Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Object detection in high resolution satellite images

Teymur Azayev

Supervisor: Ing. Michal Reinštein, Ph.D
Field of study: Cybernetics and Robotics
Subfield: Robotics
May 2016

Acknowledgements

I thank CTU for being such a good *alma mater*, Ing. Michal Reinštein, Ph.D and Anastasia Lebedeva for proofreading the thesis and suggesting improvements.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of the university theses.

Prague, date .....

.....
signature

Abstract

Given the abundance of various types of satellite imagery of almost any region on the globe we are faced with a challenge of interpreting this data to extract useful information. In this thesis we look at a way of automating the detection of ships to track maritime traffic in a desired port or region. We propose a machine learning approach using deep neural networks and explore the development, implementation and evaluation of such a pipeline, as well as methods and dataset used to train the neural network classifier. We also take a look at a graphical approach to computation using TensorFlow [13] which offers easy massive parallelization and deployment to cloud. The final result is an algorithm which is capable of receiving images from various providers at various resolutions and outputs a binary pixel-wise mask over all detected ships.

Keywords: machine learning, satellite images, remote sensing

Supervisor: Ing. Michal Reinštein, Ph.D
FEL, Katedra kybernetiky
Karlovo náměstí 1, 121 35
Praha 2

Abstrakt

Vzhledem k velkému množství dostupných satelitních snímků téměř z jakékoliv časti zeměkoule se potkáváme s úkolem interpretace těchto dat k extrakcí užitečních informací. V této práci se díváme na způsob automatizace detekce lodí pro sledování námořního provozu v zadaném regionu nebo přístavech. Navrhujeme přístup z hlediska strojového učení s použitím hlubokých neuronových sítí a vyšetřujeme vývoj, implementaci a ohodnocení takového algoritmu, spolu s metodami a datovou množinou, které jsou použité pro trénování klasifikátoru. Také se díváme na grafový přístup k výpočtu s použitím TensorFlow [13], který poskytuje možnost snadné masivní paralelizace a rozmístění na Cloud. Konečný výsledek je algoritmus, který je schopný přijímat obrázky od různých poskytovatelů v různém rozlišení a vytvářet binární masku nad všechny detekované lodě.

Klíčová slova: strojove uceni, satellitni snimky, dálkový průzkum Země

Contents

Part I
I

1 Introduction **3**
1.1 Motivation 3
1.2 Aims 3
1.3 Thesis structure 4

2 Related work **5**
2.1 Deep learning in image recognition 5
2.1.1 A short overview of traditional image recognition/analysis techniques 5
2.1.2 Deep neural networks 6
2.1.3 State of the art in object detection 8
2.2 Remote sensing 10

Part II
II

3 Methodology **15**
3.1 A naive approach using bounding boxes 15
3.1.1 Architecture 15
3.1.2 Dataset generation 16
3.1.3 Training and inference 16
3.1.4 A slightly different approach 16
3.1.5 Discussion 19
3.2 Feature extraction using VGG . . 20
3.2.1 Transfer learning 20
3.2.2 VGG architecture and training 20
3.3 Gridwise semantic labeling 21
3.3.1 Concept 21
3.3.2 Dataset generation 21
3.3.3 Architecture overview 21
3.3.4 Importance of context 22
3.3.5 Training and inference 23
3.3.6 Suggested improvements 24
3.4 Network analysis 27
3.4.1 Effective receptive field 27
3.4.2 Visualization 29

4 Computation as a graph **31**
4.1 TensorFlow 31
4.2 TensorBoard 32
4.3 Implementation 32
4.3.1 VGG implementation 32
4.3.2 Skip connections 33
4.3.3 Backpropagation algorithm . . 34
4.3.4 Optimization 34
4.3.5 Out of core training 34
4.4 Parallelization 35
4.5 Timing tests 35

Part III
III

5 Results & evaluation **39**
5.1 Image sets 39
5.2 Evaluation metrics 39
5.2.1 Total mean square error 40
5.2.2 ROC curve 40
5.3 Choosing a threshold 41

6 Discussion **43**
6.1 Network performance 43
6.2 Future work 44

7 Conclusion **47**

Appendices

A Bibliography **51**

Figures

2.1 Manually annotated examples: Object localization (bounding boxes), semantic segmentation (purple masks), instance segmentation (each ship has its own color to distinguish between instances). 5
2.2 Convolutional neural network architecture [1]. 7
2.3 Comparison of images from two different providers. 11

3.1 17
3.3 Positive samples for class 'ship' . 17
3.4 Classification results using bounding boxes 18
3.5 Poor performance due to tightly packed ships 18
3.6 Using the earlier approach as region proposals before applying blob extraction. Left most image shows positive samples trigger as 'ships' which are shown in red squares, followed by a binary mask of the triggered regions in black and white. The third image shows the saliency points found using the DOG algorithm and the last image shows those points filtered using the binary mask. 19
3.7 VGG architecture [52] . Each column, labeled with letters A through E represents the layers of that specific network. Each row denotes a layer, with the maxpool and FC (fully connected) shown as common for all the networks. The bottom table shows the total amount of parameters including the FC layers which we don't use. 22
3.8 Comparison of the variety of images 23
3.10 Mean squared error (MSE) extraction. The yellow region on the ground truth mask shows the region that we extract the cost from. . . . 23
3.11 Improved MSE extraction. The yellow square shows the valid regions from which we extract the cost. . . 24
3.12 Initial results for gridwise semantic labeling. Coarse output, innacurate. 25
3.13 *tanh* activation function. $tanh = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ 26
3.14 Gridwise semantic labeling architecture. Nodes represent operations (I/O, transformation, sub/up-sampling), arrows represent layers. Yellow blocks represent the VGG feature extractor. Blue blocks represent learnable convolution filters. Purple blocks denote upsampling and downsampling operations. Green blocks represent I/O. 26
3.15 MSE cost function of training and validation set of the architecture shown in Fig. 3.14 trained in batches of 60 for 1600 iterations (5 epochs). VGG feature extractor is left frozen and only skip connections are trained 27
3.16 From left to right: Random noise input, gradient mask at classification layer, gradient at image, natural log of gradient at image 29
3.17 Image patch (prior), annotated mask (the output pattern that we are trying to activate), gray scale optimized image, RGB optimized image 30

4.1 Simple code example in TensorFlow showing a computation of $y = ax^2 + bx + c$ along with the generated computational graph. Code example taken from [10] 32

Tables

3.1 Effective receptive field of neurons in each layer 28

4.1 Table showing forward + backward pass times for the various b microbatch sizes. 35

4.2 A screenshot of the TensorBoard opened in an internet browser showing part of the generated computation graph. On the top right corner are tabs to windows that enable us to see the histograms of our parameters and other useful information such as the training error and images. 33

5.1 Examples from Image set A. All shown patches are of size 500×500 39
5.3 Examples from Image set B. All shown patches are of size 500×500 40
5.5 ROC evaluation on imageset B using the 3.14 architecture. Legend shows the position on the curve of three thresholds. The blue dashed line shows the points where the FPR and TPR are equal. 41
5.6 Effect of different thresholds on classification. Threshold is denoted by T and ranges from -1 to 1. We can see that decreasing the threshold triggers pixels mostly around the ships. 42
5.8 Precision-recall break-even point on test set of size 10 from imageset **B** 42

6.1 TensorBoard screenshot showing training of the network on a harshly augmented dataset. We can see a flat period of cost function before gradients start to flow. MSE on the y axis and iterations on the x axis. . . 44
6.2 Comparison of the variety of test images from image set **B**. From left to right: Original image, Ground truth, raw prediction by algorithm. 45
6.4 Performance on test images from image set **A** with high cloud coverage. From left to right: Original image, Ground truth, raw prediction by algorithm. Precision-recall break-even point at 0.64. TPR: 0.62 , FPR: 0.008. No cloud covered images appear in the training set. 46

Part I

Chapter 1
Introduction

1.1 Motivation

We find ourselves surrounded by a plethora of technologies capable of interacting with the environment and gathering all types of data, ranging from motion to sound to visual data obtained by detecting various parts of the electromagnetic spectrum. A challenge then arises in the interpretation and extraction of useful information from various types of data. One of the most information-rich sources and arguably most difficult to interpret is visual data.

Nowadays we can take advantage of the growing amount of satellites in close-earth orbit equipped with high-end cameras to infer current information about the globe on a large scale, ranging from exploration to environmental, social and economic trends. Being able to analyze and interpret satellite imagery in an automated fashion can give us insights to useful information and enable us to make better decisions whether it concerns the business, environment or science in general.

1.2 Aims

The aim of this thesis is to design, implement and experimentally evaluate a deep neural network pipeline for detection and classification of objects in high resolution satellite images. The work includes algorithms and techniques for preprocessing the images as well as reasoning, methodology and algorithms for extraction of suitable training and test sets which are used for optimizing the proposed neural network architectures for the given task. The work is expected to receive large amounts of raw images and infer required information, such as the amount of area covered in the image by those objects. Consequently the detection and classification architecture is expected to be able to handle images from various providers and at various resolutions. It should also able to deal with non-ideal situations like excessive cloud coverage and defects such as severely over/underexposed images.

1.3 Thesis structure

The following chapter gives a brief history of the advent of deep learning [30] in image recognition as well as an overview of the theory and functionality of convolution neural networks which make up the heart of the classification algorithm of this work. This is followed by an overview of the field of remote sensing, including information about how satellite technologies are used to provide us with large amounts of rich visual data. Having that knowledge, the reader can get an understanding of the possibilities that emerge from having efficient automated methods of infering useful information from large amounts of raw data.

Part II focuses on detailed methodology and implementation of the pipeline as well as analysis of the functionality and mechanisms of the underlying neural networks in both training and inference. We also take a look at the implementation of the neural network in TensorFlow [13] and talk about the advantages of implementing such a structure as a computational graph rather than using conventional means.

Part III demonstrates the achieved results and methods that are used to evaluate the architecture and discusses possible improvements and other viable methods that could have been used to approach the task. It is then followed up by a conclusion.

Chapter 2

Related work

2.1 Deep learning in image recognition

2.1.1 A short overview of traditional image recognition/analysis techniques

Since the advent of the field of computer vision researchers have been developing various techniques which can enable a computer program to *understand* digital images. This concept is divided into various categories. The first one is image classification, which is the task of classifying images into a predefined category e.g. *person* or *airplane*. This usually involves using the information from the whole image and predicting one or several labels for that image. Object localization on the other hand is the task of predicting a bounding box around an object or multiple objects of interest in the image. A more recent and more difficult task in image recognition is semantic segmentation, which is basically a pixel-wise labeling of predefined categories in the image. This has been further developed into 'instance' segmentation where not only do we want to label each pixel belonging to a predefined category with a separate value, but we also want to distinguish between the multiple instances of those objects in the image.

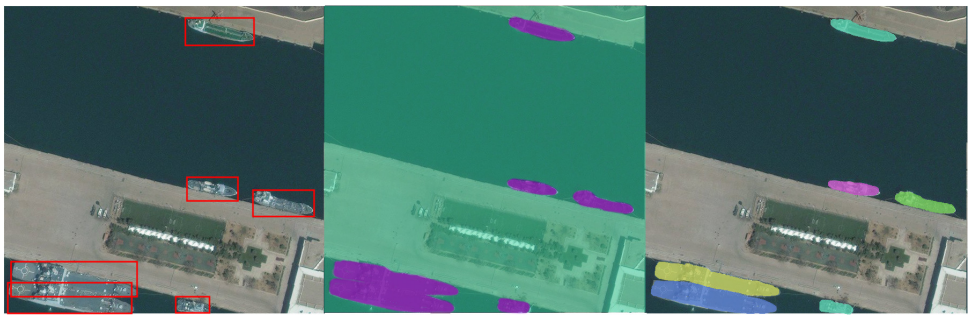


Figure 2.1: Manually annotated examples: Object localization (bounding boxes), semantic segmentation (purple masks), instance segmentation (each ship has its own color to distinguish between instances).

