Deep learning approaches for maritime ship detection from inshore and offshore imagery

Jesper Winsten, 2004757
Master's thesis in Computer Science
Supervisors: Prof. Johan Lilius, Dr. Bogdan Iancu
Double Degree:
-Faculty of Science and Engineering
Åbo Akademi University
-Department of Computer Science
Reykjavik University
2022

Abstract

Maritime surveillance and situational awareness are of topical interest but require massive amounts of human labour if done manually in a traditional setting. The human factor can make this process also highly expensive and unproductive. Computer vision techniques, through maritime object detection and tracking, can transform maritime surveillance and improve maritime navigation. Classic object detection algorithms trained on general-purpose datasets do not yield satisfactory results for maritime vessel detection, one reason being that ships constitute only a small fraction of objects in these datasets. In this context, availability of domain-specific datasets is crucial, hence a few maritime detection datasets have been published, for example, ABOships, Seaships, Singapore Maritime Dataset, MCShips and MARVEL. Various object detection algorithms (one- and two-stage detectors) have been employed for ship detection with varying performance. More recently, a new class of detectors, key-point detectors, emerged, yielding promising results on some of the maritime detection datasets mentioned above. This thesis investigates the use of transfer learning techniques on a CenterNet implementation (with various feature extractors), an evaluation of their performance is done on a locally collected maritime vessel dataset, ABOships. Results are evaluated under several augmentation techniques, using Average Precision (AP) and Intersection over Union (IoU) metrics.

Keywords: Maritime surveillance, Computer vision, Maritime object detection, Transfer learning, Deep Learning, Maritime detection datasets, CenterNet

Preface

I would like to thank my supervisors Dr. Bogdan Iancu and Prof. Johan Lilius for helping me with my thesis, as well as for their insightful advice. Thank you, Professor Johan Lilius, for giving me the opportunity to write my thesis and help with research for the Sea4Value Smarter project. I would also like to thank Valentin Soloviev for his help on various coding issues. Lastly, I would like to thank all my co-workers at the Department of Information Technology at Åbo Akademi University for keeping me motivated to work on my thesis.

Jesper Winsten
May 13, 2022

Table of Contents

1 Introduction

Detection of maritime objects gathered from inshore and offshore imagery is of high importance in the context of coastal surveillance and situational awareness. Traditionally, an extensive number of hours has been devoted to manually both classify and surveil ships in maritime environments, for example in border control or dock traffic monitoring. So far, generic object detection algorithms that have been trained on general-purpose datasets have not been able to yield adequate performance in a marine environment, a big reason for this is the fact that purely maritime vessel datasets account for only a small percentage of all object detection datasets. There is a crucial need for autonomous systems in maritime environments to be able to efficiently identify vessels through maritime object detection and for domain-specific datasets to be readily available. Notable datasets worth mentioning within the domain of maritime object detection are: ABOships [1], Seaships [2], Singapore Maritime Dataset [3], MCShips [4], and MARVEL [5], some of which will be discussed in more detail in this thesis.

This thesis investigates the usage of a key-point detector for identifying maritime vessels and objects, as well as how the performance of transfer learning from e general-purpose dataset like COCO [6] is in comparison with training from scratch on a domain-specific dataset, like ABOships.

There is a scarce number of datasets specifically tailored towards maritime object detection while in comparison there are a lot of general-purpose datasets which contain a small subsample of maritime objects. In this thesis I will discuss one-stage, two-stage and key-point detectors, and subsequently focus on transfer learning on a center-based object detector - CenterNet (Objects as points) [7].

The aim of this thesis is to explore the viability of a center-based key-point detector called CenterNet for maritime object detection. CenterNet comes with the added benefit of a certain modularity in choosing its backbone architecture (some of which have been slightly modified from their original). The backbones used in this thesis are as follows: DLA-34, Hourglass, ResNet101 and ResNet18. A set of training runs were conducted, such that each backbone in the network were at least trained from scratch and pre-trained with the COCO dataset. Each model was evaluated by means of average precision over a set of intersection over union thresholds. In order to understand the context of key-point detectors on domain-specific datasets, the thesis will cover and explain one- and two-stage detectors, transfer learning as well as the domain of maritime vessel detection.

The rest of the thesis is structured as follows. Chapter 2 present preliminary background information about generic object detection as well as a more detailed explanation of deep convolutional neural networks (DCNNs) and their main detector types. Chapter 3 discussed transfer learning, how it can be used in object detection, different approaches for deep learning as well as giving an overview of its formal definitions and scenarios. Chapter 4 discusses a multitude of maritime vessel datasets, each varying in size and scope. The chapter also introduces the dataset used to evaluate the performance of CenterNet. Chapter 5 explains the overall structure and different backbones of CenterNet in greater detail. An exploratory analysis of the ABOships dataset is conducted as well as an explanation as to how and why the dataset has been modified. The chapter also presents the evaluation methods used to measure the performance of each model trained on the dataset. Finally, the chapter ends with the results of each model being from trained from scratch or using transfer learning. Chapter 6 presents a conclusion and discussion about future work.

2 Deep learning for object detection

Before discussing the intricacies of Deep Learning (DL) or Machine Learning (ML), I briefly introduce the super-category, Artificial Intelligence (AI). AI is inspired by the complex structure of the human brain and aims to imitate intelligent human behaviour to some extent. Machine learning (ML) is a sub-category of AI, extending the capabilities of a system to learn and improve automatically, based on experience. Most common ML methods include unsupervised learning, supervised learning, semi-supervised learning and reinforcement learning. Moreover, deep learning (DL) is itself a sub-category of ML. A Venn diagram between the relationships of AI, ML and DL can be seen in Figure 1.

Deep artificial neural networks (DANNs) distinguish themselves from Artificial Neural Networks (ANN) primarily through the scale of the network, the tasks they aim to solve and how they are trained (scale in this context means the depth of the network, i.e. number of layers). Task differences between the two usually narrows down to the complexity of the task. Training differences are the following: ANNs are trained through backpropagation, meaning that the error (often called loss) is calculated backwards through the network to effectively compute the gradient. This, in turn, means that the ANNs are able to learn from previous errors and improve upon them. A more in-depth review of DL, more specifically Convolutional Neural Networks (CNNs) and its deeper variant will be discussed in a later sub-chapter. ML networks, in contrast, rely more on the tuning of parameters after each training run (epoch) [8, 9]. All of these differences will be explained in more detail throughout the thesis.

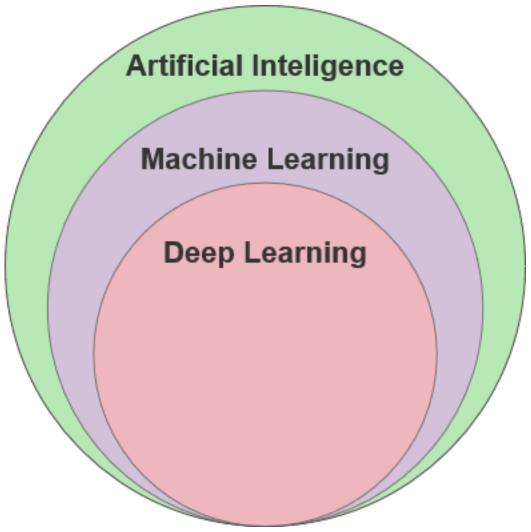


Figure 1: Venn-diagram of the relationships between Artificial Intelligence, Machine Learning and Deep Learning.

2.1 Generic object detection

Generic object detection is defined as identifying a set of objects that are of pre-established categories or classes in an image and then returning each individual object's location and semantic information. An example of a set of objects could be a set of cars or motorcycles in an image, each with their own intraclass variations. The locations of objects in an image is most commonly depicted as bounding boxes (a vector of four values, usually xmin, xmax, ymin, ymax) or as pixelwise segmentation masks [9], both depicted in Figure 2. Object detection can furthermore be categorized into four distinct image recognition problems: object classification, generic object detection, semantic segmentation and object instance segmentation [8, 9].

Object classification, as shown in Figure 2 (a), is classifying the various instances of objects in an image; in this example, there are two class types, ferry and motorboat (notice also that there is no effort in separating objects from one another). Semantic segmentation, Figure 2 (c), is a pixelwise segmentation of each object in the image, again, without distinguishing between objects. Object detection, Figure 2 (b), is the process in which an object and its object class label is identified in an image, such that a bounding box and the object label is superimposed over each object in an image. Instance segmentation, Figure 2 (d), depicts a pixelwise segmentation of each object with the added benefit of distinguishing between object instances (in contrast to Semantic Segmentation that does not separate each object).

