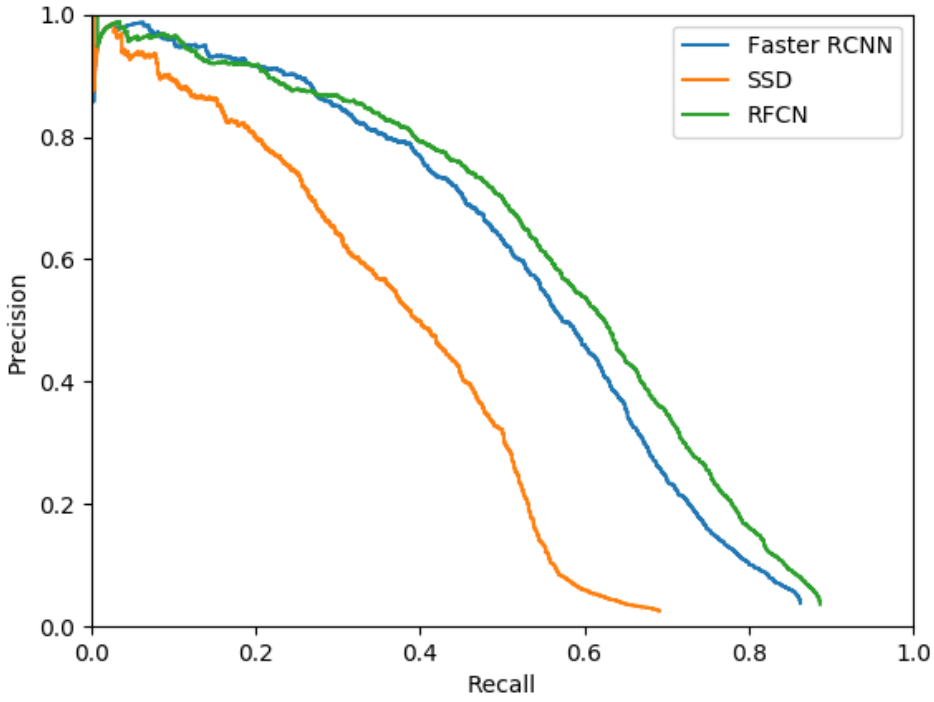

Figure 6.3: Precision-recall curves for Faster R-CNN, R-FCN and SSD on black/white images.

Object Detector	BW	No preprocessing	Increase
SSD	0.36	0.65	-0.29
R-FCN	0.57	0.68	-0.11
Faster R-CNN	0.54	0.71	-0.17

Table 6.3: Comparison of Faster R-CNN, R-FCN and SSD on the conifer seedling dataset with black and white images (BW) and no preprocessing. The evaluation used MAP@[0.5]IoU as the metric.

The usage of black/white images impaired the performance of all object detectors significantly. SSD has the biggest loss in performance followed by Faster R-CNN and R-FCN. The precision-recall curves show that the precision decreases significantly from the previous precision-recall curves. This indicates that the color is an important feature in distinguishing the seedlings from the other vegetation. Although many images suffer from bad lightning conditions and the resulting color deviations, the usage of black andwhite images is not beneficial for object detection on the seismic line dataset.

6.4 CNN architecture and layer depth

In this section we use different convolutional neural network architectures namely Inception v2, ResNet-50, ResNet-101 and Inception ResNet v2, in combination with Faster R-CNN to ascertain whether layer depth and model complexity have an advantageous effect on detector performance. See table 6.4 for a comparison of the architectures.

Object Detector	No. of layers	Parameters	MAP
Inception v2	42	10 M	0.66
ResNet-50	50	20 M	0.66
ResNet-101	101	42 M	0.81
Inception Resnet v2	467	54 M	0.71

Table 6.4: Comparison of multiple Faster R-CNN architectures on the conifer seedling dataset. The evaluation used MAP@[0.5]IoU as the metric.

We found that on smaller convolutional architectures Inception v2 achieved the same performance as ResNet-50 although it has only half the parameters. On deep architectures, ResNet-101 outperformed Inception ResNet v2 although it has 10 million fewer parameters. On this dataset the networks with Inception modules provide an advantage on small networks. On deeper networks the convolutional neural networks with residual connections benefit more than networks with the Inception modules. This is due to the degradation problem with deeper networks(see section 2.3 for more information).