Over the years researchers have been developing various techniques to enable them to solve the above mentioned challenges. Some of them such as the viola-Jones face detection [58] algorithm and the histogram of oriented gradients method for pedestrian detection [48] were very successful and made their way outside academia. The main goal is to transform the raw pixel information into a more abstract representation, called a feature space, in which we can use a classifier to infer useful information about the image. Most of the techniques that achieve this consist of or include some form of convolving a kernel across the image to aquire a feature map and statistical regional pooling of information. In the Viola-Jones method they used Haar-features [58] which are basically weighted adjacent rectangular regions which were slid across the image, summing up pixel intensities in each region to extract useful representations. In one paper [48] the authors used histograms of oriented gradients as the primary feature extractor which was then eventually fed into a linear SVM classifier [17]. What the above techniques have in common is that they used shallow features that were hand-engineered by skilled people who have experience in the field and also often required domain-expertise. This often results in a fragile framework which only works well in a given task and fails if the conditions change. This also means that if the task changes even slightly, the whole feature extractor might have to be rewritten from scratch, which is very time consuming and expensive. These disadvantages left researchers in the field searching for a more robust and effective solution.

In 1998 Yan LeCun et al. demonstrated very effective handwritten digit recognition [34] using neural networks that used convolutional kernels as their transformation layers which allowed them to achieve over 99% accuracy in the digit recognition task and re initiated interest in research of using neural networks for image recognition. Unfortunately, at the time we were severely limited by lack of computational power and techniques to efficiently train neural networks to perform more complex tasks so it was largely abandoned for many years.

The real advent and rebirth of neural networks was in 2012 when Alex Krizhevsky et al. won the ILSVRC challenge [47] by a large margin with his 7 layer convolutional neural network [33], code-named AlexNet hence re-Christening the field with the name ‘Deep learning’. This success was largely enabled by improvements in the architecture and techniques used to train the networks and the progress and development of much more powerful compute resources; notably acceleration by GPU’s. From there on researchers have developed new techniques and architectures to further improve the adaptation of deep neural networks into vision recognition with vast amounts of success and applications in and out of academia.

2.1.2 Deep neural networks

The concept of neural networks has been around for quite a while, but as mentioned in the previous chapters there have been some serious limitations which discouraged their use and even completely excluded them from tasks such as visual recognition. In this chapter we will mention the changes in

architecture and methodology, which enable us to benefit from this paradigm in many fields, one of which is visual intelligence.

The notion of deep neural network simply extends on the model of the multilayer perceptron (MLP) [45], [46]. By 'deep' we simply mean that there is more than one hidden layer. One of the limitations of earlier fully connected MLPs when applied to images was that we required a vast amount of connections between every pixel and the next layer which severely limited their use due to computational bottlenecks. As a quick example, a MLP with an image input of size 256×256 and 256×256 hidden units in the first layer would require over 4×10^9 parameters for the first layer alone. Another issue with this architecture, is that we are not exploiting spatial information which is critical in visual data.

To adapt the notion of the MLP to visual data Yann LeCun et al. [35] popularized the convolutional transformation layer for neural networks, the concept of which had already been introduced in 1980 by Fukushima [24]. This technique used the idea of sliding a multiple square convolutional kernels over the entire image to produce so called feature maps which were fed into a non-linearity and then as input to the next layer. Using the same kernel for the entire image meant that we were using a significantly lower amount of parameters and performing less operations per layer as well as respecting the spacial information of the image. In between convolutional layers, max-pooling [1] (subsampling) layers were used which introduced an implicit invariance to translation of the image. The final convolutional layers were then connected to a fully connected [2] MLP which predicted output labels for the images.

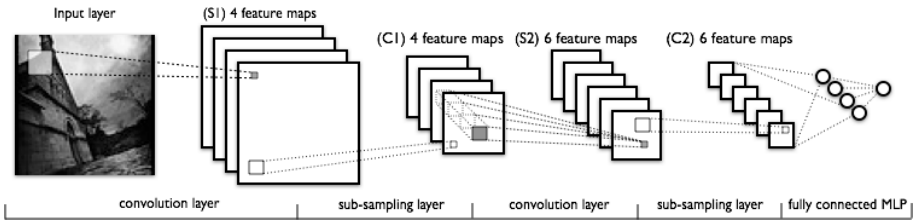


Figure 2.2: Convolutional neural network architecture [1].

There are only a handful of algorithms that can be used to train neural networks, the most efficient of which is a supervised method called Backpropagation [46] . The idea is that we perform a forward pass on the network and store the activations (values) of each neuron. We compare the output of the network with the Ground truth and calculate the error with respect to a predefined error function. We then backpropagate the error into every neuron and update the weights of that neuron in the process. In essence we

[1] Subsampling simply means to reduce the size of an image. In max-pooling the feature map is divided into a grid with 4 values in each square. The maximum value in each square is then selected for the next layer which now has a resolution of the grid that the previous image was divided into.

[2] Fully connected means that every neuron is connected with every other neuron of an adjacent layer.

are looking at how much each neuron is responsible for the error of the whole network, and correcting that neuron to minimize it.

A major hurdle that had to be overcome in training deep networks was the issue of preserving the integrity of the gradient whilst backpropagating it through the layers of the network. This is commonly referred to as the 'Vanishing gradient' problem [27] and describes the issue where the gradient diminishes with each passing layer to the point where it is almost zero when reaching the lower layers giving insignificant parameter updates at each time step. This results in the network not learning anything or taking excessively long to do so. Some attempts to remedy this were made by Hinton et al. by training the network layer by layer in a greedy fashion [28] in advance of training the whole network jointly. The issue of the vanishing gradient was largely improved with the introduction of Relu activation units by Alex Krizhevsky [33] and various papers describing proper parameter initialization, notably the Xavier initialization [27] . Various other techniques have been used/are being used such as batch normalization[33] which is a layer introduced after each convolutional layer which forces the activations to have a unit gaussian distribution. Almost all modern deep nets are trained with SGD optimization [50] algorithms with momentum [54] . Some notable of these include RMS prop [20] and ADAM [32] and will be mentioned in greater detail in Part II.

A large stride in the adaptation of neural network to various tasks and datasets is the concept of Transfer learning [59]. It turns out that the lower the layers in a deep neural network learn more general features to the point where the first layer almost always consists of gabor-like and blob-like color filters [59]. This means that a deep network can be trained on a large annotated dataset and then the lower convolutional layers can be reused on a different task as demonstrated in detail in [59]. This is a huge advantage since there are many tasks, such as this work, where only a very small annotated dataset is available and that training on it from scratch gives poor results.

2.1.3 State of the art in object detection

In this chapter we will take a look at some modern methods which use deep neural networks as classifiers for object detection in images as well as methods that fully exploit the architecture and the advantages of having a powerful feature extractor.

A classic simple technique of object localization in computer vision is to simply slide a window of various sizes at all possible locations in the image and classify the box at every step [19], [57]. This gives a result of multiple windows, called bounding boxes which can then be post-processed by an algorithm such as non-maxima suppression [39] to remove overlaps. The use of a deep neural network as a powerfull classifier for each window can lead to reliable results which was showed by Overfeat in 2013 [49] who got state of the art results on the ILSVRC localization and detection challenge. The obvious disadvantage of this is the very high expense of running a deep neural net for hundreds or thousands of windows per image. Overfeat [49]

improves on this by converting the fully connected layer of the neural net into a convolutional layer, enabling the network to be run on windows of various sizes. Thus for a given position we can run classification on bounding boxes of various sizes efficiently by reusing the computed feature maps, effectively sharing computation. Bounding box approaches unfortunately do not use the full potential of a deep neural net and unsuprisingly have been overshadowed by other more powerfull methods like R-CNN [44].

For single or a fixed amount of objects in an image one can formulate the localization problem into one of regression [55]. The idea is that we use attach a regression head to the last convolutional layer in a deep conv net and use it to regress coordinates and sizes of a fixed amount of bounding boxes and we use the rest of the trained network, the 'classification head' to classify the object present in each of those boxes. The disadvantages of this is that it is more difficult to adapt to a variable amount of present objects. An approach named YOLO (you only look once) [43] uses a similar technique by regressing a fixed amount of boxes and their confidence not for the entire image, but for each square in a fixed grid on the image. The results are then filtered and aggregated to produce the final bounding boxes.

Probably the most efficient and accurate method for object detection is a technique based on regional convolutional networks, R-CNN [26] . The idea is that we use a region proposal method such as EdgeBoxes [60] or Selective Search [56] to find regions of interest (ROI) in the image, which we would warp innto a fixed size and use a deep neural network to predict the class of the object that was proposed. This method was then improved by using the network to convolve the entire image [25] , producing a higher-level feature map onto which we then project the region proposals and use the projected area on the feature map to classify the object. Although this exploits the use of shared computation by only computing the feature map of the entire image once, we find that the bottleneck of the method is then in the external region proposal algorithms. A final improved version called faster R-CNN [44] uses a region proposal network [44] on top of the convolved feature map to predict scores for anchor-boxes, which are a fixed amount of n boxes of different shapes and sizes. These boxes are then used as regions which are classified by the top level classifier. This technique completely eliminates the need of external region proposals [44] and cleverly exploits the flexibility of neural network architecture, enabling almost realtime performance.

Another technique for object detection is using a segmentational approach. The idea is that we input the entire image into the network and predict a label map for each pixel. There are a few various approaches to this. One of them, is to convolve the entire image to produce pixel predictions all at once at various scales and then upsample and concatenate the results to produce a final pixel map [22]. When using a neural network to predict spatial information, we get a significantly smaller map than the input image due to the subsampling layers in the network. The authors of one work [37] use learnable upsampling layers which reconstruct the image from the high level extracted features. They also take advantage of so called skip-connections

[37], which are methods of using spacial information from higher resolution lower layers as well as lower resolution semantic information from higher-level layers to construct a prediction. Noh et al. [40] uses a VGG network to extract high level features from the image and an upside down VGG network to reconstruct the semantic map using unpooling (reusing spacial information from the pooling activation maps) and convolution-transpose. V. Mnih and G. Hinton used a similar approach [38] to label areal images.

An extension to semantic segmentation is instance segmentation where we are interested in segmenting and labeling the individual object instances in the scene. A notable work [18] which won the MS COCO 2015 challenge [36] works by identifying the instances of the objects in the scene by a region proposal network as in faster R-CNN and then generating a mask and segmenting the individual instances. The pipeline is similar to object detection. The semantic and instance segmentation techniques are significantly more difficult and require much more work and high-quality training sets [36].

2.2 Remote sensing

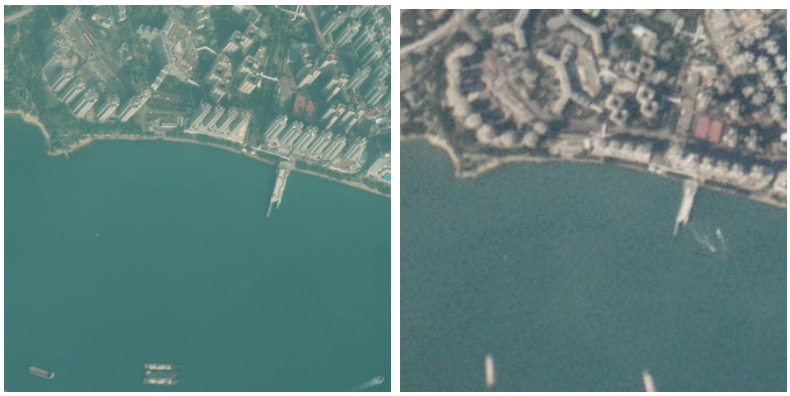
Remote sensing is the acquisition of information about an object without being in direct contact; usually by means of sonar or electromagnetic radiation. When talking about satellite imagery, we refer to the category of remote sensing named 'passive sensing' using the electromagnetic spectrum. This means that we are not sending out any signals or information, but simply observing the reflection of radiation from the sun using a specialized sensors mounted on the satellite. Satellite imagery has found many uses both in scientific fields such as Earth sciences and geology to economic, social and military intelligence niches.

Remote sensing satellites such as the GeoEye[5] and WorldView [12] are launched into low-earth orbit, which lies at around 600-800km from the surface. The satellites are deployed into circular sun-synchronous near polar orbits which means that they pass the equator multiple times a day and stay at roughly the same angle in respect to the sun which makes it ideal for earth imagery. At this distance the orbital periods range at about 90 minutes [5] and satellites can revisit the same points on earth in intervals of 1.1-3.7 days [12].