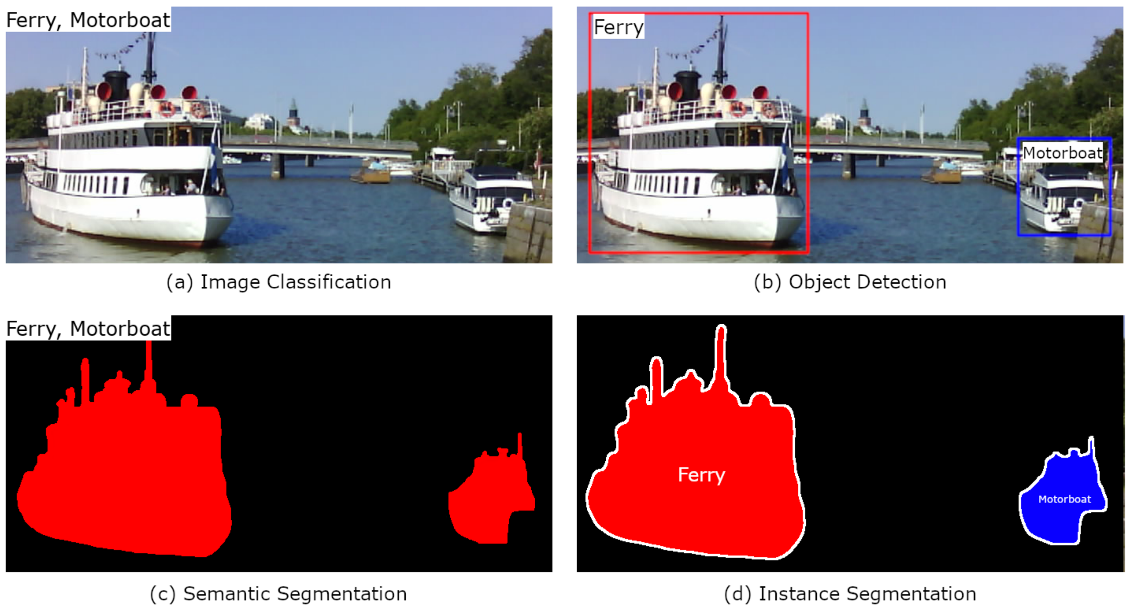


Figure 2: The different image recognition problems exemplified on the ABOships dataset [1]

However, before the onset of mainstream deep learning methods, object detection was performed by first generating proposals for regions of interest (RoI), then extracting feature vectors and finally classifying each RoI. The most common method for proposal generation was the use of *sliding windows*. The sliding window algorithm, as the name implies, slides a grid of a predefined size over an input image with a predefined stride which is done until the whole image has been worked over. The process is then repeated with a different size and stride to cover more regions. For each step that the sliding window takes, a feature vector extraction is obtained, meaning that semantic information of the particular regions are gathered [10, 8]. To identify what objects (class labels) are in the image, region classification was generally performed with support vector machines (SVMs) [8].

Often the focus of object detection was to manually create high quality feature descriptors, which was slow but effective at the time. In spite of the fact that these detectors generated impressive results for their time, they came with certain limitations, namely the massive number of generated proposals and the fact that a large sub-set of them were unusable as they returned false positives at the classification step. The sliding window algorithm was in itself inefficient, as the search space grows according to the amount of pixels in an image, $10^4 - 10^5$ windows per image growing to as much as $10^6 - 10^7$ windows per image, if multiple scales and aspect ratios are used [9]. An optimal global solution was not feasibly attainable, as each step of the process was tuned and optimized separately. Another reason for the inefficiency of precursor methods of DL are the fact that feature descriptors were manually created by humans, resulting in difficulties to capture quality semantic information in complex images [8].

The most common ANN for object detection is CNNs, which were first popularized by Krizhevsky et al. in 2012 with its impressive results in the ImageNet Large Scale Visual Recognition Competition (ILSVRC) competition [11]. A CNN is comprised of three distinct layers: convolutional layers (or just a convolution), pooling layers and, lastly, fully connected layers. These layers together can be considered as building blocks when it comes to bigger networks, such as, combining multiple such blocks to form a DCNN [12]. A high-level overview of a CNN can be seen in Figure 3.

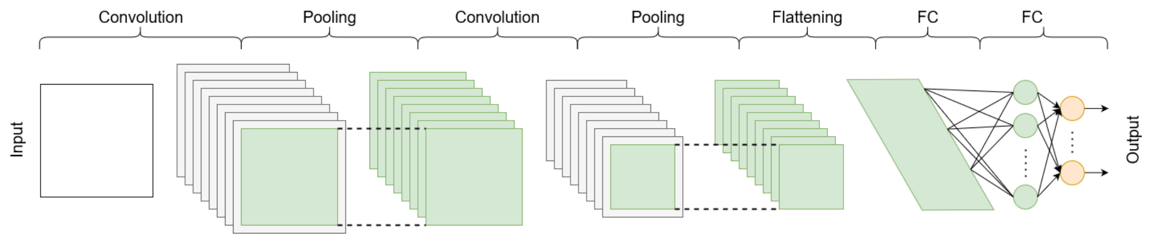


Figure 3: A high-level overview of a convolutional neural network.

The convolutional layers of CNNs incorporate two different operations in tandem for feature extraction, a convolutional operation (linear) followed by an activation function (non-linear) such as sigmoid, a hyperbolic tangent (tanh) or the most commonly used activation function, a rectified linear unit (ReLu) each of which can be viewed in Figure 4.

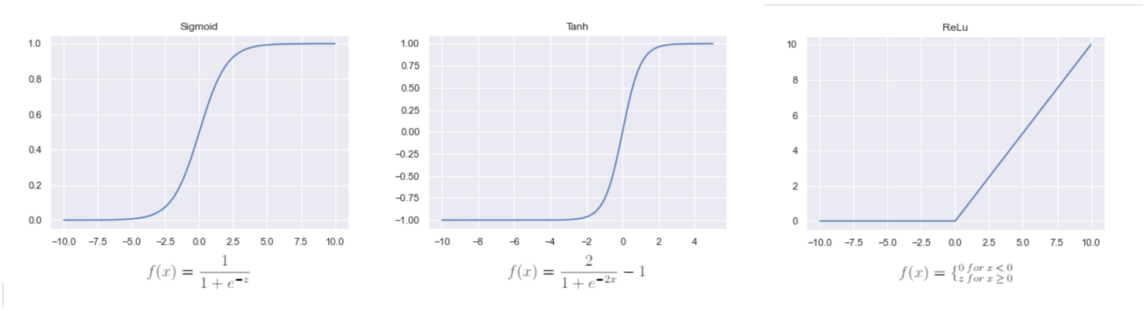


Figure 4: Three different activation functions: Sigmoid, Tanh and ReLu.

A convolutional operation sequentially uses a kernel (a matrix) on a tensor. In this context, a tensor is an image with any number of channels (or a multi-linear map). To obtain a feature map, an element-wise product is calculated for each element of the kernel and tensor and then summed to create resulting feature maps for each position of the output. The aforementioned process is performed multiple times after each other for as many times as the output requires. Two different values or hyperparameters are used to change the behaviour of the convolutional operation, the size of the kernel (usually an odd size like 3x3, 5x5, etc.) and the number of channels used (which controls the depth of the feature map). The distance that a kernel travels from one position to another is known as a stride, which also is a hyperparameter that can be tuned, a stride of one (1) is a common default. However, without the use of padding, the network is unable to keep its in-plane dimension and each feature map in succession would shrink and lose information. To address this problem, zero-padding is used, i.e, new rows to the outermost columns and rows are added and filled with zeroes in the tensor.

The biggest advantage of convolutions is the fact that kernels are shared across the image space. Figure 5 demonstrates a convolutional operation with a zero-padding of one (1) and a stride of one (1). Each convolution has an activation function, in the case of modern CNNs, a non-linear activation unit, i.e. a rectified linear unit (ReLu) which is shown in Figure 4. A ReLu computes the positive part of an argument, so for example, if the argument is a negative number, a zero (0) is returned and any positive number if the argument is a positive number, i.e. $f(x) = max(0, x)$ [12].

To decrease the ensuing learnable parameters and save on resources, pooling

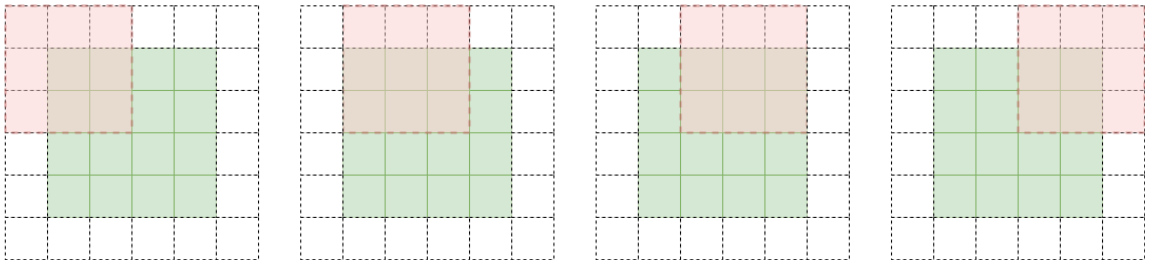


Figure 5: A convolutional operation with a zero-padding of one and a stride of one.

layers are used. Other than decreasing the number of parameters that the network needs to learn, pooling also introduces an effective way of downsampling to alleviate small shifts and distortions in the feature maps. The two most common pooling methods are: max pooling and average pooling. There are, however, some more specialized methods such as fractional max pooling which will not be discussed in this thesis. A pooling operation traverses each feature map sequentially extracting patches, most commonly at a filter size (the patch) of 2x2 with a stride of two (2), meaning then that the pooling operation effectively downsamples the feature map by a factor of two (2). Max pooling extracts the maximum value of each kernel, while average pooling extracts the average of each element in a kernel; average pooling is, however, the lesser used pooling operation of the two [12]. An example of a max-pooling operation with a kernel size of three (3) and a stride of two (2) is illustrated in Figure 6.