6.5 Data augmentation

In this section we present the results from experiments concerning whether data augmentation techniques improve object detection performance. For the SSD object detector we used random horizontal flips and random crops as data augmentation options. Since the SSD object detector has difficulties with small objects (see [LAE+15]), we hoped that the usage of random crops would function as a zoom mechanism and increase the performance of the object detector, as the authors described. For Faster R-CNN and R-FCN we used random horizontal flips and random 90 degrees rotations as data augmentation options. An overview of the results is presented in figure 6.4 and table 6.5.

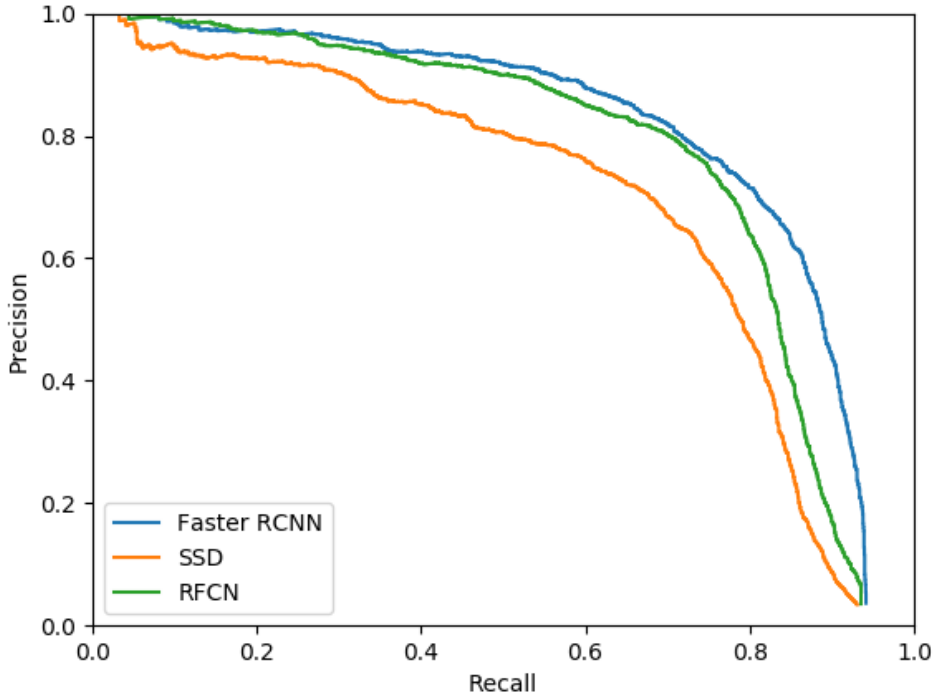

Figure 6.4: Precision-recall curves for Faster R-CNN, R-FCN and SSD with data augmentation.

Object Detector	No augmentation	Data augmentation	Increase
SSD	0.58	0.69	0.11
R-FCN	0.71	0.76	0.05
Faster R-CNN	0.81	0.80	-0.01

Table 6.5: Comparison of Faster R-CNN, R-FCN and SSD on the conifer seedling dataset with and without data augmentation. The evaluation used MAP@[0.5]IoU as the metric.

For the SSD feature extractor, data augmentation resulted in a large increase of 19%, R-FCN also benefited from the random rotations and horizontal flips with an increase of 7%. For Faster R-CNN we saw no increase in performance compared to no data augmentation. The increase for the SSD object detector is mainly based on the performance increase (+27%) on small objects (see section 7.1.1 for more information).

6.6 Seasonal influence

To understand the influence of the seasons on the object detector performance, we formed winter and summer data sets (see section 4.7). Table 6.6 gives an overview of the different training configurations and the object detection performance.

Object Detector	S	W	Train W | Eval S	Train S | Eval W
SSD	0.45	0.41	0.08	0.13
R-FCN	0.69	0.59	0.21	0.33
Faster R-CNN	0.71	0.65	0.27	0.41

Table 6.6: Overview of the seasonal training configurations. S stands for summer and W for winter. The used metric is MAP@[0.5]IoU.

Contrary to our belief during the labeling, the object detection performance is actually higher in the summer dataset than the winter dataset. Faster R-CNN (9%), R-FCN(16%) and SSD(9%) all perform better on the summer dataset. This result is even more surprising since the winter training dataset($n = 761$) is slightly larger than the summer dataset($n = 519$). A further result is that the model trained on the summer data detects conifer seedlings in winter images with a higher accuracy than vice versa. This is because the models trained on the winter data are not able to distinguish the conifer seedlings from other vegetation.