The satellites are equipped with state of the art stabilization and sensor technologies that enable them to acquire accurate high precision images of the desired location. The onboard sensors consist of multispectral RGB systems, panchromatic sensors of typically higher resolution, IR sensors which give additional temperature information and even super and hyperspectral systems with over 10 and 100 channels respectively for monitoring precise geological changes. Resolutions of the sensors typically range from 0.5m to 1.6m per pixel with 0.5m being the legal limit in most countries with the exception of the US government, who allowed image acquisition at limitless precision and commercial sale of images at 0.25m panchromatic [11]. Additional information from the channels could potentially make the task of object recognition easier,

but it is unfortunately more difficult to adapt because of limitations of transfer learning [59]. Annotated datasets are difficult and expensive to obtain and most of the times are not even available to the public. There simply aren't networks that are trained on multi channel inputs other than RGB and grayscale.

The images used in this thesis come from two major providers: Digital Globe [3] and Planetlabs [8]. The former owns and operates various major satellites, including the GeoEye and WorldView I,II and III and provides digital images to many big customers such as Google and Apple. These images range in resolutions from a few meters , down to 0.5 meters per pixel. Planetlabs, on the other hand own a constellation of smaller Cubesats [2] which provide earth imagery at 3-5 meters per pixel. They also differ by their repeat rates and pricing. Figure 2.3 shows a comparison of two images of the same port from two different providers at different resolutions.



(a) : Sample from Digital Globe size: 2000×2000

(b) : Sample from PlanetLabs size: 500×500

Figure 2.3: Comparison of images from two different providers.

Part II

II

Chapter 3

Methodology

3.1 A naive approach using bounding boxes

The objects that we are trying to detect are any kind of water vessel (ship) in the image that is discernible by a human being. The most basic way of handling such a task in computer vision is to use the sliding window technique. As mentioned in section 2.1.3, the basic idea is that we slide a window across the image at a fixed stride. At each iteration we extract the patch corresponding to the location of the window in the image and use some classifier to classify that patch. If it is classified as the object of interest then we simply state that the object of interest exists in that patch. Usually windows of several sizes and aspect ratios are applied to detect objects of interest in various scales and poses. It could be said this method is basically a brute-force approach because we are trying windows of many sizes and we are trying them everywhere. Initially we will only use window of size 128x128 pixels and a fixed stride of (64,64) . A window that is too small does not fit larger ships inside it and does not have enough contextual information, whereas a window that is too large gives too much ambiguity as to what is being classified.

3.1.1 Architecture

We use a simple convolutional neural network (convnet) as a robust classifier for the sliding window approach in the task of detecting ships. With the following architecture we attempted to classify the image into 3 classes: *ships*, *sea* and *land*. For testing we bundle the classes *sea* and *land* into a single *background* class. The net consists of 3 convolutional layers with 32, 64 and 128 feature maps respectively, followed by dropout [53] after each activation function with a keep probability of 0.6. The final feature maps are flattened into a 1-dimensional vector and fed into two fully connected layers 32 units wide, also each with dropout after the activation with a keep probability of 0.8. The output consists of a vector of size 3 which is fed into a softmax activation.

3.1.2 Dataset generation

The dataset is generated manually. An application written in python allows the user to open a desired image in an interactive figure, select a desired class, save path and start clicking/dragging across the image to select patches for the training set. Care has to be taken not to gather false/ambiguous samples as they can negatively influence the training process. Using this method a set of about 3000 patches of 128x128 for each class from about 12 different images from image set **A** 5.1 were obtained giving a total set size of about 9000 patches. This set is also augmented at training time by random horizontal, vertical flips and by adding various types of noise to the image [33]. The feature space size then becomes $[128 \times 128 \times 3 \times 256]$ assuming RGB input and 8bit pixel color depth.

3.1.3 Training and inference

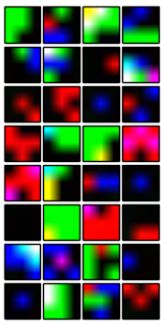
The architecture is trained with a gradient descent optimizer ($\alpha = 10^{-2}$) with momentum [54] with a batch size of 48 for 4500 iterations which is about 25 epochs. On TensorFlow this takes approximately 1 hour on a low-tier machine with GPU acceleration. The dataset is split with a ratio of 9 to 1 of training to validation respectively. For every port we subtract the mean for each color channel before dividing it into patches. No other preprocessing is done. On the test set of 1000 random patches extracted from image set **A** this architecture achieves a true positive rate (TPR) of 92 % and false positive rate (FPR) of 3 %. However, on a test set of 1000 random patches extracted from image set **B** we get a TPR of 60% and FPR of 7% .This experiment is done to get an idea of how a 5-layer convnet will perform on such a dataset. It showed that the network is not able to generalize well, i.e. doesn't perform well on examples that are very different to the ones that it has been trained on. This is obviously due to the small, low variance training set that we have used and because we are training the network from scratch.

Taking a look at the filters of the first layer in Fig. 3.1 we can note that there are duplicate or similar features and generally there is not much variance in the range. This is understandable since the network was trained on a relatively small set which was taken from less than 12 ports. We can compare the filters to the first layer of the VGG [52] network trained on ILSVRC [47]. The pixels in the image are interpolated because it makes it arguably more interpretable by the human eye.

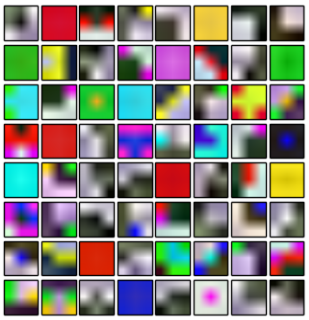
We can see from Fig. 3.1 that the weights trained from scratch exhibit a much smaller variance of colors and have similar or redundant patterns such as the filter with a single blue dot in the middle which appears almost 4 times.

3.1.4 A slightly different approach

A more targeted way of tackling the issue of object detection is to use a region proposal method which detects salient points (points of interest) in



(a) : First layer weights of the trained network



(b) : First layer weights of the VGG 16 D network

Figure 3.1:

the image and then attempts to use a more narrow classifier on those regions. This is in contrast to trying a window everywhere in the image, as in the first approach. As our saliency point detector we use the Difference of Gaussians (DOG) algorithm with a sigma range of 3 to 5 and threshhold of 0.02 which were found by trial and error to trigger on most of the objects of interest. The range is limited to only small blobs. The reason for this is that there is a large variety in size of the ships and large blobs falsely trigger on clusters of smaller ships and other larger structures which do not give us any useful information. After finding the salient points we attempt to classify each point of interest with a narrower rectangular classifier.

We use a window of size 86×26 pixels which we rotate in $12°$ increments at each point of interest. The size of the window was decided by manually measuring 50 randomly selected ships from various ports and taking an average. The classifier is trained to classify a whole (or most) of a ship that is reasonably aligned ($\pm 15°$) with the window against a 'background' class. Fig. 3.3 shows some positive samples for class 'ship'

Figure 3.3: Positive samples for class 'ship'

It is already ominous from Fig. 3.3 that the classifier does not have enough context 3.3.4 to make an accurate prediction in many cases. Fig. 3.4 shows

an attempt of detection using this method.

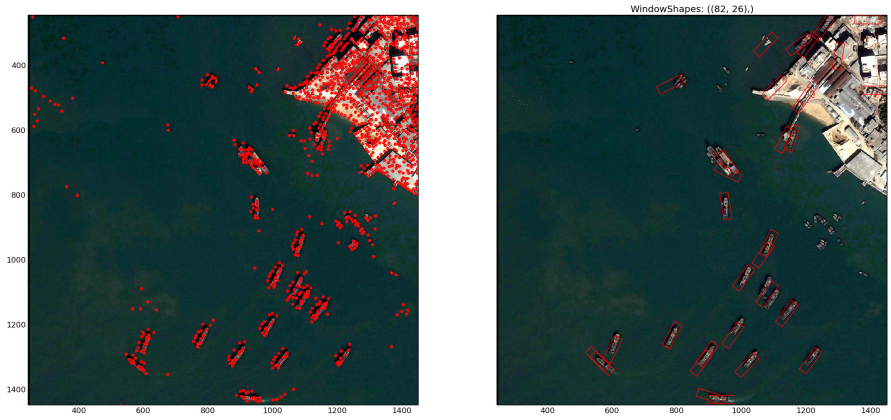


Figure 3.4: Classification results using bounding boxes

Due to multiple redundant detections at each point we use a Non-maxima suppression algorithm [39] modified to allow for rotated boxes. The result is unsatisfactory due to many reasons. First of all, the ships vary heavily in size and trying many different boxes for each is computationally infeasable. Due to the fact that there are regions of the image with many ships of different sizes stacked against each other, this also means that loose boxes might shadow smaller ships which eludes their detection.

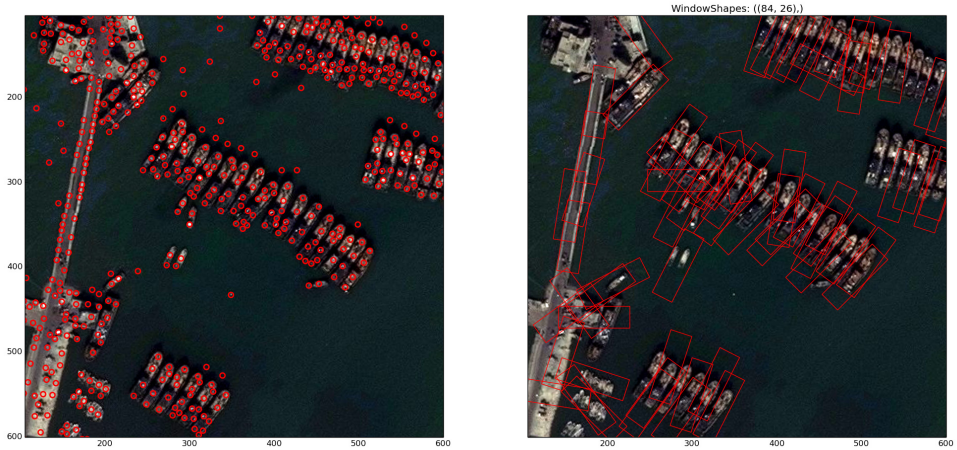


Figure 3.5: Poor performance due to tightly packed ships

Another issue is that the points of interest are not necessarily in the middle of the ship and sometimes there is no box that sufficiently captures the entire ship. As mentioned above, the smaller the box the less context, which means that the predictions are much less reliable. As a consequence of this, we see many false positives on land regions of the image. Finally, the search of salient points of interest can take up to a few minutes on a 1000×1000 image, which can put the total processing time for an image to over 10 minutes.

Making use of the first approach of classifying wider regions we can alleviate some of the above issues. The idea is that we first pass the convnet over the image and create a binary mask of regions that have been positively classified to have ships in them. We then multiply the mask with the original image, effectively performing a logical 'and' function and then apply the above algorithm. This significantly reduces the amount of computation time on the DOG algorithm, the amount of patches that have to be tried using the expensive neural network and also many of the false positives on land. It must be noted that the binary mask is first blurred using a Gaussian filter with a $\sigma = 30$ to smooth out the edges of the mask to prevent the DOG algorithm triggering along them.

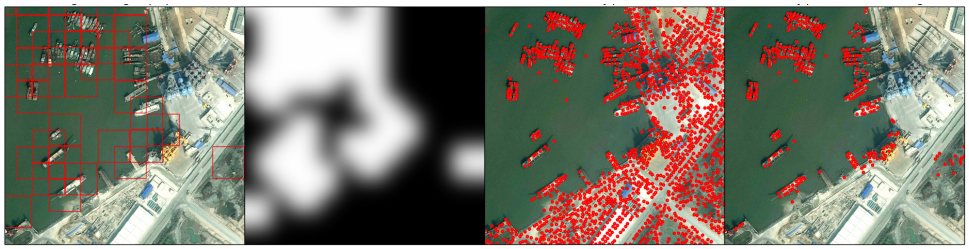


Figure 3.6: Using the earlier approach as region proposals before applying blob extraction. Left most image shows positive samples trigger as 'ships' which are shown in red squares, followed by a binary mask of the triggered regions in black and white. The third image shows the saliency points found using the DOG algorithm and the last image shows those points filtered using the binary mask.