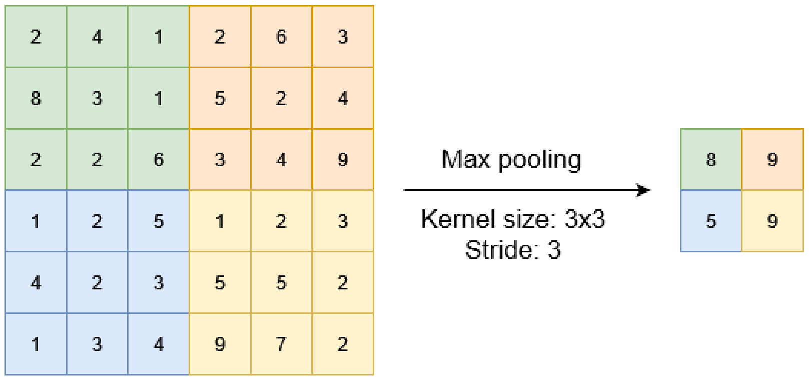


Figure 6: A max-pooling operation with a kernel size of 3x3 and a stride of three (3).

Fully connected layers (FC) of a CNN are used as a means to flatten the feature maps into one-dimensional arrays, i.e. vectors. The reason they are called fully connected layers is that each input of these layers is connected to each and every output, more often than not each output node of the last fully connected layer represents a class to be detected by the network. As mentioned earlier, each of these fully connected layers always ends with an activation function (non-linear in this case), typically a ReLu. The last FC layer of a CNN always ends with an activation function. The type of activation function that is used depends on the task that the network is required to solve, for example, Sigmoid is used for binary classification while Softmax is used for multiclass classification. Softmax takes the input vector and normalizes it according to class probabilities, ranging from 0-1 where all values summarize to 1 [12].

2.2 Main detector types

The two main types of DCNNs detectors are one-stage detectors and two-stage detectors [8] with a third becoming increasingly popular: key-point detectors [7]. One-stage detectors combine proposal generation and feature extraction through one pass of the neural network, meaning that they attempt to immediately predict the bounding boxes for each RoI: either as background or as a known object class. Two-stage detectors use two distinct stages for predictions: first generating a set of sparse proposals and then extracting feature vectors from them. Each of the detector types have a different set of use cases, for example, one-stage detectors are in general faster than two-stage detectors, making them excellent candidates for real-time tasks, while two-stage detectors are more accurate but less efficient (slower training/inference) [8].

2.2.1 One-stage detectors

OverFeat [13] is one of the first one-stage detectors to use convolutions to train a network for multiple different tasks, such as, classification, localization and object detection. Since OverFeat is one of the first networks to utilize CNN for feature extraction it shares many architectural similarities with the CNN popularized by Krizhevsky et al., with some added improvements to some areas. For classification each image that is fed through the network has one label assigned that correlates with the main object in the image. The network is allowed to make five proposals (guesses) per image. Localization works in much the same way, meaning that it also has five proposals per image but also tries to predict a bounding box for each proposal. To be considered a true positive the bounding box must have an Intersection over Union (IoU) value of at least 50% over the ground truth as well as be labeled correctly to its respective object class. For detection there is no limitation for the number of objects that can be detected in each image as well as the need to predict the background if no other object is present in the image. OverFeat also uses *negative training* which selects a couple of negative examples based on their offending negative error or just chosen at random. *Negative training* is used to prevent or reduce the chance of a mismatch of negative examples and training times, as well as reducing the chance of overfitting on smaller sets [13].

You Only Look Once (YOLO) [14] is unique in the sense that it handles object detection as a single regression problem, meaning that YOLO can detect objects and bounding boxes from a single sweep over an image. Consequently, the network can detect objects very fast (real-time). YOLO splits each image into an SxS grid, where each grid cell is only able to predict a single object in an image and where the grid cell that is in the middle of an object becomes responsible for the aforementioned object. Each grid cell is capable of predicting a set number of bounding boxes, each bounding box contains five different predictions: x, y, width, height and a confidence score. The x and y values represent the center of the bounding box while the width and height are calculated relative to the entire image. The confidence score is evaluated for each bounding box that the network can predict, the confidence score is the probability of an image containing a certain bounding box and the object within it (from 0 1). The confidence value is calculated using IoU and Non-maximum suppression (NMS). IoU calculates the overlap between the prediction and the ground truth label while NMS takes each similar IoU value over an object and filters out the best performing detection against a given threshold. Each individual grid also calculates a class probability of the cells containing an object. In addition to the class probabilities and bounding boxes (with their confidence values) YOLO keeps each prediction with a confidence score over or equal to a predetermined value for each object in an image and discards the rest to form the final detections. The drawback to YOLO is the fact that it identifies at most one object class per grid cell, meaning that the network has a hard time finding smaller objects in crowded images. YOLO is also highly generalized to the data it trains on, which leads it to generate bad predictions if new images are in a different aspect ratio [14].

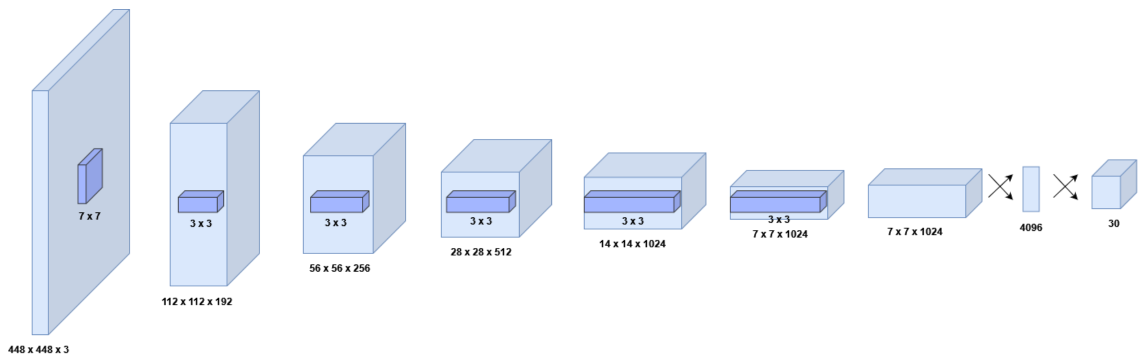


Figure 7: Overall architecture of YOLO.

YOLOv2 and YOLO9000 [15] are both distinct iterations of the original YOLO detector, each with their own unique set of features and use cases. YOLOv2 has many incremental improvements over YOLO all of which add up to both much faster detection speed and higher accuracy (average precision). Instead of using dropout YOLOv2 uses batch normalization on all of its convolutional layers, which helps with convergence and in turn to regularize the network. The starting resolution of the classifier is also increased, from 224x224 to 448x448, this makes it easier for the filters to adjust to images of higher resolution. The FC layers used in YOLO are replaced by anchor boxes while also removing the class prediction mechanism in favour of predicting class objects with the aforementioned anchor boxes. To find each anchor box, YOLOv2 uses IoU based k-means clustering, the author states that a cluster size of five (5) gives a good trade-off between recall and complexity. To further combat model instability YOLOv2 implements direct location predictions, i.e., predicting locations according to the feature map grid cell. The last incremental improvement for YOLOv2 involves multi-scale training, meaning that for every tenth epoch of a training run, YOLOv2 chooses a new image dimension at random, with a minimum resolution of 320x320 and a maximum of 608x608. This makes the network predictions robust over a larger variety of resolutions which also means that YOLOv2 can train on distinct resolutions. YOLO900 further built upon YOLOv2 with the added benefit of being able to detect over 9000 object classes by combining classification and detection optimizations. The author uses a WordTree to combine two different datasets (ImageNet and COCO), which the network is then trained on in the aforementioned way [15]. There are newer versions of YOLO, i.e., incremental revisions, version three (3) and version four (4), whereas that at the time of writing this thesis YOLO has evolved to a fifth version which is still under development and can be found at $https://github.com/ultralytics/yolov5$.

Single shot Multibox Detector (SSD) [16], much like YOLO, is built with the intent of detecting objects in real-time. SSD is based on the model architecture of VGG16 [17], but the author notes that any other backbone network should give good results. SSD uses small convolutional filters to compute the location and object class in images, i.e., feature extraction with the combination of a set of default bounding boxes. For feature extraction, SSD uses multi-scale feature maps from multiple layers instead of just one, where each consecutive layer reduces the spatial dimension and in turn reduces the resolution of each feature map. This entails that larger objects get reduced to a smaller resolution to be more easily detectable. Every feature map has a set of default bounding boxes applied to it, each of a different shape to effectively compute each distinct output shape that could be used in the final prediction. The default bounding boxes are manually chosen with a scale value and an aspect ratio for each feature map, both of which increase in a linear fashion as it goes through each layer and finally forming the width and height of the default bounding boxes. SSD like most other detector uses a set IoU thresholds to determine if a detection is positive or negative [16].

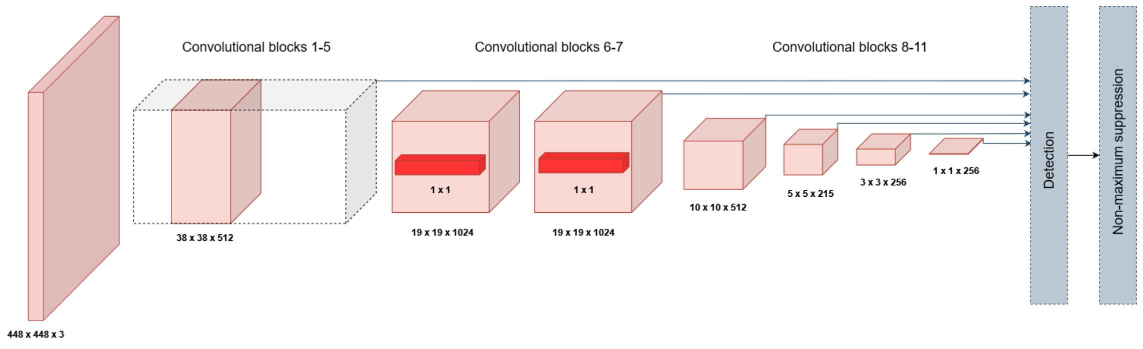


Figure 8: Overall architecture of SSD.