6.7 Resolution dependency

In order to understand how the flight height influences the object detection accuracy we conducted experiments on down-sampled images(see section 4.8 for detailed information on the down-sampling process). We trained models

on the down-sampled training set and evaluated them on the down-sampled evaluation set (see table 6.7 for results). We also used the models trained on the standard training set and evaluated the performance on the down-sampled evaluation set (see table 6.8 for results).

Object Detector	5 m	15 m	25 m	35 m	45 m	105 m	155 m
SSD	0.58	0.56	0.56	0.52	0.50	0.45	0.38
R-FCN	0.71	0.72	0.66	0.68	0.67	0.57	0.44
Faster R-CNN	0.81	0.78	0.75	0.72	0.72	0.66	0.55

Table 6.7: Comparison of Faster R-CNN, R-FCN and SSD on the conifer seedling dataset. All models were trained and evaluated on the down-sampled dataset.The evaluation used MAP@[0.5]IoU as the metric.

The object detector performance decreases gradually with increasing flight height across all object detectors. The results show that Faster R-CNN is able to detect conifer seedlings reliably even in images shot from 155 m.

Object Detector	5 m	15 m	25 m	35 m	45 m	105 m	155 m
SSD	0.58	0.51	0.44	0.31	0.17	0.01	0.00
R-FCN	0.71	0.59	0.44	0.29	0.16	0.00	0.00
Faster R-CNN	0.81	0.71	0.56	0.38	0.22	0.03	0.02

Table 6.8: Comparison of Faster R-CNN, R-FCN and SSD on the conifer seedling dataset. All models were trained on the standard data and evaluated on the down-sampled dataset. The evaluation used MAP@[0.5]IoU as the metric.

The object detection performance decreases significantly in evaluation on the down-sampled evaluation set with the usage of standard models (models trained on 5 m images). The models rely on the sharpness of the conifer seedlings for detection. Through the down-sampling process and the resulting loss in sharpness the object detector loses the ability to detect them. The Faster R-CNN standard model can still be used on images taken from up to 25 m, whereas SSD and R-FCN already have problems($mAP < 0.5$) with 15 m images.

Figure 6.5 below shows the dependency between the flight height in meters and the performance of the object detector in terms of MAP. It should be kept in mind that the graph consists of lines in order to improve the perception of the trend. We only evaluated the models at the points on the line. The values between the evaluation points are approximations.

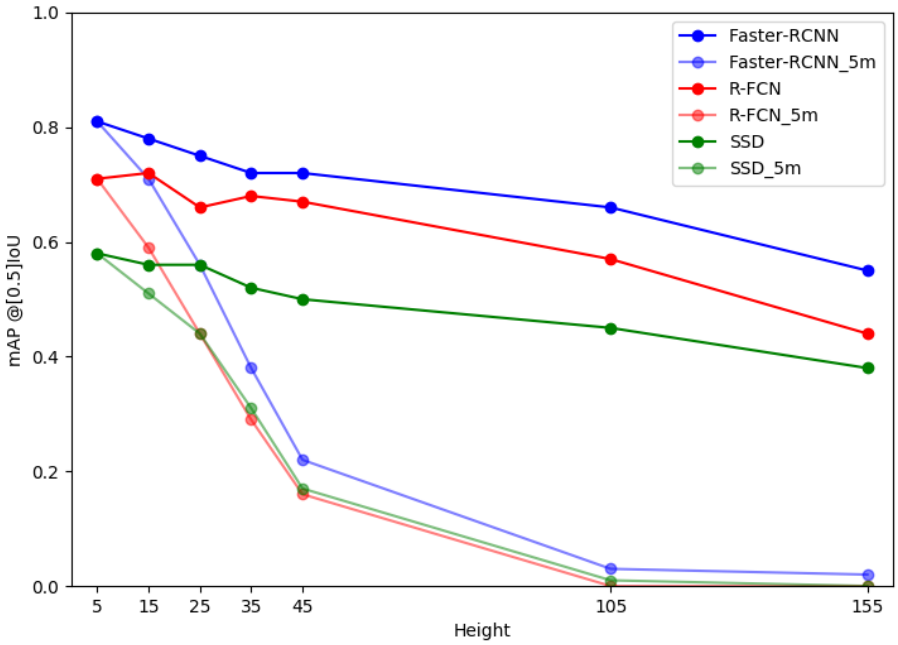

Figure 6.5: Dependency between the accuracy of the object detectors (MAP@[0.5]IoU) and the image resolution (in meters of flight height).