3.1.5 Discussion

We can clearly see that the above methods are not the best choice to detect objects in these types of images due to the many disadvantages that they present. The external region proposal algorithms are unacceptably slow for large images and unreliable due to the variety in the statistics of the images. The results also need a substantial amount of post processing to remove redundant hits as well as a lot of manual tuning to make it work. The above methods also do not make use of the full potential of neural networks, but rather use it as a classifier with a confidence rating. Taking into account our initial task of tracking maritime traffic, we realize that it is not entirely necessary to find segment each ship and draw a bounding box around it. Instead we could simply find the amount of area of the image covered by ships using other methods, such as semantic segmentation. Due to the above reasons we did not pursue or look deeper into more advanced methods that can ameliorate some of the issues.

One more very important thing to note is that the classifier is poor due to the small size and poor variance of the set and training a deep neural network on such a dataset does not usually produce good results due to overfitting. In the next chapter we take a look at a method to overcome these issues.

3.2 Feature extraction using VGG

3.2.1 Transfer learning

The notion of transfer learning has been around for over 2 decades [41] in the realm of machine learning and basically describes the ability to use information gained from one task and applying it to another. In deep learning this is used very often. It has been found that the features that are learned by the convnet in the first layer are always the same and resemble color blobs and gabor filters [59]. This implies that the low level features are general and understandably the higher up we go in the network the more task-specific the features are [59]. Yosinski et al.[59] also quantifies the degree to which a particular layer is general or specific and generally described the transferability of features between tasks using convnets.

The idea of using transfer learning in deep learning is the following: We initially train a neural network on a high quality and large size dataset such as Imagenet [21] . This forces the network to learn a wide range of features that help it perform that task that it was trained on. We assume that the features in the lower layers of the network are general and can be 'transfered' to our task. We use the lower part of the trained network as a feature extractor on top of which we can train another neural network or linear classifier such as SVM [29] for other regression or classification tasks. The work of Razavian et al. [42] shows state of the art results in using this technique.

3.2.2 VGG architecture and training

In this thesis we have chosen to use a VGG 16-layer network D [52] as our feature extractor. This network gets an error of about 8.5% on the ILSVRC [47] . This is about a full 1% higher than the 19 layer version but in interest of easier handling and computation speeds the 16 layer net has been used for most of the work. The VGG has been chosen over Alexnet and other architectures for its simplicity, consistent 3x3 convolutions and depth, giving hope that we can tap from more general features. VGG was chosen over Resnet, the state of the art at that time with about 3.5% on the ILSVRC once again in interest of simplicity and computational flexibility.

The VGG network was trained (by the original authors) on the well known ILSVRC-2012 dataset from Imagenet [21] which consists of 1.3 million images, divided into 1000 classes which makes it a good feature extractor. The authors also used various type of image augmentation during training such as scale-jittering.

3.3 Gridwise semantic labeling

3.3.1 Concept

In this chapter we will take a look at a different approach to detecting desired objects in satellite images. The method draws on the concept of semantic segmentation and uses the structure of the deep neural network to make a prediction in image space in contrast to simple classification as in the previous approach. The input image is fed to the network by $m \times n$ patches and the network outputs a prediction grid for each patch. The prediction grid is essentially a binary mask classifying each pixel as to whether the image at that point contains the object or not. The grid is of a lower resolution than the actual image itself and it is projected into image space by simple resizing by an appropriate factor.

3.3.2 Dataset generation

The training set for the initial experiments consists only of 10 ports from image set **B** . The task is to detect ships of all shapes and sizes present in the image. The images are annotated manually using GIMP [6] image manipulation software. A binary mask is applied on top of the image using a simple paint brush and then exported separately. The image-mask pairs are then loaded and random $m \times n$ patches are extracted with random rotations and random scale-jittering [33]. The per-image mean is subtracted from each image before the patches are segmented. The patches from various ports are then randomly shuffled and saved as 1000-sample patches for fast loading during training. The set is split into training - validation sets with a 9 to 1 ratio, giving a total of 9000 training and 1000 validation samples. During training, the convnet is evaluated on the entire evaluation set every 50 iterations. The necessary use of the entire validation set is due to the sparse presence of ships in many regions and taking smaller batch sizes gives noisy and misleading evaluation results. During training the patches are also randomly flipped horizontally and vertically to synthetically expand the dataset.

3.3.3 Architecture overview

We make use of the VGG network as a powerful feature extractor for the task of the gridwise semantic segmentation. The feature extractor consists of 13 convolutional layers which feature 4 pooling layers in between as can be seen in Fig. 3.7. One should note though that the Number of parameters shown in the figure for each network includes the last 3 fully connected layers. Without them, network D has only 14 710 464 parameters. We initially tapped the feature maps at layer 10 which for a patch size of 128×128 gives 512 feature maps of size 16×16 . These are then fed into 2 more convolutional layers with 64 feature maps each and filter sizes of 3×3. The final layer has only 1

feature map which corresponds to the prediction. At this point no activation function was used at the final layer.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Figure 3.7: VGG architecture [52] . Each column, labeled with letters *A* through *E* represents the layers of that specific network. Each row denotes a layer, with the maxpool and FC (fully connected) shown as common for all the networks. The bottom table shows the total amount of parameters including the FC layers which we don't use.

3.3.4 Importance of context

By context we basically mean everything that surrounds and helps us classify an object of interest. An example of context surrounding ships in a satellite image would be a homogeneous, sometimes wavy surface (sea), other ships and textured edges that look like a port dock.

In terms of the input to the neural network, the context is the entire region that the patch represents. One of the major things that differ this task from other object detection tasks is that although the images are of high resolution most of the time, the objects of interest are sometimes very small and hard to discern, even for the human eye. Another issue is that the images are taken by satellites at different at varying lighting conditions and in various places on earth, which means that we cannot depend on natural landmarks to take their assumed color. The color of the sea, for example, can vary anywhere from a shade of gray, through blue, to brown, as can bee seen in Fig. 3.8. Because of this, adequate image context around the object is critical for successful inference.

(a): (b): (c): (d):

Figure 3.8: Comparison of the variety of images

3.3.5 Training and inference

The way we implement the notion of context into our architecture is that we do not design the network to predict a map for the whole patch of $m \times m$, but instead we design it to predict a sub-patch of $(m - q) \times (m - q)$ where q is the amount of pixels left on each side of the prediction window which acts as the 'context'. This is shown in Fig. 3.10

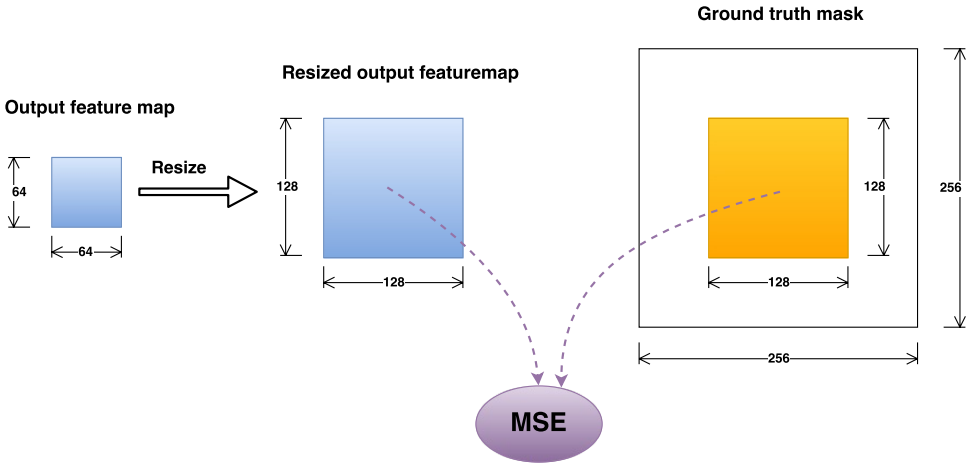


Figure 3.10: Mean squared error (MSE) extraction. The yellow region on the ground truth mask shows the region that we extract the cost from.

A naive way to do this is to simply state that since all the neurons in the final feature map f_{m_o} are differentiable functions of the input and that we can simply draw an error function as shown in 3.1 and 3.10.

$$mean((f_{m_o} - P(q : m - q, q : m - q))^2) \tag{3.1}$$

where P is the ground truth binary segmentation mask corresponding to the whole input patch S. It turns out that although this should work in theory, the network has hard time adapting and does not train very efficiently, or even fails to converge.

The above problem is mostly because we are only training a few layers on top of a fixed deep network whose features in the convolutional layers correspond spatially to the input. By trying to undo this in the last few layers we are not respecting the structure of the feature maps and therefore getting

poor results. This can be resolved by adding the following modification to the architecture: Instead of drawing an error function from the entire output layer f_{m_o} we treat f_{m_o} as the prediction of the entire input patch S but penalize only the predicted pixels that correspond to the projection of the ground truth sub-patch P(q:m-q, q:m-q) onto f_{m_o}. This is shown in Fig. 3.11.

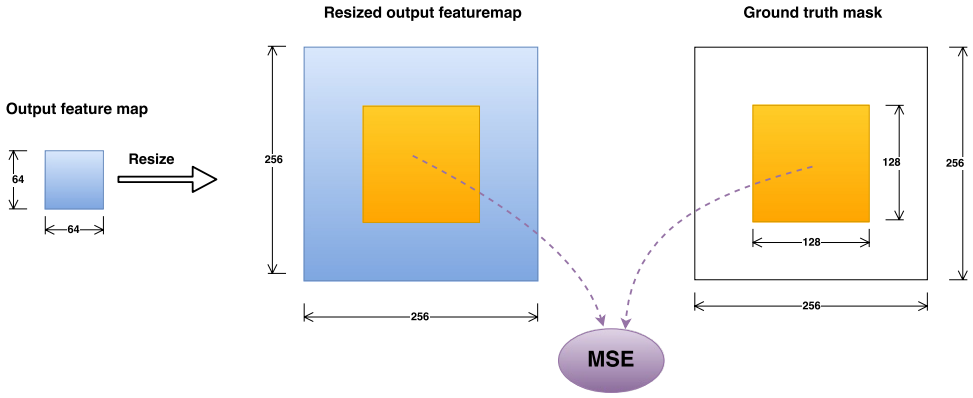


Figure 3.11: Improved MSE extraction. The yellow square shows the valid regions from which we extract the cost.

So far we have looked at the error function as mean of the l_2 distance between the prediction and the ground truth, with the output layer fitted with a *tanh* 3.13 activation function. We could also treat the problem as a pixelwise class detection problem where the final feature map f_{m_o} is of dimension $i \times j \times k$ where k is the amount of predicted classes (in this case 2). We can then use standard classification metrics such as crossentropy on a softmax activation which are superior to l_2 metrics [7] for classification. The initial results of this work, however, use a mean squared error (MSE) metric. Fig. 3.12 shows an example of the result using the above setup. The setup was trained on the annotated dataset with a batchsize of 60 for a total of 1400 iterations. A visual inspection of the result of a few test images showed that output prediction is too crude and lacks structure.

3.3.6 Suggested improvements

Two simple ways how to improve the above architecture for gridwise semantic labeling are to increase the patch size and to decrease the size of the prediction space in relation to the patch itself. We use a window size of 256×256 and predict a grid of only 128×128 which is centered in the patch. This gives us an arguably adequate context of 64 at least pixels on either side of the prediction grid.