RetinaNet and Focal Loss [18] for object detection. The creator of RetinaNet and the novel loss function Focal Loss Lin et al. states that the main thing that prevents one-stage networks of gaining equal or better performance than two-stage networks is the extreme foreground-background class imbalance for one-stage networks. To combat this discrepancy a Focal Loss added to the standard cross entropy is implemented, this in essence creates a narrower focus on hard negative examples. Focal Loss is the normal loss function reformulated to focus more heavily on harder negative examples instead of easy negative examples, where which a modulating factor is attached to the cross entropy with an adjustable focus parameter: $-\alpha_t(1-p_t)^\gamma log(p_t)$ where gamma controls the curve of the loss while alpha helps to balance between negative and positive examples. However to reduce the loss of easily classified examples Focal Loss also introduces a multiplicative factor to the Cross Entropy loss, i.e., $(1-p_t)^\gamma$. A more detailed explanation can be found in [18]. RetinaNet uses a network based on a Feature Pyramid Network (FPN) [19] as backbone with a Residual Neural Network (ResNet) [20] for deep feature extraction. FPN takes in an input image and creates a multi-scale feature pyramid by feeding the image through a set of top-down and side-way connection from ResNet where each iteration through the connections brings a different scale for an object to be detected through, this can be seen in 9 [18]. FPN in essence can be seen as an improvement for multi-scale predictions on Fully Connected Networks (FCN) which will be discussed in the next sub-chapter. ResNet will also be further discussed more in detail in Chapter 5.1.1.

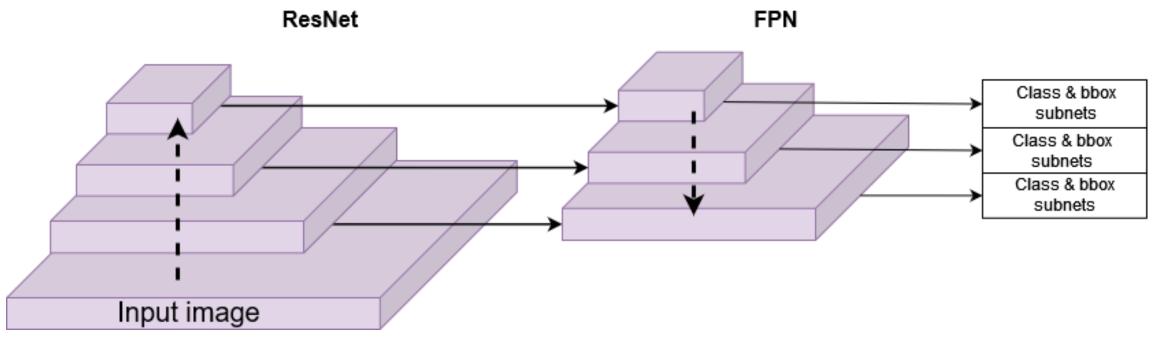


Figure 9: Overall architecture of RetinaNet.

2.2.2 Two-stage detectors

Region-based Convolutional Neural networks (R-CNN) [21] for object detection follows three consecutive steps to identify class objects in an image. The first step is the generation of candidate region-proposals, which are selected through *selective search*. Selective search works by finding an initial sub-set of regions with image segmentation, iteratively grouping similar regions together and combining the most similar regions to form larger regions. This process is repeated until the whole image becomes a single region [22]. The selective search algorithm results in 2000 candidate regions. Each of these 2000 candidate regions are then reformed into a square that is fed through a DCNN that produces a fixed length (4096-dimension) feature vector for each candidate region as an output. The third and final step then uses a set of class specific SVMs to identify the object classes within each candidate region proposal [21]. An overview of the overall candidate region-proposal is visualized in Figure 10. There are however certain downfalls to R-CNN, training is very cost-inefficient since 2000 candidate region-proposals per image adds up very quickly if a large dataset is used. Testing is also very inefficient as detection for VGG16 takes 47 seconds per image using a GPU, meaning that using it in real-time is unfeasible. Selective search has fixed parameters, preventing the network to improve significantly over time and leading to bad region-proposals. All of these downfalls are addressed in the next iteration of R-CNN; Fast-RCNN [23] which will be discussed in a following sub-chapter.

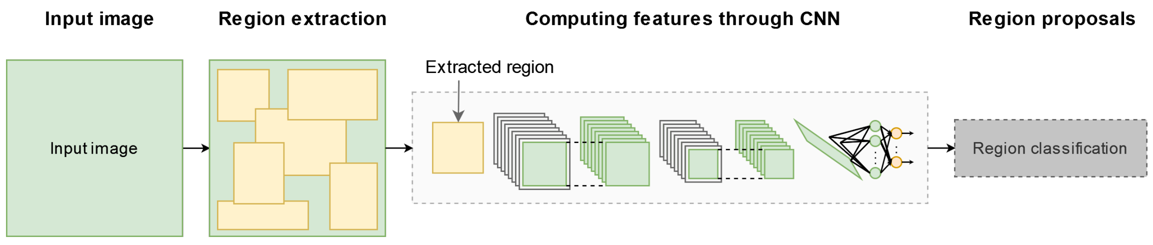


Figure 10: A high-level overview of how R-CNN generates candidate region-proposals.

Spatial Pyramid Pooling Networks (SPP-net) [24] for object detection was built as an improvement on R-CNN both in detection speed and accuracy. In its simplest form SPP-net (for object detection) is a combination of R-CNN and a Spatial Pyramid Pooling (SPP) layer added to the last convolutional layer of the network. The SPP layer pools each feature vector, generating fixed-length outputs that are then classified by FC layers, this in turn negating the need for reforming images before inputting them into the DCNN for feature extraction. SPP divides each image into multi-level spatial bins of granularity (finer to coarser) and adds their local features together using max pooling layers to form a fixed-length representation. Figure 11 illustrates the overall structure of the network with an SPP layer added to it. In contrast to R-CNNs use of sliding window, pooling means that SPP can create a set length of outputs regardless of the input size. Not only that, but SPP can also take advantage of different aspect ratios and scales, i.e., the image resolution in datasets can vary, not only for testing but also for training. However, the biggest advantage of using SPP-net over using R-CNN is the fact that SPP-net only needs to run the convolutional layer one time over each image, leading to feature extraction effectively becoming 24-102 times faster, making SPP-net viable for real-time applications [24]. However, unlike R-CNN, SPP-net cannot update and therefore not fine-tune its convolutional layers that come before the spatial pyramid pooling layer, limiting the performance accuracy in deeper networks [23].

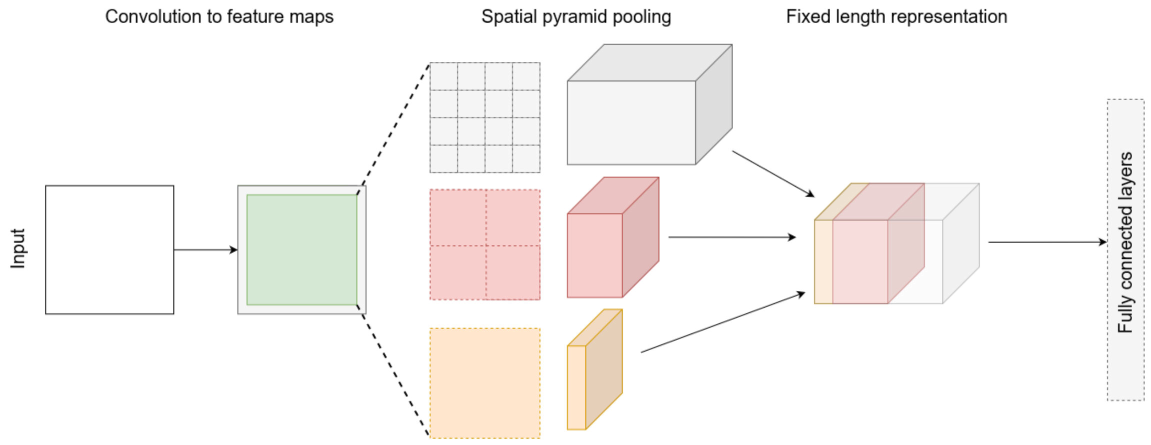


Figure 11: Spatial pyramid pooling layer structure.

Fast R-CNN [23], as was stated earlier on, is the next iteration of R-CNN (one could even consider it to be an update to SPP-net). Fast R-CNN improves upon the same problems as SPP-net does for R-CNN, such as not having to parse through 2000 candidate regions and overall having higher detection speeds. Each image is input through Fast R-CNN instead of having to go through individual candidate region-proposals and then through max-pooling layers to finally generate a convolutional feature map. Fast R-CNN is also fed several object proposals that are run through a RoI pooling layer (which reforms the image into a square) to gather fixed-length vectors from each feature map. The feature maps are then input into a series of fully connected layers. Each fully connected layer is split into two separate output layers, one that predicts the class (and background) of proposed regions with SoftMax, and the other output layer which gives the offset values of each bounding box. Since the 2000 candidate region-proposals aren't fed through the CNN anymore, and the fact that it is the images themselves that are used in the convolution operation once per image, means that the detection process is much faster than the previous two-stage networks discussed in this thesis. In fact, training VGG16 with Fast R-CNN is nine times faster than R-CNN and three times faster than SPP-net while also having an overall higher detection quality [23]. Figure 12 illustrates the overall architecture of Fast R-CNN. Even though this is a clear improvement on earlier iterations, Fast R-CNN still has one big bottleneck; how each candidate region-proposal is gathered, more specifically the fact that selective search is both slow and time-consuming influencing the performance of the overall network [25].