6.8 Dataset size

To determine how many annotations we need for good object detection performance ($MAP > 0.5$), we performed several benchmarks on different training set sizes. Table 6.9 shows the performance of Faster R-CNN, R-FCN and SSD on different numbers of annotations during training. We also used pretrained models to further investigate the influence of transfer learning by comparing the results to models without pretraining.

Object Detector	200	500	1000	2000	3940
SSD	0.16	0.30	0.39	0.51	0.58
R-FCN	0.66	0.67	0.69	0.73	0.71
Faster R-CNN	0.68	0.70	0.75	0.76	0.81

Table 6.9: Comparison of Faster R-CNN, R-FCN and SSD on the conifer seedling dataset on multiple training sets; all models were pretrained on COCO. The evaluation used MAP@[0.5]IoU as the metric.

Faster R-CNN and R-FCN showed high accuracy on the evaluation set even with only 200 annotations. SSD needed all annotations to get close

to the performance of Faster R-CNN and R-FCN. The comparison without pretraining (see table 6.10) shows a similar result.

Object Detector	200	500	1000	2000	3940
SSD	0.15	0.24	0.36	0.48	0.65
R-FCN	0.39	0.49	0.58	0.61	0.68
Faster R-CNN	0.52	0.59	0.62	0.65	0.71

Table 6.10: Comparison of Faster R-CNN, R-FCN and SSD on the conifer seedling dataset on multiple training sets; all models had no pretraining. The evaluation used MAP@[0.5]IoU as the metric.

Figure 6.6 shows the dependency between the number of annotations and the object detection performance. Here we also used lines to show the slope of the graphs although we performed the evaluation only at certain points (as marked with in the graph). By roughly doubling the amount of training data from 200 annotations to 500 annotations the performance of SSD increased by 87.5% (60% without pretraining). Faster R-CNN and R-FCN have an increase of only 3% (14% without pretraining) and 1.5% (25% without pretraining). Models without pretraining, in particular, profit from additional training data.

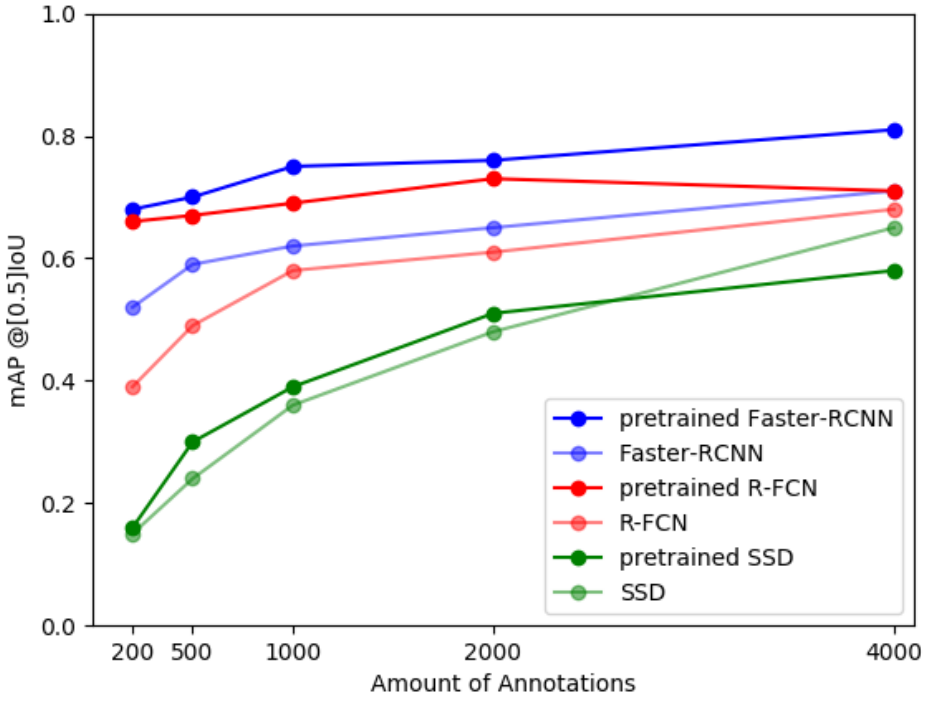

Figure 6.6: Dependency between the accuracy of the object detectors (MAP@[0.5]IoU) and the number of annotations.