Another clever technique, inspired by [37] is to combine the features of the network from various layers, called 'skip connections' [37] . Due to the multiple downsampling layers in the network, a lot of the spacial information about the original image is lost at the higher layers due to their low resolution. To preserve the output structure information and predict a more accurate output mask we can use the higher layers for semantic information and the lower

Figure 3.12: Initial results for gridwise semantic labeling. Coarse output, innacurate.

ones for structural information. In the above architecture we tap the network at the 10th convolutional layer. We expect that additional information from the 13th layer will help the network determine whether the objects in the scene are ones that we are interested in and that the additional information from the 7th convolutional layer will help us draw a more accurate grid. The feature maps at layers 7,10 and 13,which we will denote as l_l, l_m, l_u (with the subscripts meaning 'lower', 'mid' and 'upper') for a patch of 256×256 are 64×64 , 32×32 and 16×16 respectively. This means that layers l_m and l_u will have to be upsampled (we use bilinear interpolation, compared to the learnable upsampling in [37]) to the size of l_l. The feature maps from layers l_l, l_m, l_u are then taken separately and convolved twice using 3x3 convolutions to produce 64 feature maps at each layer. The final feature maps from all three layers are concatenated and convolved using a 1x1 filter to produce a map in the prediction space. We add a *tanh* 3.13 activation on the output to limit the prediction to a range of $\{1, -1\}$. It was experimentally found that predicting the pixels with tanh in the range of $\{1, -1\}$ provides superior results and trains better than predicting in the range $\{0, 1\}$. Fig. 3.14 shows the final architecture with the combined feature maps from layers 7,10 and 13. The Relu activation functions are omitted for brevity.

The final architecture was trained in batches of 60 for 1600 iterations (5 epochs) by freezing the VGG extraction layers and training only the skip connections 3.15. The training error is not too far from the validation error which is good because we know that we are not overfitting the dataset. It should be noted, however, that the validation set and training sets are very similar to each other so this is expected.

It was experimentally found that after training the newly added convolutional layers, the performance was significantly increased by freezing the new layers and fine-tuning the last 3 convolutional layers of the VGG network for 1000 iterations using a smaller learning rate, followed by a last joint training of the new layers and the last 3 convolutional layers for another 3000

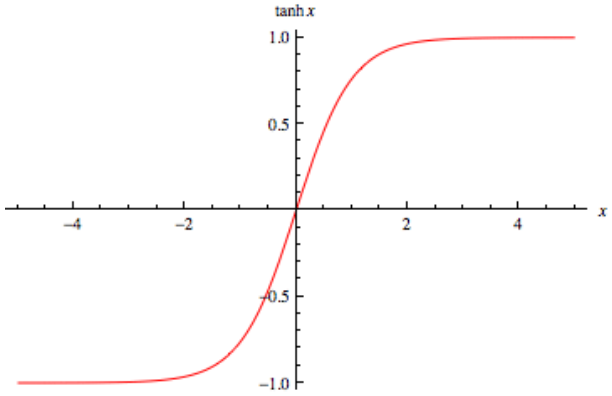


Figure 3.13: *tanh* activation function. $tanh = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

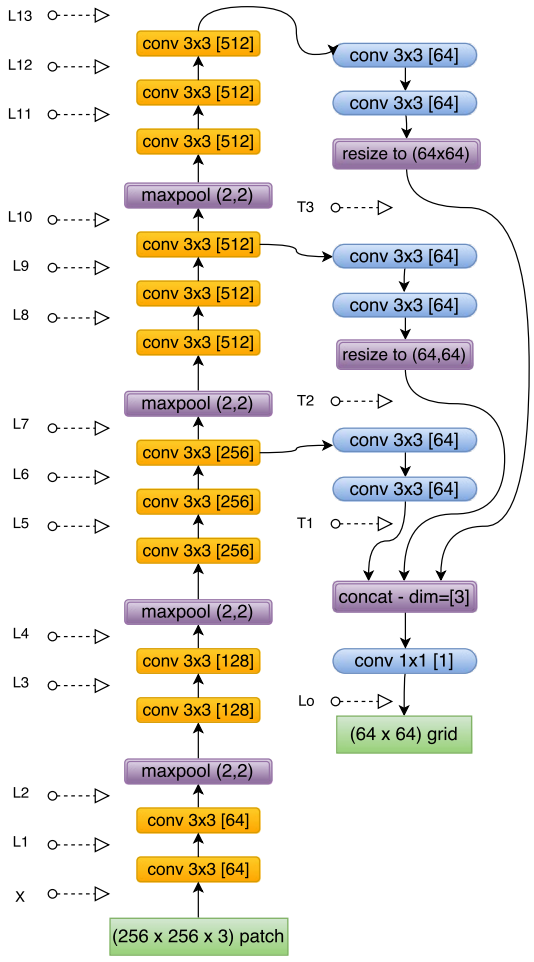


Figure 3.14: Gridwise semantic labeling architecture. Nodes represent operations (I/O, transformation, sub/up-sampling), arrows represent layers. Yellow blocks represent the VGG feature extractor. Blue blocks represent learnable convolution filters. Purple blocks denote upsampling and downsampling operations. Green blocks represent I/O.

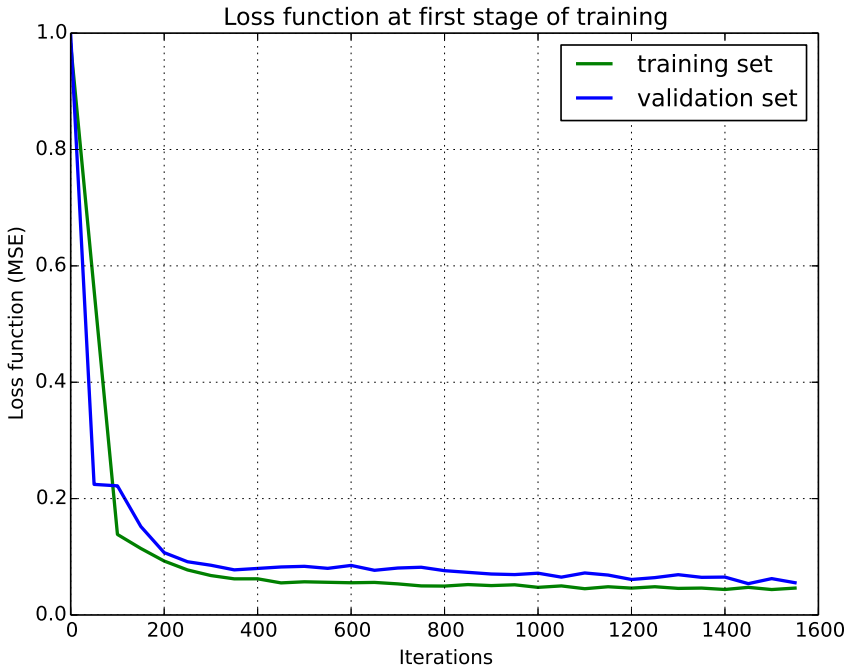


Figure 3.15: MSE cost function of training and validation set of the architecture shown in Fig. 3.14 trained in batches of 60 for 1600 iterations (5 epochs). VGG feature extractor is left frozen and only skip connections are trained

iterations. Fig. 6.2 shows some results using the architecture with the above improvements.

3.4 Network analysis

In this section we take a look at some interesting details of the network.

3.4.1 Effective receptive field

A key question asked when designing such an architecture is whether the network has enough information to make a reasonable prediction in the task that it is designed for. When using a neural network for image classification we use the information from the entire image to make a prediction. In our task what we are basically doing is asking the network to make a prediction for each output pixel based on the input image. For this to be possible we have to make sure that the effective receptive field (ERF) of each final neuron is looking at sufficient spacial information in the input image to make a prediction. In other words, we want to examine the projection of each neuron into pixelspace.

One way to do this is to mathematically examine the equations that lead up to the output of that neuron. Since we are using 3×3 convolutions we

can say that a neuron in a feature map in layer l_i has an effective receptive field of 3×3 on the layer l_{i-1}. Since we are using maxpooling with strides of $(2, 2)$ throughout the architecture we can say that pooling doubles the ERF of the neurons after the pooling in respect to the previous layer. We can then attempt to form a table of the ERF at each layer.

Layer	OP	ERF
l_1	conv 3×3	3×3
l_2	conv 3×3	5×5
l_{2_p}	$maxpool(2, 2)$	10×10
l_3	conv 3×3	12×12
l_4	conv 3×3	14×14
l_{4_p}	$maxpool(2, 2)$	28×28
l_5	conv 3×3	30×30
l_6	conv 3×3	32×32
l_7	conv 3×3	34×34
l_{7_p}	$maxpool(2, 2)$	68×68
l_8	conv 3×3	70×70
l_9	conv 3×3	72×72
l_{10}	conv 3×3	74×74
l_{10_p}	$maxpool(2, 2)$	148×148
l_{11}	conv 3×3	150×150
l_{12}	conv 3×3	152×152
l_{13}	conv 3×3	154×154

Table 3.1: Effective receptive field of neurons in each layer

The tapped layer l_{t_u} receives feature maps from the highest layer and performs two more convolution operations, which means that each neuron in the output has an ERF of only 158×158. This is sub-optimal since we are using 256×256 patches. This also means that larger objects that are partially outside the ERF of that particular region might get misclassified.

3.4.2 Visualization

To get a clearer and more accurate view we can use the network to visualize the ERF of a particular output neuron. One way to do that is to perform a forward pass on the network until we reach the desired neuron and then find the gradient 3.2 on the image S with respect to the neuron q_{ij} in layer l.

$$W = \frac{\partial S}{\partial q_{ij}^{l}} \tag{3.2}$$

By doing that we can see which part of the image influenced the activation of that particular neuron. We can implement equation 3.2 by nulling the gradients of all the neurons in the layer except of the one that we are interested in, and then backpropagating the gradient back into RGB pixelspace. We then perform an argmax on the absolute value of the RGB channels 3.3 to get a heatmap of the gradient [51].

$$M_{ij} = max_c |W_{h(i,j,c)}| \tag{3.3}$$

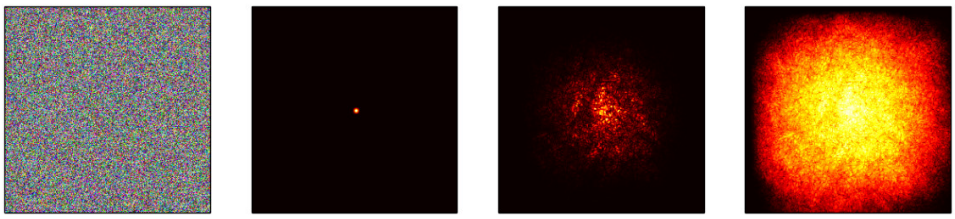


Figure 3.16: From left to right: Random noise input, gradient mask at classification layer, gradient at image, natural log of gradient at image

In Fig. 3.16 we can see the extent of the receptive field for a single output 'pixel'. Taking the natural log of the heatmap shows that the field reaches out across most of the patch but decays fast from the corresponding midpoint. This 3.16 is a curious result because it shows us that the ERF is larger than what we calculated in table 3.1.

We can use another visualization technique to get a rough glimpse as to what features in an image make it activate a certain output. We can do this using the same supervised backpropagation algorithm as we do for optimizing the weights of the neural network, except that in this case the weights are kept static and instead we optimize the image. In essence we are trying to find an image that most activates a certain output pattern by minimizing the error on the prediction.

In Fig. 3.17 we fed in a patch as an image prior and the annotated ground truth mask. The third and fourth images show the patches that would maximally activate the given mask after 300 iterations of minimization using the ADAM [32] optimizer 4.3.4 in TensorFlow. A learning rate of $\alpha = 0.01$ was used as well as l_2 regularization on the image so that optimized values do not grow too large.

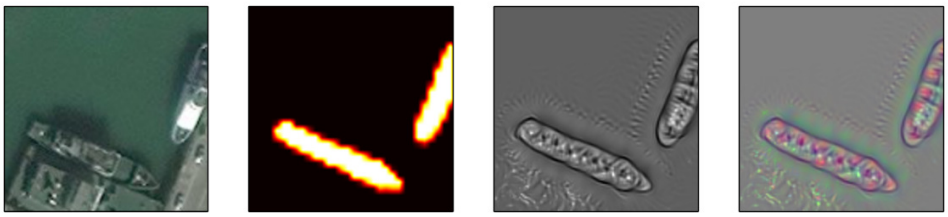


Figure 3.17: Image patch (prior), annotated mask (the output pattern that we are trying to activate), gray scale optimized image, RGB optimized image

We can see that the network looks for a well-defined border and prefers to see wave-like background around the ships. Interestingly enough it cares about the wavy property only up to a certain proximity to the ships. Using these images we get a rough idea of what our algorithm is looking for in the image to classify an object as a ship.