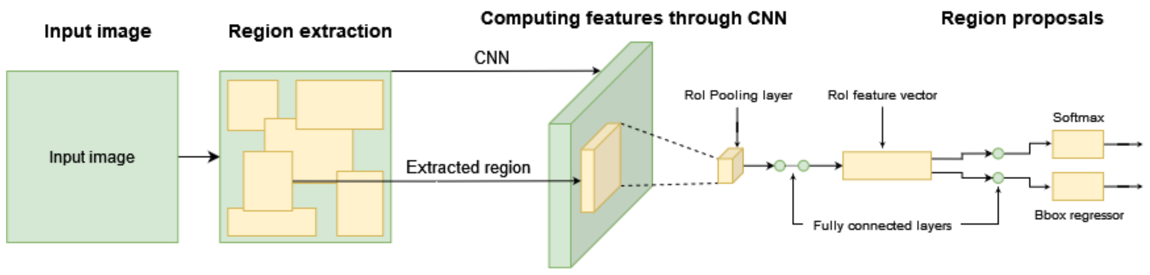


Figure 12: Overall architecture of Fast R-CNN.

Faster R-CNN [25] replaces the selective search algorithms in favour of a Region Proposal Network (RPN) which is able to share each entire images computation with the detection network, vastly improving the efficiency of identifying candidate region proposals. Faster R-CNN is similar to SPP-net in the sense that it uses a combination of two modules to effectively perform object detection [25]. Faster R-CNN is a combination of two networks, RPN to form candidate regions-proposal and Fast R-CNN that is fed the proposals forming a unified network. RPN takes in an image and inputs it to a spatial sliding window that combines a classification layer as well as a regression layer (both of which are fully connected) with a set of multi-scale anchors to process the image. The classifier determines at what probability a proposal has in it a target object while the regressor regresses the coordinates of each proposal. The anchors are layered on top of each other creating a "pyramid of anchors", the anchors themselves are used as a regression reference for the regression layer, meaning that the labels are assigned according to the anchors with the highest IoU with regards to the ground truth bounding box (IoU threshold is usually 70%). It is worth nothing that RPN is robust against translations, making it translation invariant. When training Faster R-CNN the authors use *alternate training* to share convolutional layers, which means that RPN is first trained from which Fast R-CNN uses the proposals that are gathered. The network is then switched around, i.e., Fast R-CNN is used to activate RPN, this process is repeated until the network converges. This must be done since otherwise both networks would have to be trained and compute their convolutions separately [25].

Region-based Fully Convolutional Network (R-FCN) [26], so far each two-stage detector (except R-CNN) has been *semi-convolutional*, meaning that a convolutional network first shares the computation on each entire image and then another (non-convolutional) network that identifies each candidate region. R-FCN still uses RPN for candidate region-proposal gathering, but are now applied to *position-sensitive score maps* with *position-sensitive RoI pooling layers*. The position-sensitive score maps differ from normal feature maps in the sense that each map gives a score for a sub-region of an object, such that each sub-region is unique from the previous sub-region (top-left, top-middle,... bottom-right). Position-sensitive RoI pooling layers compute the score map per unique sub-region and finally takes an average of each region. Since the network is fully convolutional, all the features are shared between RPN and R-FCN similarly to Faster R-CNN. The outstanding difference from previous two-stage detectors is the fact that the last convolutional layer in the network creates an array of position-sensitive score maps for each object class (or the background), the scores are then fed through a position-sensitive RoI pooling layer which is aggregated to form a final score of each RoI. An average is then produced from the RoIs to form the object class score, i.e., the IoU overlap on the image. R-FCN boasts 20 times faster run time over Faster R-CNN and an average precision 13% higher (comparison on PASCAL VOC 2012, using ResNet-101) [26].

2.2.3 Key-point object detectors

The two main key-point object detection approaches that will be discussed in this chapter are: corner-based and center-based approaches. This chapter is meant as an overview of these methods of detection.

Direct Sparse Sampling Network (DeNet) [27] is an RPN which estimates objects of interest based on the corners of bounding boxes in conjunction with a naive search looking for candidate region proposals. Direct Sparse Sampling (DSS) is referred to as the combination of a two-stage approach, i.e., the first part of the network does an estimation of object locations and the second identifies them as classes. Being a corner-based object detector, DeNet estimates $Pr(c_t|(x, y))$ where c_t is one of the four corner types (top-left, top-right, bottom-left, bottom-right) whereas (x, y) are the pixel coordinates on the image. A rank is then derived from the most probable top-left and bottom-right combinations in the image with a Naive Bayes classifier. To estimate new bounding boxes with their respective object classes Tychsen-Smith and Petersson uses a modified NMS. The modified NMS (called Fitness NMS) predicts the class and proposal with and added IoU estimate. Finally a *nearest RoI pooling layer* is used to extract feature vectors, the *nearest pooling layer* selects the feature that is closest to each vertex of the grid [27] similar to the nearest neighbour algorithm would.

CornerNet [28] is a key-point network that is able to predict the top-left and bottom-right corners of objects without the need for anchors to make predictions.

Law and Deng also introduces a novel pooling layer; corner pooling, to more efficiently find corners in images. The architecture of CornerNet consists of a backbone CNN (Hourglass originally), a set of prediction modules (top-left corner and bottom-right corner). The two corners are then combined into a final bounding box of the object in an image. Figure 13 shows a high-level architectural view of CornerNet. As mentioned CornerNet is able to predict object with two key-points; the top-left and bottom-right corners of an image with the help of a corner pooling layer followed by a convolutional layer that generates a set of heatmaps, an embedding and an offsets. The corner pooling layer for the top-left corner combines a horizontal max pooled (right to left) feature map with a vertical max pooled (bottom to top) feature map while the bottom-right corner pooling layer does the inverse. This is done to circumvent the problem where there is no local evidence of corners in an image (for example with round objects). The two heatmaps (one for each corner) indicate the two corners with their respective predicted object class. Each heatmap has several channels that represents a class and its width and height. The embeddings (one for each corner) predict the distance between the two corners to check if they belong to the same object, i.e, if the distance is small it most likely belongs to the same object class. The location offset helps the network readjust the distance between corners to get them closer to each other (of the same corner type) [28].

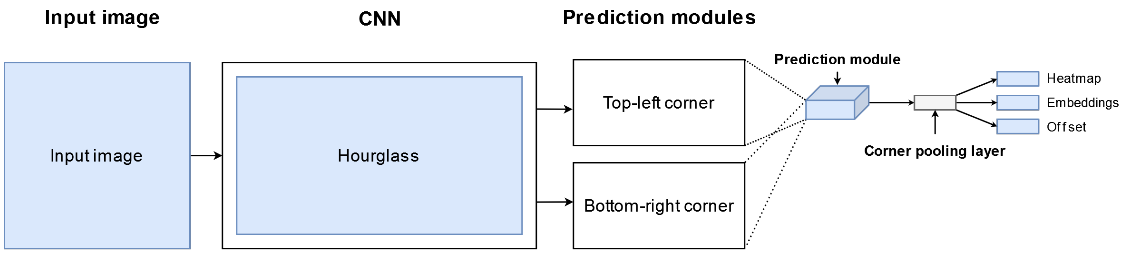


Figure 13: Overall architecture of CornerNet.

Feature Selective Anchor-free (FSAF) [29] is an add-on solution meant to improve FPN's that have two key constraints, *heuristic guided feature selection* and *overlap-based anchor sampling*. The FSAF module consists of two convolutional layers per pyramid layer of the original FPN which do the box offset predictions in an anchor-free branch. Each anchor-free branch is connected to an anchor-based branch from the original network, this branch is responsible for classifying predictions. Both the anchor-free and anchor-based branches are combined end-to-end to be able to share their features across all the levels of the pyramid. Based on the anchor-free branch FSAF is able to gather instance information more effectively as it does not need to select the instances based on size, but instead with the help of arbitrary pyramid levels based on the content [29].

CenterNet (Key Point triplets) [30] relies on CornerNet's [28] architecture, such that it predicts a set of corner-based key-points but with an added third that predicts a center-most key-point; hence the key-point triplets. Figure 14 shows the overall architectural similarities to CornerNet with the added center key-point heatmap. CenterNet also adds two novel kinds of pooling layers to improve the corner and center predictions; *Center Pooling* and *Cascade Corner Pooling*. The center pooling layer takes the maximum values (in a similar manner to CornerNet's corner pooling layer) of the horizontal and vertical feature spaces of the feature map and aggregates them. This greatly helps the network to find visual patterns of object centers in images. The cascade corner pooling layer (shown above the center pooling layer in Figure 14) takes in contrast to normal corner pooling the maximum values within a boundary of a directions, i.e., the maximum outer boundary of the top-left corner as well as the internal boundary of the bounding box and aggregating the max values. This helps the network find the visual patterns of object corners as well as their respective boundary information [30].