Chapter 7

Discussion

This chapter presents the results from an error analysis on the standard model and on resolution dependency problem. In conclusion, we summarize our contributions from the evaluation chapter.

7.1 Error Analysis

We will take a closer look at the standard model to improve our understanding of error sources.

7.1.1 Small, medium and large objects

To further analyze the source of errors, we split the evaluation set into three parts. The seedlings specified as small ($n = 197$) have less than 7,000 pixels, with a ground sampling distance of 3.5 mm per pixel; this includes all seedlings smaller than 30×30 cm. The medium dataset contains all seedlings which have more than 7,000 pixels and less than 35,000 pixels, which comprises all seedlings up to 66×66 cm ($n = 128$). The large dataset consists of seedlings bigger than 66×66 cm ($n = 36$). We evaluate the standard models from section 6.1 on the divided evaluation set.

Object Detector	Small	Medium	Large	All
SSD	0.43	0.79	0.99	0.58
SSD (augmentation)	0.55	0.84	0.97	0.65
R-FCN	0.48	0.85	0.99	0.71
Faster R-CNN	0.72	0.91	1.00	0.81

Table 7.1: Comparison of Faster R-CNN, R-FCN and SSD on the conifer seedling dataset depending on seedling size. The evaluation used MAP@[0.5]IoU as the metric.

All three object detectors function almost without errors on the large seedlings. On medium seedlings all detectors perform reasonably well, but Faster R-CNN here begins to outperform the other detectors. Only Faster R-CNN is able to reliably detect small seedlings. Figure 7.1 show the precision-recall curves from Faster R-CNN on small, medium and large seedlings. The model is able to detect 100% of the large seedlings with a precision of 90%. 90% of medium sized seedlings can still be detected with a false positive rate of 20%. Small seedlings are more difficult for Faster R-CNN to detect. To improve the overall performance, the best way would be to increase the detection rate of small seedlings.

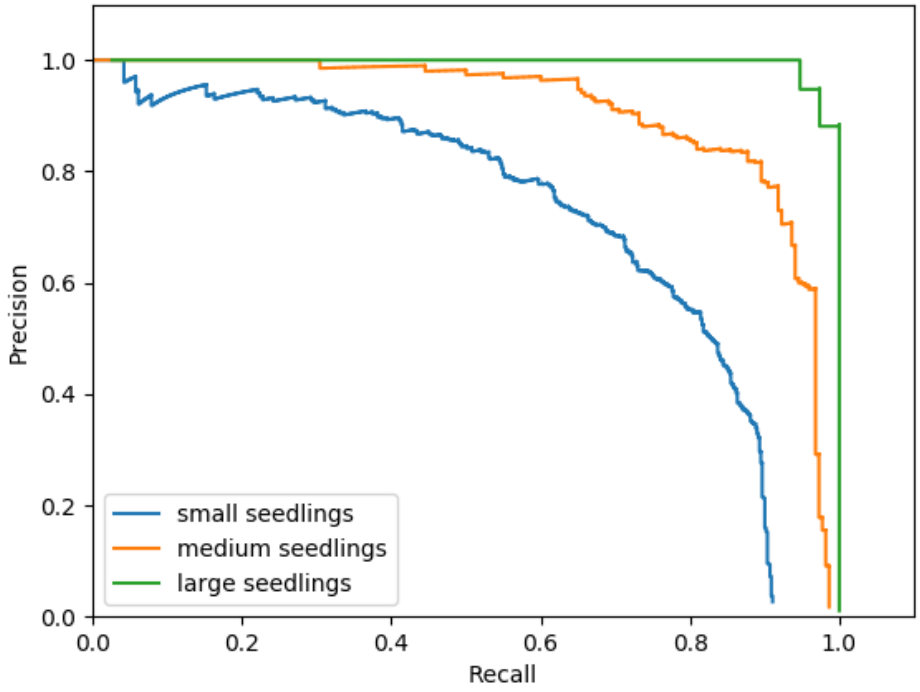

Figure 7.1: Precision-recall curves from Faster R-CNN for small, medium and large seedlings

To understand how the resolution influences the detection rate of the seedlings we evaluated 45 m and 105 m models using the divided evaluation set.