Chapter 4

Computation as a graph

In this chapter we will take a look at the software that the experimental detection pipeline was implemented in and talk about an approach to computational algorithms in TensorFlow that is different to the traditional procedural programming paradigm.

4.1 TensorFlow

The experimental object detection pipeline is almost entirely implemented in TensorFlow [13] using a python front end API. TensorFlow is an open source machine learning framework by Google which allows the user to quickly and efficiently implement various algorithms. This framework was chosen over other well known and widely used frameworks such as Caffe [31] and Torch [16] for various reasons. One of them is the wide choice of functions and operations that the framework provides, ranging from simple convolutions to whole optimizers which can be applied and adapted anywhere with a single line of code. Another reason is the potential to parallelize the computation on a heterogeneous set of devices which is made possible by the architecture of the framework.

As mentioned in the introduction, TensorFlow uses a different approach to the standard imperative programming paradigm in which the user writes code which is sequentially executed. Instead, in TensorFlow the user declares the computational architecture as symbolical operations and equations which are then compiled into a computational graph and is then run as many times as required. This type of computation is already implemented in Theano [14], [15] and is improved upon in TensorFlow.

The computational graph in TensorFlow is a directed graph which consists of Nodes and edges. Nodes are operations which take in and output tensors. Almost all operation kernels are defined as both CPU and GPU implementation so that they can be executed on the appropriate device depending on the data. Edges are tensors themselves or special dependency control edges which do not contain data [13]. By tensors we mean multidimensional arrays of various supported data types. Most tensors are immutable and do not live past a single execution. TensorFlow also provides variables, which are tensors that live on the appropriate device, minimizing unnecessary memory

transfers. Since the architecture is compiled into a computational graph consisting of basic operations, we can get automatic gradients at any edge in the graph in respect to any other because gradient flow across basic mathematical operations can be defined according to simple rules. This is a huge advantage for numeric methods that require gradients, be it for simulation or for optimization such as training a neural network. Fig. 4.1 shows a code example in TensorFlow.

```
import tensorflow as tf

# Define input variable
x = tf.placeholder(dtype=tf.float32, name='X')

# Define constants a,b,c
a = tf.constant(0.3, name='a')
b = tf.constant(5., name='b')
c = tf.constant(1.7, name='c')

# Symbolic operation
y = a*tf.square(x) + b*x + c

# Open a session
sess = tf.Session()

# Run and print the symbolic variable 'y', feeding in the
# necessary x input having a value of '2'
print sess.run(y, feed_dict={x:2})

# Log the graph
writer = tf.train.SummaryWriter("/tmp/tbg", sess.graph_def)
```

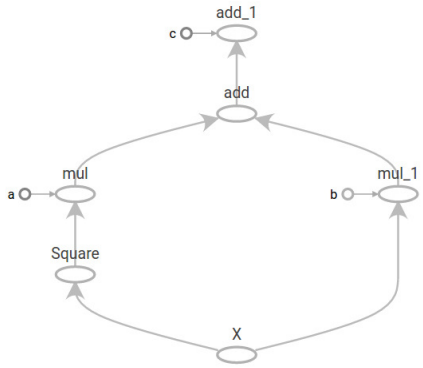


Figure 4.1: Simple code example in TensorFlow showing a computation of $y = ax^2 + bx + c$ along with the generated computational graph. Code example taken from [10]

4.2 TensorBoard

TensorBoard is a very convenient visualization tool natively provided by TensorFlow. During a TensorFlow session a user may choose to log various variables and statistics at each iteration of an optimization algorithm. This data is logged into a file which can be opened by TensorBoard in any browser on a localhost in an HTML format. Figure 4.2 shows a print screen of the TensorBoard [1].

4.3 Implementation

In this section we will take a quick look at the functions in TensorFlow used to implement the neural network pipeline.

4.3.1 VGG implementation

The VGG feature extractor network consists of 13 convolutional transformations with activations and 4 maxpool operations. The convolutions use 3×3 filters with stride 1 with 'SAME' padding [10] . This means that the layer is padded with zeros such that the resulting feature map dimension doesn't

[1] TensorBoard: Visualizing Learning [9]

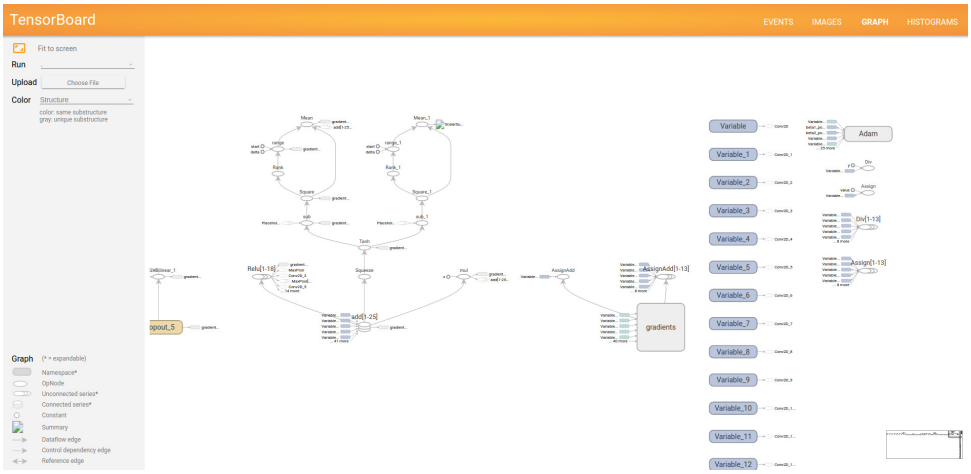


Figure 4.2: A screenshot of the TensorBoard opened in an internet browser showing part of the generated computation graph. On the top right corner are tabs to windows that enable us to see the histograms of our parameters and other useful information such as the training error and images.

shrink. The convolutional dot products are followed by a ReLu activation [33]. The weights of the VGG network are loaded from a file provided by the VGG.

4.3.2 Skip connections

As described in the architecture section 3.3.6, we can make a better prediction on the output by using information from various layers in the network; lower for structure and higher for semantics. We tap feature maps from layers 7,10 and 13 which are the last layers in the last 3 slices of the network. By a slice we mean the layers which do not have pooling operations between them and therefore have the same resolution. At each tap point we convolve the feature maps twice. We decided that the final prediction map will be of the same resolution as layer 10. This means that the results from layers 7 and 13 have to be downsampled and upsampled accordingly. The downsampling is done by maxpooling and the upsampling is done by unpooling. Unfortunately at the time of implementing the pipeline there is no official support for an unpooling layer in TensorFlow so we make due with an image resize function which uses bilinear interpolation to upsample a given feature map. The function is not intended to be used as an unpooling layer but it is fully differentiable so we can pass gradients through it. The resized feature maps are then concatenated using a concatenation operation and then put through a final 1×1 convolution and fed into a *tanh* 3.13 activation function. This produces a two-dimensional prediction map which corresponds spatially to the the input patch. Note: At inference, as during training only the centered subpatch is valid.

4.3.3 Backpropagation algorithm

We use backpropagation [46] to train our network. As mentioned in section 2.1.2, in each iteration of the algorithm we perform a forward pass on a training example and observe the effect of each neuron on the output error. For each neuron we calculate the gradient with respect to the output and then jointly update all neurons. The update rule for each neuron is shown in equation 4.1 where w_{ij} is the ij_{th} weight between two neurons i, j in two adjacent layers, α is the learning rate and E is the cost function

$$w_{ij} = w_{ij} - \alpha \cdot \frac{\partial E}{\partial w_{ij}} \tag{4.1}$$

It is unfortunately computationaly inefficient to calculate the gradient for each neuron separately. That is why the transformations between the layers are treated as dense matrix multiplications which can be efficiently computed and parallelized on both CPU and GPU.

4.3.4 Optimization

Optimizing (training) a deep neural network can be a tedious process due to the non-convexity of the problem and the diminishing gradient [27] as it gets propagated to the lower layers. Traditionally, the SGD algorithm [50] is used to optimize neural networks but for deeper networks it starts becoming essential to add momentum [54] to the optimization model. TensorFlow provides us with efficient ready optimizers that we can use to minimize a cost function. The one used in this work is an implementation of the ADAM [32] algorithm. ADAM is an optimizer that uses SGD with adaptive momentum. We can compare the basic update rule of SGD 4.1 with the update rules of ADAM 4.2,4.3,4.4 . We can see that the variable x is not updated directly from the noisy gradient vector d_x, but rather from the estimated lower-order moments m, v. The ϵ constant is there to avoid division by zero.

$$m = \beta_1 \cdot m + (1 - \beta_1) \cdot d_x \tag{4.2}$$

$$v = \beta_2 \cdot v + (1 - \beta_2) \cdot d_x^2 \tag{4.3}$$

$$x = x - \alpha \cdot \frac{m}{\sqrt{v} + \epsilon} \tag{4.4}$$

4.3.5 Out of core training

It was found by trial and error that the training batch size was critical for the optimization of the neural net. For a larger network an adequate batchsize needs to be used. A size of 60 was found to work decently. A batchsize that is too low results in erratic progress or even complete failure to converge. This is because for a deep network, a small batchsize gives very sharp, noisy gradients which constantly knock the network off the data manifold and make

the training difficult. This is especially true for a dataset such as the one described in this task where the objects of interest are very sparsely located.

Most of the experiments were done on a mid-tier desktop machine with a low-end GPU with 2GB memory. Usually when performing an optimization proceedure in TensorFlow we define an optimizer on a cost function and then we iteratively calculate the gradients and apply updates for the given batch. Due to memory limitations on the GPU it is not possible to buffer the activations and gradients for the required batch size onto the GPU and is therefore necessary to accumulate gradients on smaller batches (microbatches) or individual examples and then perform the update.

4.4 Parallelization

As was stated earlier in this chapter, upon compilation of the computation graph, TensorFlow spreads out the computation across multiple devices if available which allows it to take advantage of model parallelism [2]. For a regular desktop PC these devices are a CPU and a single or multiple GPU(s). TensorFlow allows the user to explicitly define variables and operations on specific devices using context tags such as 'with tf.device('/gpu:1'):' which means that we can spread the computation out as we desire. TensorFlow also released support for computation across a cluster of servers which is called 'Distributed TensorFlow'. [4]

4.5 Timing tests

Table 4.1 of training passes (using GPU) on the proposed object detection network using the VGG network as a feature extractor is shown in figure 3.14. The table shows the training passes per sample of size 256×256 pixels for the various microbatch 4.3.5 sizes for the out of core training.

	b = 8	b = 6	b = 4	b = 2	b = 1
Skip connections only	97 ms	100 ms	102 ms	115 ms	136 ms
Skip + last 3 layers VGG	125 ms	132 ms	135 ms	182 ms	226 ms

Table 4.1: Table showing forward + backward pass times for the various b microbatch sizes.

We can see from Tab 4.1 from that training into the VGG network requires significantly more time and resources than just training the skip connections because we have to propagate the gradient further. This also demonstrates that higher batch sizes require smaller compute times per sample. The forward pass only for the gpu per patch is about 85ms.

We also compare the forward and backward passes on this architecture on the CPU. The forward pass is about 1250 ms per sample and the full

[2]Model parallelism here means that parts of the computation graph are executed on the CPU and some on the GPU

pass is about 1750ms. This is approximately 14 times slower than on the GPU which makes it impractical to perform any experiments with this sort of architecture without having access to GPU acceleration.

Part III

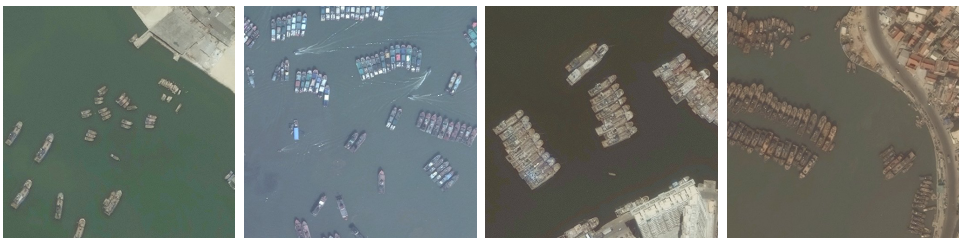
III

Chapter 5

Results & evaluation

5.1 Image sets

Throughout the course of this work we train and test our model on two different sets of images. The first set is compiled out of 35 images of size 1500×1500 pixels but contains only about 12 unique ports, provided by Digital Globe at a resolution of 0.5m. This set contains regions that are densly packed with ships, with up to 20 % of area covered in some images. The images exhibit similar color statistics and contain ships that have roughly the same profile and size, with most being about 70-100 pixels long. This imageset will be referred to as set **A** 5.1.