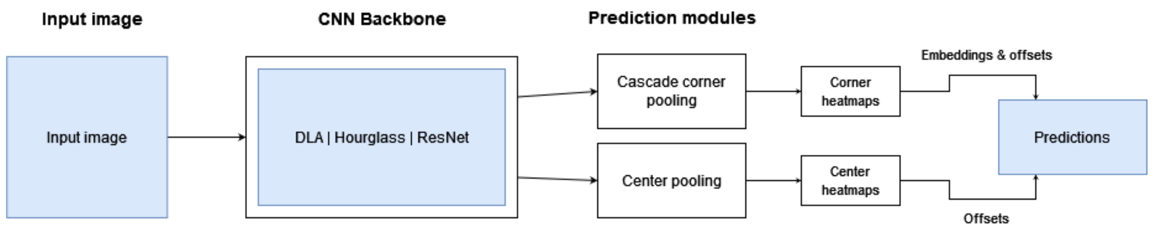


Figure 14: Overall architecture of CenterNet (Key point triplets) with the added center heatmap.

CenterNet (Objects as Points) [7] is similar to anchor-based one-stage detectors with the difference being that CenterNet uses key-point estimation with a center-based approach. The center point of an object is a *shape-agnostic anchor*. Each of these points are assigned from location instead of IoU overlap. To get the location of an object in images, CenterNet uses key-point heatmaps, which look peaks associated with the center of an object in the image. From these heatmaps CenterNet is also able to regress an estimate for each object's bounding box width and height. It is also worth noting that CenterNet only uses a single center-point for each object without having to use any post-processing and thus negates the use of NMS. CenterNet is able to use four distinct fully convolutional networks as a backbone for various tasks, such as, object detection and human pose estimation. Each backbone: Hourglass, Deep Layer Aggregation (DLA) and Residual Networks 18 and 101 (ResNet-18 & ResNet-101) have their own set of advantages and disadvantages, namely a training and inference speed versus accuracy [7]. Each backbone will be discussed in more detail in Chapter 5, particularly their respective architectures and how they have been modified for CenterNet. An illustration of the general architecture of CenterNet can be seen in Figure 15.

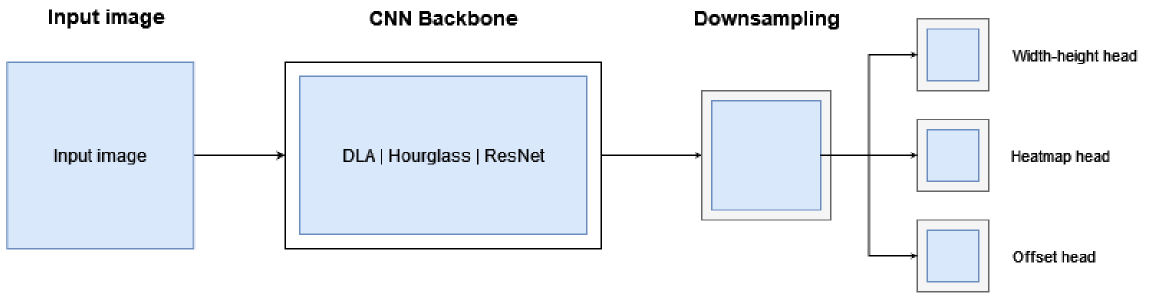


Figure 15: An overview of the general architecture of CenterNet (Objects as points).

3 Transfer Learning

Transfer learning is defined as: given a source domain and task the goal is to transfer the information from the source to a target domain and task.

An example use case of transfer learning is the process of using a model that is trained on one particular task and using it on a different task within the same domain, i.e. utilizing the knowledge gathered from the first task and adapting it to another related task. Traditionally, ML models have been created and trained to solve a specific task, this has then also entailed that models have to have been re-trained or even re-made from scratch again when the feature space has changed. Take for example a human being who has learned to walk and now has the challenging task of learning to run. The human beings do not have to learn running completely from scratch, as they transfer the previously gathered knowledge (walking) and are able to apply what they have learned to the new task of running. This, in essence, is a good analogy for transfer learning [31]. An illustration of the differences between a transfer learning and a non-transfer learning method can be seen in Figure 16.

One could then argue that there are three main issues regarding transfer learning: When should one apply it, what should be transferred and how would one transfer it? When to use transfer learning of course depends on the task at hand, usually a good reason to use it is when one has a small or limited dataset to work with, and by then utilizing the transferred knowledge from a much larger dataset. This then facilitates the necessity of transfer learning for distinct use cases and if the two datasets are related. There are also times when transfer learning can be harmful to the performance of one's model, as when using two distinct datasets that both differ in their domain-specific features. What to transfer, again depends on the task at hand. Zhang et al. give four aspects of what to transfer: homogeneous feature spaces and labels, heterogeneous feature spaces, heterogeneous label spaces and heterogeneous feature spaces and label spaces, all of which will be explained in the next sub-chapter. Finally, how to perform transfer learning comes down to when and what one wants to transfer and it depends on the approach used [31, 33, 32].

This thesis will mainly focus on transfer learning approaches for deep learning which will be discussed later on in this chapter.

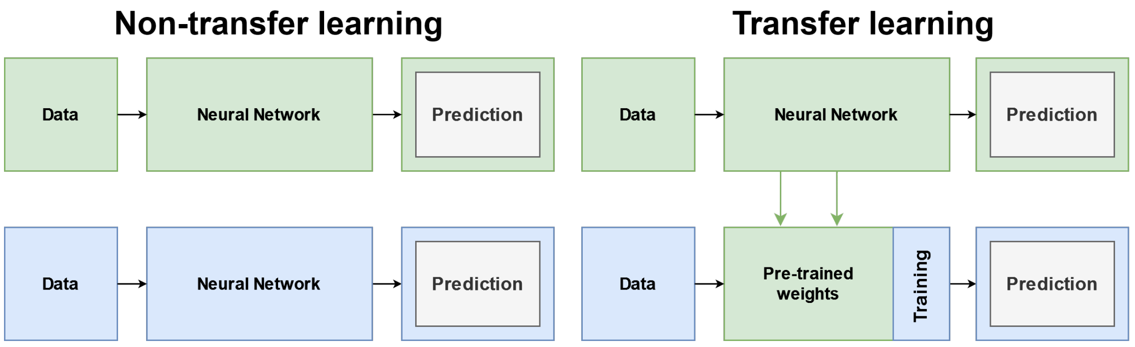


Figure 16: Illustration of transfer learning.

3.1 Formal definition and Scenarios

3.1.1 Definition

The formal definition of a **domain** is $\mathcal{D} = \{\mathcal{X}, P(X)\}$, whereas $\mathcal{X}$ denotes the feature space and $P(X)$ the marginal probability distribution in which $X = x_1, ..., x_n, x_i \in \mathcal{X}$, meaning that every x_i represents an feature vector in the marginal probability distribution [32, 33].

A **task** is defined as per a given domain $\mathcal{T} = \{\mathcal{Y}, f(x)\}$, whereas $\mathcal{Y}$ denotes the label space and $f(x)$ the predictive function. $f(x)$ can thus be seen as a conditional distribution $P(y|x)$ of which that is learned from the feature vector pairs $(x_i, y_i), x_i \in \mathcal{X}, y_i \in \mathcal{Y}$ [32, 33].

Transfer learning can then be defined as follows: give a source domain $\mathcal{D}_S$, a source task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$, a target task $\mathcal{T}_T$, the main goal of transfer learning is to learn the conditional distribution $P(\mathcal{Y}_T|\mathcal{X}_T)$ within $\mathcal{D}_T$ with the information passed on from $\mathcal{D}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$. However, as was mentioned earlier, there is often a limited amount of data in the $\mathcal{D}_T$, while $\mathcal{D}_S$ contains a vaster amount of data to be used [32].

These definitions then give rise to a set of scenarios in which transfer learning can be used.

3.1.2 Scenarios

Homogeneous feature spaces and label spaces mean that the feature spaces as well as the label spaces for both source and target domains are the same; this then entails the fact that the different datasets have to be divergent in their data distribution.

This could also be denoted as $\mathcal{X}_S = \mathcal{X}_T$ and $\mathcal{Y}_S = \mathcal{Y}_T$ following the previously stated definitions.

Heterogeneous feature spaces is a scenario where the feature spaces in the source domain and target domain differ. This is generally referred to as cross-domain adaption. In the occurrence of heterogeneous feature spaces, it can be assumed that the source domain has a sufficient amount of label data to properly transfer to the target domain.

This could also be denoted as $\mathcal{X}_S \neq \mathcal{X}_T$ and $\mathcal{Y}_S = \mathcal{Y}_T$ following the previously stated definitions.

Heterogeneous label spaces is a scenario where the source task and the target task have a differing label space, meaning that there is some shared knowledge in the domain but that target domain has a new set of labels to be learned. For example, $\mathcal{D}_S$ contains of a set class of boats (sailboat, motorboat) and the knowledge is to be transferred to $\mathcal{D}_T$ that contains another set class of boats (ship, ferry). The source domain thus transfers its knowledge to the target domain that is now able to learn a new set of classes, i.e. it is now able to predict on four classes instead of two.

This could also be denoted as $\mathcal{Y}_S \neq \mathcal{Y}_T$ and $\mathcal{X}_S = \mathcal{X}_T$ following the previously stated definitions.

Heterogeneous feature spaces and label spaces refers to a scenario where the features spaces, label spaces, source domain and target domain differ, i.e. a complete divergence in both source and target data distribution.

This could also be denoted as $\mathcal{X}_S \neq \mathcal{X}_T$ and $\mathcal{Y}_S \neq \mathcal{Y}_T$ following the previously stated definitions.