Object Detector	Small	Medium	Large	All
SSD (5 m)	0.43	0.79	0.99	0.58
SSD (45 m)	0.37	0.67	0.87	0.52
SSD (105 m)	0.31	0.72	0.86	0.45
R-FCN (5 m)	0.48	0.85	0.99	0.71
R-FCN (45 m)	0.58	0.80	0.98	0.68
R-FCN (105 m)	0.52	0.70	0.78	0.57
Faster R-CNN (5 m)	0.72	0.91	1.00	0.81
Faster R-CNN (45 m)	0.66	0.82	0.97	0.72
Faster R-CNN (105 m)	0.60	0.79	0.90	0.66

Table 7.2: Comparison of Faster R-CNN, R-FCN and SSD on the conifer seedling dataset depending on seedling size. The evaluation used MAP@[0.5]IoU as the metric.

The down-sampling of the images from 5 m to 105 m decreases the overall performance of each object detector by an average of 0.14 MAP. The decrease applies in the same way to all seedling sizes. All three object detectors are able to detect all large and medium seedlings reliably on the 105 m images. Only Faster R-CNN and R-FCN perform accurately on the small seedlings.

7.2 Overview of the results

Our main task in this thesis has been to analyze whether modern object detectors can extract meaningful information from drone footage. Our best model, a Faster R-CNN with ResNet-101 as a feature extractor pretrained on the COCO dataset, achieved an mean average precision of 0.81. This performance makes it possible to detect 8 of 10 conifer seedlings with an error rate of 20%. On large and medium sized seedlings the accuracy is nearly flawless. For simple tasks like counting the number of seedlings in a seismic line, studying the growth and survivability of conifer seedlings, and visualizing the distribution of the seedlings, this detector is suitable. Further modifications regarding the hyperparameters and introducing regularization is likely to increase the performance even more since overfitting on the training set occurred.

Our benchmarks showed that, even with few labeled data available, complex object detector architectures like Faster R-CNN are much better than small architectures such as SSD. Pretraining on these networks resulted in a significant performance increase, especially with little data.

Data augmentation techniques improved the performance of R-FCN and SSD

(especially on small seedlings) but had no influence on the Faster R-CNN network.

Even with a relatively small dataset of 4000 annotations, the Faster R-CNN architecture benefited from deeper convolutional neural nets such as ResNet-101.

We also found that using drone footage from a combination of summer and winter images yields the highest performance. The seasonal effect section showed that the summer dataset is somewhat easier for the object detector to handle than the winter dataset, despite the fact we had more problems labeling the summer data.

We addressed the question of the influence of flight height on object detection performance. In the section on resolution dependency, we saw that Faster R-CNN performs reasonably well even on down-sampled images similar to images taken from a flight height of 145 meters. This creates further possibilities to increase the amount of data by collecting it with a wide range of unmanned or manned aerial vehicles.

The training of object detectors does not rely on a massive amount of annotated data. Our experiment showed that Faster R-CNN performs well even on small datasets consisting of 200 annotations.

Chapter 8

Conclusion and Future Work

Topics for further study include the usage of instance segmentation instead of object detection. Instead of drawing a bounding box around objects instance segmentation attempts to classify each object with a dynamic mask. The mask assigns each pixel a value depending on whether it is object or background. The technique would allow much more precise localization of the conifer seedlings. Mask R-CNN [HGDG17] is an evolution of Faster R-CNN with an additional branch for predicting an object mask in parallel with the existing branch for bounding box recognition. It adds only a small amount of overhead to Faster R-CNN and replaces the RoIPool layer with a RoIAlign layer, which greatly improves the performance on small objects for object detection.

The usage of instance segmentation would make it possible to further add objects such as fallen trees or tipped trees which can be used to create a microsite for vegetation establishment and seedling protection, since, according to the restoration framework LIRA, they create a line-of-sight and movement barrier for wolves. Bounding boxes are not precise enough to capture diagonally positioned trees. A further improvement to the information extraction process would be the usage of a super-resolution network in front of the object detection pipeline.

As we saw in the resolution dependency section our object detection performance depends on the flight height of the aerial vehicle which captures the images. In [LBG+18] the authors used a convolutional neural net to perform end-to-end up-sampling. As input they used down-sampled images, and standard images as output. The network was able to learn the mapping from $40 \rightarrow 20m$ and respectively from $360 \rightarrow 60m$ GSD. This would enable the network to increase the resolution and sharpness of the images to further improve object detection performance.