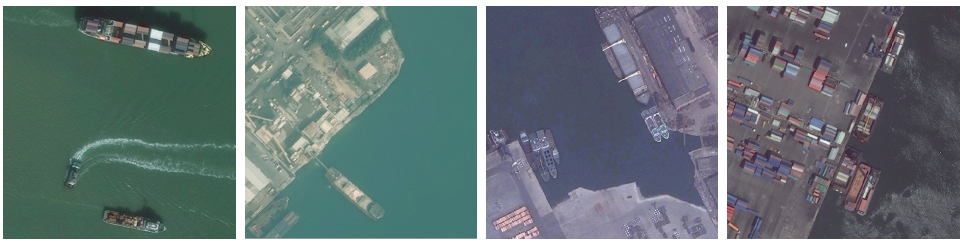
(a):

Figure 5.1: Examples from Image set A. All shown patches are of size 500×500

The second set consists of 25 images of size 2000×2000 pixels which contain 25 unique ports, and includes mostly images provided by Digital Globe at a resolution of 0.5m, with a few images from Planet Labs at a resolution of 3m. This set contains regions that are mostly sparsely packed with ships. These images exhibit very diverse color statistics and contain ships that have very different profiles and sizes, ranging all the way from tens of pixels long up to 500 pixels long. This imageset will be referred to as set **B** 5.3 and is considered as a more 'difficult' set than set **A**.

5.2 Evaluation metrics

To test the performance and generalization of our network we use image set **B**. The set is annotated in the same fashion as the training set. The images



(a):

Figure 5.3: Examples from Image set B. All shown patches are of size 500×500

are then tiled up in patches of 256×256 with strides of $(128, 128)$ (stride is half of the patch because we are only predicting the inner subpatch of each patch) and saved into a 4-dimensional tensor for quick access and evaluation.

5.2.1 Total mean square error

The mean square error (MSE) is the most basic error metric for our network and gives us a general idea on how well it is doing. The metric is used as the validation metric during training. The error is given by equation 5.1

$$E = \frac{1}{k} \sum_{N} (\sum_{m} \sum_{n} (P^{j}_{s_{m,n}} - G^{j}_{s_{m,n}})^2) \tag{5.1}$$

where k is the batchsize, m and n index the rows and columns of the patches respectively, N is the set of all elements in the batch, j indexes the j_{th} element in the batch, P_s and G_s are the prediction and ground truth subpatches. The 's' subscript denotes that it is indeed the subpatch from which we calculate the error and not the whole patch. Note that we do not threshold the classification in any way and therefore this metric takes into account the classification confidences of each pixel.

5.2.2 ROC curve

To evaluate a binary classifier on a dataset with disproportionate distribution of classes we cannot simply use an accuracy metric. This is because for a very sparse occurrence of class A versus class B, a classifier that always predicts B will be very accurate. To evaluate such a classifier we can use a ROC (Receiver operating characteristic) [23], which is a plot of the true positive rate (TPR) against false positive rate (FPR) for all classification thresholds in the range of the output of our classifier. The TPR and FPR in our case are calculated on a pixelwise basis and are shown in equation 5.2, where TP is the total amount of positively predicted (ship) pixels, P is the total amount of positive pixels, FP is the amount of background pixels that were predicted as positive and F is the total amount of background pixels. The output of the network is fed into a $tanh$ 3.13 activation function which means that for each pixel in the grid we get an output which ranges from -1 to 1. We can use the AUC (area under curve) to evaluate our classification. The area ranges

from 0.5 to 1, with an area of 0.5 meaning random classification and an area of 1 meaning a perfect classification. Using the ROC curve we can also get a visual idea of how the rates change with the various thresholds.

$$TPR = \frac{TP}{P}, \quad FPR = \frac{FP}{N} \tag{5.2}$$

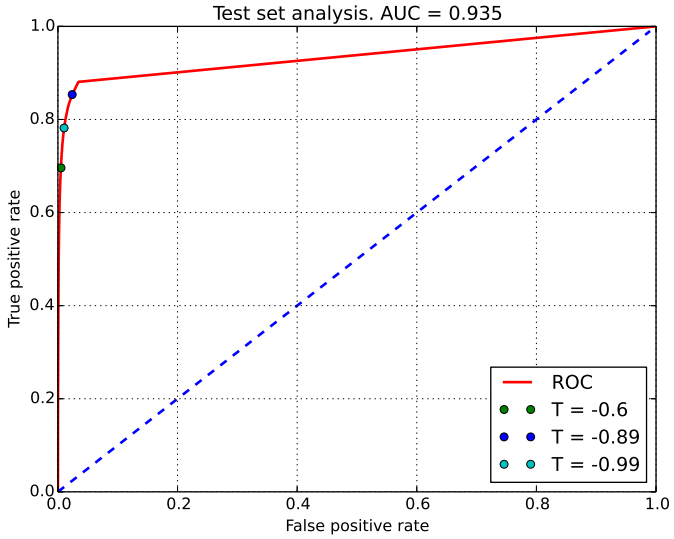


Figure 5.5: ROC evaluation on imageset B using the 3.14 architecture. Legend shows the position on the curve of three thresholds. The blue dashed line shows the points where the FPR and TPR are equal.

5.3 Choosing a threshold

To help us choose a threshold for our classifier we can make use recall versus precision plot5.3. The point where the precision and recall have the same value is called the break-even point. We choose the threshold that corresponds to the break-even point.

$$Precision = \frac{TP}{TP + FP}, \quad Recall = \frac{TP}{TP + FN} \tag{5.3}$$

where TP, FP, FN denote true positive, false positive and false negative respectively.

Fig. 5.8 shows a plot of the precision vs recall calculated for thresholds ranging from -1 to 1 in increments of 0.01. We can see that the break-even point is at about 0.69, giving us a TPR of 0.69 and FPR of 0.005 This corresponds to a classification threshold of -0.6, meaning that we classify every pixel that is predicted above -0.6 as ships. Fig. 5.6 shows classification results with various thresholds.

(a) : Original image **(b) :** Ground truth **(c) :** Raw predicted mask

(d) : $T = -0.99$ **(e) :** $T = -0.89$ **(f) :** $T = -0.6$.

Figure 5.6: Effect of different thresholds on classification. Threshold is denoted by T and ranges from -1 to 1. We can see that decreasing the threshold triggers pixels mostly around the ships.

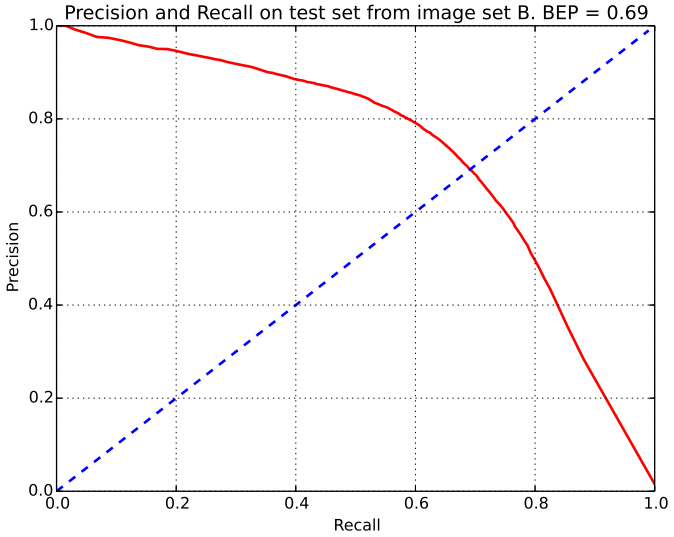


Figure 5.8: Precision-recall break-even point on test set of size 10 from imageset B

Chapter 6

Discussion

6.1 Network performance

We saw that given the task of distinguishing only one type of object from a background, we can expand on the notion of using a region proposal network [44] (weak classifier) into an object detector by modifying the architecture. Since we are predicting a mask for a given input patch we have to choose at which layer we will use the feature maps for the reconstruction. Using our understanding of hierarchical abstraction in neural networks, we can state that using feature maps that are closer to the input will give us more accurate structural information and higher level layers will give us more semantic information [40]. It was shown that tapping from layer 10 in the VGG feature extractor gives crude results both structurally and semantically. We saw that adding skip-connections [40] from the 7th and 13th layers significantly improves performance in both structure and semantics respectively.

During the development of the neural network architecture for this task it was noted that besides the actual choice of layers and connections, small details play a crucial role in the training and performance of the algorithm. Details such as the dropout, learning rate and batch size during training, which when inappropriately configured, can result in a network that fails to converge or that simply outputs a blank prediction grid. We also noted that the generation of the training set dictates the performance and training process of the network. If the training set is too severely distorted/augmented from the start then the network might fail to converge or take a very long time before the gradients start flowing as can be seen in the flat region in the graph of the validation error in TensorBoard 6.1.

One of the most important things that we noted during the development of the algorithm is that when using deep neural networks for more complicated tasks it is crucial to respect the structure of the network. We talked about an example of this in the initial attempt at the gridwise prediction where we attempt to project a prediction of an input subpatch into the whole output grid 3.3.5. This was unsuccessful because the output prediction grid corresponds to the entire input patch, not just the subpatch.

Overall, the performance of the network exceeds expectations considering that the training dataset consists of 25 annotated images. We have observed

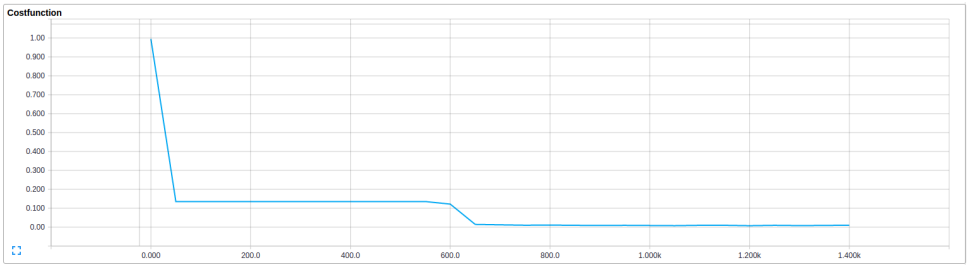


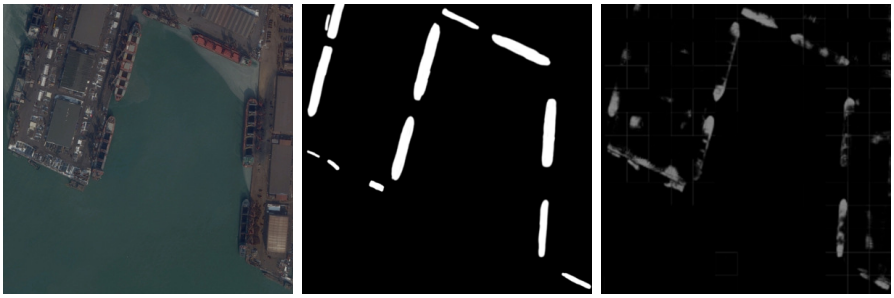
Figure 6.1: TensorBoard screenshot showing training of the network on a harshly augmented dataset. We can see a flat period of cost function before gradients start to flow. MSE on the y axis and iterations on the x axis.

that it is capable of generalizing to ships and environments which it has not seen before. Fig. 6.2 shows the performance at various resolutions from highest (0.5m) to lowest (3m).

We can see in 6.2 that the issue with the large ships is that they do not fit inside our 256×256 context window and so are detected mostly on the front and back where the characteristic of the bow and stern are more prominent. The medium sized ships are reliably detected because they are most prevalent in the training set. The algorithm also attempts to generalize to much lower resolution images from PlanetLabs even though the training set includes no such images 6.2. Fig. 6.4 shows the performance of the algorithm on crops of three test ports from image set **A** 5.1 with heavy cloud coverage. A break-even point of 0.64 is calculated on this set with a FPR of 0.008 but a TPR of only 0.62. The low TPR is expected because many of the ships are heavily obstructed by the clouds.