3.2 Different Approaches for Deep Learning

3.2.1 Domain Adaptation

Domain adaptation is a case of heterogeneous label spaces where the source and target domains are not the same, but where they both share the same task that they attempt to solve. This method of transfer learning could, for example, be useful for a wide variety of computer vision tasks, such as where two datasets share similarities in their objects (same kind of classes) but the backgrounds are vastly different. This means that one set of images could, for example, have a white background while another set has a crowded background. With the formal definition, this could be defined as variation in the conditional distribution between the two datasets $P(\mathcal{X}_S, \mathcal{Y}_S) \neq P(\mathcal{X}_T, \mathcal{Y}_T)$. This would be considered as standard domain adaptation where one could assume that there is enough labeled data from the source domain with a negligible amount in the target domain. This then also means that one has to rely on approximations of the conditional distribution of the target using the source domains distribution estimate. Venkateswara et al. argue that this is possible because of the correlation between the source and target domains [33].

Partial domain adaptation is an approach that is used when only a small amount of the labeled data between the source and target domain overlap. This problem is solved with a combination of two approaches: domain adaptation and zero-shot learning [32], the latter of which will be discussed later on in this chapter. In general, cases where partial domain adaption is an appropriate solution often present a distribution shift between classes in both domains [32].

3.2.2 Multi-task Learning

To better generalize on a source task $\mathcal{T}_S$ multi-task learning can be a viable option. Multi-task learning is the simultaneous transference of knowledge between source and target tasks to then be trained and learned at the same time. This means that the model in the end receives a multitude of learnable labels with each task being different. This in, essence, organizes each task together into one cohesive task where, at first, the trainable model has no clue about what the target task is (as they are all learned simultaneously). This again comes down to approximation of the conditional distribution, whereas an estimate for a joint distribution is not possible [33].

3.2.3 Zero-shot and Few-shot Learning

Zero-shot learning is a case where the target domain contains no data, whereas few-shot learning means that the target domain contains one to a few trainable target labels. For zero-shot learning this creates the problem that if the model were to be directly applied to the target domain, the model would be heavily biased towards the source domain's labels. To combat this an unlabelled dataset of the target domain is required. Zhang et al. also discussed future solutions to this problem, such as further investigation into higher-level semantic spaces for connecting known and unknown classes. In DL, few-shot learning usually suffers from overfitting, as DL models require a larger amount of information [32].

4 Maritime Vessel Datasets

In this section, I will briefly discuss some of the most prominent maritime vessel detection datasets: ABOships, Singapore Maritime Dataset, McShips, and SeaShips.

4.1 ABOships

The ABOships dataset consists in its original form (more on this in Chapter 5) of 9880 images containing 41967 annotations with 11 different object classes. All the images were taken in the Turku Archipelago and within the city of Turku along the river Aura from a camera mounted on top of a ferry. Each image in the dataset is derived from a set of 135 videos that were recorded in 720p (1920 x 720px) at 15 frames per second. The videos were recorded over a timeline of 13 days in July 2018 under a varying set of weather and lighting conditions as well as variations in background. It is worth mentioning that LiDAR data was also recorded; this will, however, not be further discussed as it has no relevance to this thesis [34].

As mentioned, the dataset was captured in varying weather, lighting and background conditions, such as, certain atmospheric conditions consisting of cloudy, rainy and sunny days, each of which also affects the lighting in each given instance, such as less lighting when it is raining or cloudy and more lighting when it is sunny. As the dataset was gathered from the top of a ferry that sailed along a set route from Turku to Ruissalo, the background in any given image can vary, for example, images from the Aura River (in the city centre) consist of a multitude of objects, such as many different boats and people which might be crowded. This also varies to another extreme, such as, open sea conditions with relatively few objects within the image. As with most quality datasets, there is also an aspect of a variety in the scales of both class objects and intra-class objects [34].

As mentioned, the ABOships dataset is annotated with 11 different object classes, whereas nine of which are a boat type and two are either a seamark or categorized under miscellaneous. The types of boats area as follows; motorboat, sailboat, misc boat, passenger ship, military ship, boat, cruise ship, ferry and cargo ship. Each annotation was annotated by hand, checked and finally relabelled with a CSRT tracker [34].

A multitude of CNNs were used to evaluate the performance of ABOships, such as, baseline one-stage detectors like SSD and EfficientDet and two-stage detectors like Faster R-CNN and R-FCN. Most of these networks have already been discussed in detail in this thesis. Results for each network that was used can be seen in Table 1, of which *Inception ResNet V2* had the overall best average precision of the CNNs tested. Each network was separately trained on Microsoft's COCO dataset and then tested on the ABOships dataset to evaluate its performance [34].

Neural network	**AP**
Faster R-CNN Inception ResNet V2	0.3518
Faster R-CNN ResNet50 V1	0.2649
Faster R-CNN ResNet101	0.3026
SSD ResNet101 V1 FPN	0.3003
SSD MobileNet V1 FPN	0.2859
SSD MobileNet V2	0.1748
EfficientNet D1	0.3383
R-FCN ResNet101	0.3246

Table 1: Overall AP results on the ABOships dataset [34].

4.2 Singapore Maritime Dataset

The Singapore Maritime Dataset (SMD) consists of three different kinds of videos: 40 on-shore, 11 on-board and 30 infrared on-shore videos, making it a total of 81 videos with a total number of 241,171 annotations [35, 3]. The videos were taken in and around Singapore waters with a resolution of 1080p (1920 x 1080px).

The dataset diversity varies in respect to atmospheric conditions as well as background. Weather conditions are as follows: Clear sky (brighter), cloudy sky (lower light) and slight foggy weather (lower visibility). Other factors were also taken into consideration, such as, blurry images (camera out of focus), open sea situations without any maritime activity and crowded situations (many boats overlapping). Scale variation is also a key point in the SMD dataset as the distance from and between boats vary greatly. Also, as the angle at which the camera is pointing, e.g., due to windy weather, the boat (that the camera is on) might rock back and forth creating a slightly tilted image [35].

There are 10 different object class annotations in the dataset, whereas seven are of maritime vessels and three are of a miscellaneous nature. Each annotation was hand annotated in MATLAB by volunteers. The object classes were categorized as follows: vessel/ship, other, ferry, speed boat, buoy, sailboat, boat, kayak, flying bird/plane and, finally, swimming person. It is worth mentioning that the dataset also includes optional fields, motion type (moving, stationary and other) and distance type (near, far and other) [35], which are not used in this thesis and thus not discussed further.

4.3 McShips

The McShips dataset consists of 14,709 images that were either collected through a web crawler or simply downloaded through a search engine. The annotations were split into two distinct categories: military ships and civilian ships with six and seven object classes respectively. Besides using a web crawler or search engine, images were also split from surveillance and ship videos. The authors also mention that certain images in the dataset have been chosen, annotated and re-annotated up to five times. The image resolution varies in the dataset, but a minimum threshold of 500^2 pixel resolution was chosen with the data diversity rule set below. In total, the dataset contains 26,529 different annotations [4], meaning that there are on average 1.8 annotations per image.

As with most quality large-scale datasets, there is a fine-grained amount of data diversity. In this context, a fine-grained dataset diversity means Zheng and Zhang has shown the utmost care for obtaining a differing set of weather conditions as well as different scenarios when creating the dataset. As such, McShips consists of a multitude of images with varying lighting conditions; during clear sky, low light scenarios such as nighttime, morning and evenings and even greyscale images. McShips also contains an impressive set of foggy states (light, medium and heavy fog) in images, some even rendering it difficult for a human being to distinguish between objects in images. As the images are taken from different points view (both geographically and anglewise), there are stark differences in both scale and the number of ships within the images. For example, while most of the images might only contain a single vessel, some might contain up to as much as 11 vessels in one image. Another aspect to consider in the McShips dataset is the scale variation. The author deliberately put in a large number of small objects (fewer than 50^2 pixels) to better classify smaller ships [4].

McShips has been manually annotated according to three distinct rules. First, each video was split into individual frames where each frame with little to no noticeable movement was removed. Then frames that had no objects or target object classes within them were removed, as they would serve no purpose. At last, each frame was annotated, and a bounding box was drawn over each object. As mentioned, McShips contains two different categories: military ships and civilian ships, each with their own labels. The military ship labels are as follows: aircraft carrier, auxiliary ship, landing ship, destroyer, submarine and missile boat. The civilian ship labels are as follows: container ship, fishing boat, passenger ship, sailboat, speedboat, tugboat and support ship [4].

To evaluate the quality of McShips a set of CNNs were trained on it and, finally, evaluated against each other. Zheng and Zhang trained three different detectors on McShips with a varying degree of tuning and changes. Faster R-CNN was trained with different layer counts (50, 101 and 152), YOLO was trained with three different versions (version 2, version 3 and version 3 SPP) and SSD was trained in its original form in conjunction with VGG16 [4]. Table 2 shows the different performance metrics per network and network configuration. ResNet152 had the overall best average precision out of all the networks tested.

Neural network	**AP**
ResNet50	0.7807
ResNet101	0.7873
ResNet152	0.7913
SSD	0.7711
YOLOv2	0.7492
YOLOv3	0.7797
YOLOv3SPP	0.6944

Table 2: Overall AP results on the McShips dataset [4].