Since labeling involved an enormous amount of effort because of the dense vegetation in the summer images, our evaluation set is not perfectly accurate. Small seedlings or partially covered seedlings might not be detected in the ground truth. GPS tagged seedlings from experts would improve the evaluation set. It would further allow the use of images taken at high altitudes, as such images are difficult to annotate manually.

List of Figures

List of Tables

Bibliography

[BMF+13] Julien Beguin, Eliot Mcintire, Daniel Fortin, Steven G. Cumming, Frédéric Raulier, Pierre Racine, and Claude Dussault. *Explaining Geographic Gradients in Winter Selection of Landscapes by Boreal Caribou with Implications under Global Changes in Eastern Canada.* In PloS one, 2013.

[DLHS16] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. *R-FCN: Object Detection via Region-based Fully Convolutional Networks.* CoRR, abs/1605.06409, 2016.

[EEG+15] Mark Everingham, S. M. Eslami, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman. *The Pascal Visual Object Classes Challenge: A Retrospective.* Int. J. Comput. Vision, 111(1):98–136, January 2015.

[GBC] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press. http://www.deeplearningbook.org.

[GBI+13] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay D. Shet. *Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks.* CoRR, abs/1312.6082, 2013.

[GDDM13] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. *Rich feature hierarchies for accurate object detection and semantic segmentation.* CoRR, abs/1311.2524, 2013.

[Gir15] Ross B. Girshick. *Fast R-CNN.* CoRR, abs/1504.08083, 2015.

[Heb17] Mark Hebblewhite. *Billion dollar boreal woodland caribou and the biodiversity impacts of the global oil and gas industry.* Biological Conservation, 206:102 – 111, 2017.

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. *Mask R-CNN.* CoRR, abs/1703.06870, 2017.

[Hor91] Kurt Hornik. *Approximation capabilities of multilayer feedforward networks*. Neural Networks, 4(2):251 – 257, 1991.

[Hui] Jonathan Hui. *Fast R-CNN and Faster R-CNN*. https://jhui.github.io/2017/03/15/Fast-R-CNN-and-Faster-R- CNN.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. CoRR, abs/1512.03385, 2015.

[LAE+15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. *SSD: Single Shot MultiBox Detector*. CoRR, abs/1512.02325, 2015.

[LBD+89] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. *Backpropagation Applied to Handwritten Zip Code Recognition*. Neural Computation, 1(4):541–551, 1989.

[LBG+18] C. Lanaras, J. Bioucas-Dias, S. Galliani, E. Baltsavias, and K. Schindler. *Super-Resolution of Sentinel-2 Images: Learning a Globally Applicable Deep Neural Network*. ArXiv e-prints, March 2018.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. *Microsoft COCO: Common Objects in Context*. CoRR, abs/1405.0312, 2014.

[RDGF15] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. CoRR, abs/1506.02640, 2015.

[RDS+15] 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*. Int. J. Comput. Vision, 115(3):211–252, December 2015.

[RHGS15] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. CoRR, abs/1506.01497, 2015.

[San] Leonardo Santos. *Object Localization and Detection.* https://leonardoaraujosantos.gitbooks.io/artificial-inteligence.

[SIV16] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning.* CoRR, abs/1602.07261, 2016.

[SLD17] Evan Shelhamer, Jonathan Long, and Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation.* IEEE Trans. Pattern Anal. Mach. Intell., 39(4):640–651, 2017.

[SLJ+14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. *Going Deeper with Convolutions.* CoRR, abs/1409.4842, 2014.

[SM86] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc., New York, NY, USA, 1986.

[TH12] T. Tieleman and G. Hinton. *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude.* COURSERA: Neural Networks for Machine Learning, 2012.

[Tzu15] Tzutalin. *labelImg.* https://github.com/tzutalin/labelImg, 2015.

[UvdSGS13] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, and A.W.M. Smeulders. *Selective Search for Object Recognition.* International Journal of Computer Vision, 2013.

[YCBL14] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. *How Transferable Are Features in Deep Neural Networks?* In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14, pages 3320–3328, Cambridge, MA, USA, 2014. MIT Press.

[ZC88] Y. T. Zhou and R. Chellappa. *Computation of optical flow using a neural network.* In IEEE 1988 International Conference on Neural Networks, pages 71–78 vol.2, July 1988.