6.2 Future work

As the method in this work was partially inspired by Faster RCNN [44] one could note that it would also be possible to take it one step further and predict bounding boxes around each object, hence enabling counting objects. This could work by regressing scores for a set of anchor boxes for each pixel in the predicted grid, alongside the semantic prediction of whether that pixel is a region of interest or not. The semantic region proposal would be trained separately and then the box regression would be trained on top. A classification metric or intersection over union could be used as the cost function for the predicted anchor boxes. The tricky part would be to the post-process and merge the predicted boxes into the final result.



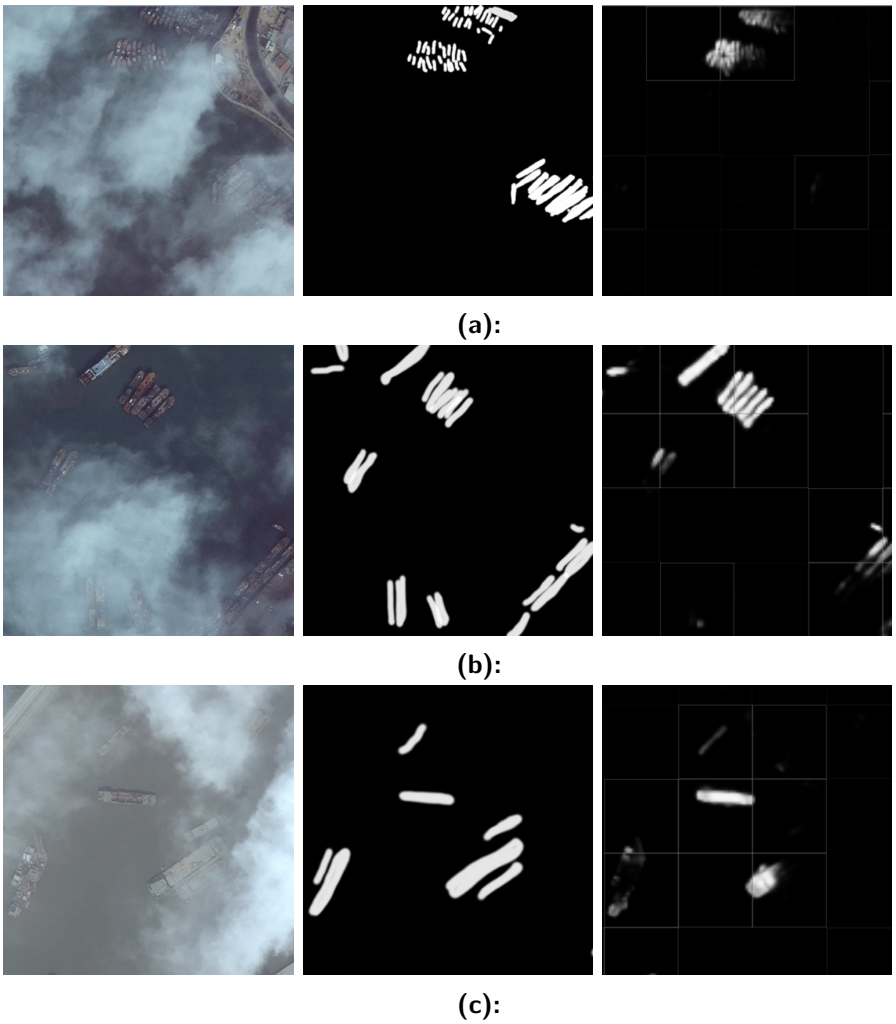
(a) : Image size: 2k x 2k pixels, provider: Digital Globe, resolution: 0.5m. Big ships are up to 500 pixels long which is too big for the 255×255 context window. We can see that the network detects the characteristic bow and stern of the ships but is unable to discern the bay most of the times.

(b) : Image size: 1k x 1k pixels, resolution: 0.5m, provider: Digital Globe. The largest ships are about 170 pixels long. We can see that the network does very well for ships this size and resolution.

(c) : Image size: 500 x 500 pixels, resolution: 3-5m, provider: Planet Labs. The algorithm attempts to generalize to images at this resolution even though such images are not in the training set.

(d) : Image size: 1000 x 1000 pixels, resolution: 0.5m, provider: Digital Globe. Result on a grayscale image. No grayscale images appear in the training or test sets.

Figure 6.2: Comparison of the variety of test images from image set **B**. From left to right: Original image, Ground truth, raw prediction by algorithm.



(a):

(b):

(c):

Figure 6.4: Performance on test images from image set **A** with high cloud coverage. From left to right: Original image, Ground truth, raw prediction by algorithm. Precision-recall break-even point at 0.64. TPR: 0.62 , FPR: 0.008. No cloud covered images appear in the training set.

Chapter 7

Conclusion

In this work we demonstrated a successful attempt at designing and implementing a deep neural network pipeline in TensorFlow for detection of objects in high resolution satellite images. The initial task of producing a bounding box around each object of interest was scrapped due to difficult conditions such as very tightly packed ship clusters and ambiguous regions where the boundary between the ships is blurred. We instead showed a working method for pixelwise labeling of the scene. The method is able to distinguish between the object of interest and the rest of the background and can be used to track the maritime traffic in a chosen port or region. The network, trained on a small set of only 25 annotated images, was tested on a dataset of 10 ports from various regions with ship sizes ranging from 30 pixels to 500. The result is a precision-recall break-even point of 0.69, with an FPR of 0.69 and FPR of 0.005, which is a satisfactory result given the small size of the dataset. We conjecture that this result can be vastly improved simply by training the network on a larger dataset and increasing the context window to detect larger ships.

Due to the lack of annotated datasets and metadata to determine resolution we are unable to thoroughly quantify the generalization of the network to specific resolutions. A visual observation 6.2 shows that the algorithm generalizes well to test images of a similar resolution (0.5-1m) as the training set but does not perform so well on images of much lower resolution (3-5m). We conjecture that this can easily be improved by including low resolution images in the training set. We also observe that the algorithm can deal with defects in the image and obstructions such as clouds without additional preprocessing as can be seen in Fig. 6.4. By 'deal' we mean that it has lower performance but does not falsely trigger on the obstructions. It has to be noted, however, that corrupt and bad quality images are scrapped at the time of inference on the basis of provided metadata. At the moment the input is limited to RGB and grayscale images and more research will have to be done to adapt it to additional channels.

Appendices

Appendix A

Bibliography

[1] Convolutional neural networks (lenet). http://deeplearning.net/tutorial/lenet.html. Accessed: 2016-05-09.

[2] Cubesat. http://static1.squarespace.com/static/5418c831e4b0fa4ecac1bacd/t/56e9b62337013b6c063a655a/1458157095454/cds_rev13_final2.pdf. Accessed: 2016-05-09.

[3] Digital globe. https://www.digitalglobe.com/. Accessed: 2016-05-09.

[4] Distributed tensorflow. https://www.tensorflow.org/versions/r0.8/how_tos/distributed/index.html. Accessed: 2016-05-09.

[5] Geoeye-1 datasheet. http://global.digitalglobe.com/sites/default/files/DG_GeoEye1.pdf. Accessed: 2016-05-09.

[6] Gimp (gnu image manipulation program). http://www.gimp.org. Accessed: 2016-05-09.

[7] Neural network cross entropy error. https://visualstudiomagazine.com/articles/2014/04/01/neural-network-cross-entropy-error.aspx. Accessed: 2016-05-09.

[8] Planet labs. https://www.planet.com/. Accessed: 2016-05-09.

[9] Tensorboard. https://www.tensorflow.org/versions/r0.8/how_tos/summaries_and_tensorboard/index.html. Accessed: 2016-05-12.

[10] Tensorflow api. https://www.tensorflow.org/versions/r0.8/api_docs/index.html. Accessed: 2016-05-09.

[11] U.s. department of commerce relaxes resolution restrictions digitalglobe extends lead in image quality. http://web.archive.org/web. Accessed: 2016-05-09.

[12] Worldview-2 datasheet. http://global.digitalglobe.com/sites/default/files/DG_WorldView2_DS_PROD.pdf. Accessed: 2016-05-09.

[13] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[14] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[15] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[16] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[17] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[18] J. Dai, K. He, and J. Sun. Instance-aware semantic segmentation via multi-task network cascades. *CoRR*, abs/1512.04412, 2015.

[19] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, June 2005.

[20] Y. N. Dauphin, H. de Vries, J. Chung, and Y. Bengio. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. *CoRR*, abs/1502.04390, 2015.

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

[22] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929, Aug 2013.

[23] T. Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006.

[24] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[25] R. B. Girshick. Fast r-cnn. *CoRR*, abs/1504.08083, 2015.

[26] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[27] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, pages 249–256, May 2010.

[28] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[29] C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. 2003.

[30] A. C. Ian Goodfellow, Yoshua Bengio. Deep learning. Book in preparation for MIT Press, 2016.

[31] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[32] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[35] Y. LeCun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. Handwritten digit recognition: Applications of neural net chips and automatic learning. In F. Fogelman, J. Herault, and Y. Burnod, editors, *Neurocomputing, Algorithms, Architectures and Applications*, Les Arcs, France, 1989. Springer.

[36] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, chapter Microsoft COCO: Common Objects in Context, pages 740–755. Springer International Publishing, Cham, 2014.

[37] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[38] V. Mnih and G. Hinton. Learning to label aerial images from noisy data. In *Proceedings of the 29th Annual International Conference on Machine Learning (ICML 2012)*, June 2012.

[39] A. Neubeck and L. J. V. Gool. Efficient non-maximum suppression. In *ICPR*, 2006.

[40] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, 2015.

[41] L. Y. Pratt. Discriminability-based transfer between neural networks. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 204–211. Morgan-Kaufmann, 1993.

[42] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, abs/1403.6382, 2014.

[43] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[44] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[45] F. Rosenblatt. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Washington: Spartan Books (1962).

[46] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[47] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[48] C. Schmid. *CVPR 2005 : proceedings : 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, Los Alamitos, Calif, 2005.

[49] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013.

[50] O. Shamir and T. Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. *CoRR*, abs/1212.1824, 2012.

[51] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[52] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[53] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[54] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1139–1147, 2013.

[55] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. *CoRR*, abs/1312.4659, 2013.

[56] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 2013.

[57] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511–I–518 vol.1, 2001.

[58] P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154.

[59] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 27 (NIPS '14)*, pages 3320–3328. Curran Associates, Inc., 2014.

[60] C. L. Zitnick and P. Dollár. *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, chapter Edge Boxes: Locating Object Proposals from Edges, pages 391–405. Springer International Publishing, Cham, 2014.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Teymur A z a y e v

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Bachelor Project: Object Detection in High Resolution Satellite Images

Guidelines:

The aim is to design, implement, and experimentally evaluate a deep neural network pipeline for detection and classification of objects in high resolution satellite images. Beside the deep network, resulting pipeline should include image pre-processing algorithms to cope with input images of varying quality, resolution, and number of channels. Recommended framework for implementation is the TensorFlow https://www.tensorflow.org/ for Python. Datasets, including the ground truth, will be provided in close cooperation with Spaceknow, Inc.

Bibliography/Sources:

[1] Mnih, Volodymyr, and Geoffrey E. Hinton. "Learning to label aerial images from noisy data." Proceedings of the 29th International Conference on Machine Learning (ICML-12). 2012. http://www.cs.toronto.edu/~fritz/absps/noisy_maps.pdf

[2] Abadi, Martın, et al. "TensorFlow: Large-scale machine learning on heterogeneous systems, 2015." Software available from tensorflow. org. http://download.tensorflow.org/paper/whitepaper2015.pdf

[3] Ian Goodfellow Bengio, Yoshua, and Aaron Courville. "Deep learning." An MIT Press book in preparation. Draft chapters available at http://www.deeplearningbook.org/

Bachelor Project Supervisor: Ing. Michal Reinštein, Ph.D.

Valid until: the end of the summer semester of academic year 2016/2017

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 4, 2016