4.4 SeaShips

The SeaShips dataset consists of 31,455 images which have been split into individual frames from a set of 10,080 video segments amounting to a total of 40,077 annotations, averaging about 1.3 labelled annotations per image. SeaShips is annotated with six different vessel types, each of which has its own distinct features and scales to better diversify the dataset. The videos themselves have been gathered from a set of onshore surveillance cameras positioned throughout the area of Hengqin Island in the city of Zhuhai, China. All in all, the video segments were recorded with 45 different cameras from 45 locations. The video interval between each video segment ranges from January 2017 to October 2018 from early morning to evening, recording at a resolution of 1080p (1920 x 1080px) [2].

Atmospheric variations, such as lighting and different weather conditions, are present in the SeaShips dataset. For example, the time of day from when the images were captured changes both the weather and lighting conditions quite drastically, for example, early morning and late evenings are much darker than images taken in the middle of the day. Since the dataset is collected in 45 different locations, SeaShips is able to have a vast diversity in background environments, for example, urban areas with many buildings and boats, rural areas with little to no buildings and boats, as well as open-sea and port environments. Since there is such a varied diversity in the dataset, there is also a big variation in both scale and occlusion levels in the images, such as port areas where there are many vessels and, therefore, a large amount of occlusion between each vessel and for the same reason a large-scale variability in the dataset. Another challenging part of the SeaShips dataset has to do with the fact that some images only show partial vessels, since the images are split from video streams. For example, some images might only have the front part of a boat visible while other images might only have the back part [2].

As many of the other datasets that have been discussed in this chapter, SeaShips has been hand-annotated following a set of guidelines. As mentioned, the images in SeaShips were gathered from video surveillance footage in the area of Hengqin Island, where a total of 168 videos from 45 locations were split into frames every two seconds, equalling a total of 302,400 images. After each video was split, each frame had to be manually checked for usable objects and if the frame had any noticeable changes from previous frames (i.e., the image was removed if it was empty or was unchanged from the previous frame). This led to a final tally of 31,455 images that were then finally annotated manually in the same format as PASCAL VOC2007. SeaShips was annotated with the following object classes: ore carrier, bulk cargo carrier, general cargo ship, container ship, fishing boat and passenger ship [2].

To evaluate the performance of the SeaShips dataset Shao et al. set out to test it with a number of CNNs that all had been pre-trained on ImageNet and then had their hyperparameters tweaked for object detection with a maritime dataset. The results for each CNN tested on SeaShips can be seen in Table 3, where ResNet101 has the overall best average precision [2] out of all the networks tested.

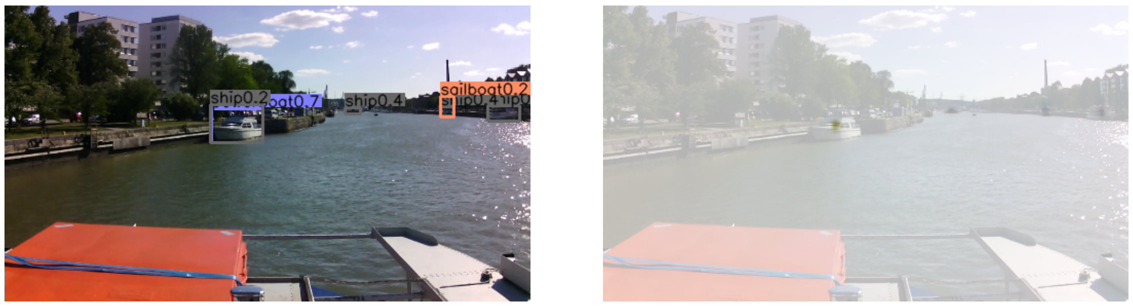
Neural network	**AP**
Fast R-CNN VGG16	0.7103
Faster R-CNN ZF	0.8916
Faster R-CNN VGG16	0.9012
Faster R-CNN ResNet18	0.9063
Faster R-CNN ResNet50	0.9165
Faster R-CNN ResNet101	0.9240
SSD 300 MobileNet v1	0.7766
SSD 608 MobileNet v1	0.7950
SSD 300 VGG16	0.7937
SSD 512 VGG16	0.8673
YOLO v2	0.7906

Table 3: Overall AP results on the SeaShips dataset [2].

5 CenterNet evaluation on ABOships

5.1 CenterNet overview

CenterNet is an anchor free object detection network that relies on keypoint heatmaps to properly detect object classes in images, more specifically it uses the center-most keypoint of an object to find the bounding box coordinates. CenterNet does not use the conventional methods of object detection, regional classification or anchor bases methods, but instead uses key-points heatmaps peaks to predict centers of objects in images. From the center-most heatmap, CenterNet is able to predict the width and height of each object without the need of NMS. Each heatmap peak is also heavily correlated with any particular object class, meaning that the network is effectively able to use the key-points, object dimensions and object class probabilities to detect any given object class that it has been trained on [7]. Figure 17 shows the predictions of each object in an image as well as their respective heatmaps. The next sub-chapter will go into more detail about each backbone that is available with using CenterNet.



(a) Predictions (b) Heatmaps

Figure 17: Predictions with their respective center-most heatmaps side-by-side.

5.1.1 Different backbones

As was mentioned in Chapter 2.2.3, CenterNet is modular in the sense that the backbone of the network is interchangeable out of four distinct architectures. Each of which have their own use case scenario, either being more accurate or faster. Some of these architectures are in their original form while others have been modified with added deformable convolutional layers.

Deep Layer Aggregation: DLA-34 is an upstream aggregation of layers, specifically for deeper networks. As such DLA is better able to gather semantic information and features. With the aggregate nature of DLA, it is able to create a fusion of both semantic and spatial information. Another advantage of DLA is the use of *skip-connections* which further improves upon the information sharing between layers, Figure18 illustrates the usage of skip-connections between layers. There are two variations of DLA: Iterative Deep Aggregation (IDA) and Hierarchical Deep Aggregation (HDA). IDA stacks the shallowest parts of the network into blocks, each block placed in a distinct division depending on the feature resolution of the block. Skip connections from each block are then connected and merged with the scale and resolution of each block aggregated together. Because of the aggregation already beginning at the shallowest part of the network and iteratively merged with deeper layers, DLA manages to enhance the shallow features as the network propagates through its layers. Figure 18(a) illustrates the iterative approach of IDA. Instead of iteratively aggregate layers, HDA does it in a treelike manner. With HDA both shallow and deeper layers are merged throughout the network, this is done to preserve the features from shallower features in combination with a waster feature diversity [36]. Figure 18(b) illustrates the hierarchical approach of HDA. CenterNet uses an combination of both IDA and HDA, shown in figure 18(c) with added skip-connections to the shallowest layers and every convolutional layer changed to deformable convolutional layers in the upsampling part of the network [7].

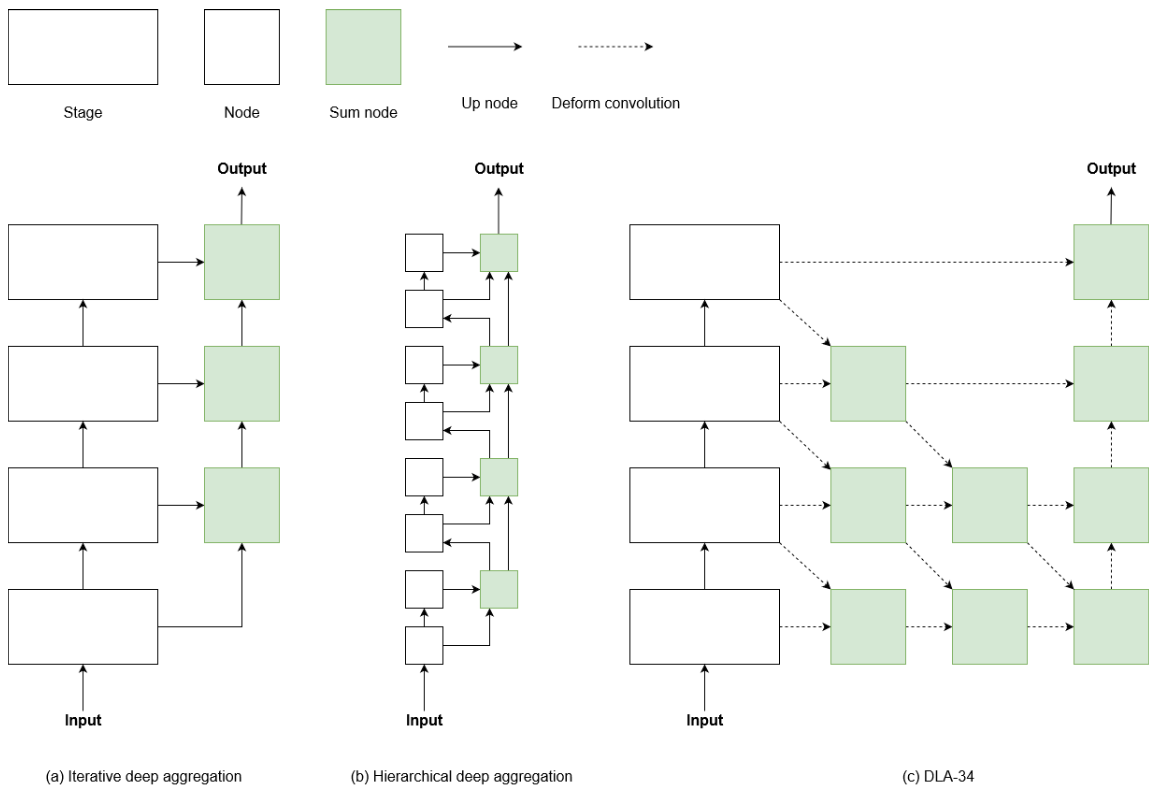


Figure 18: Three different variations of DLA: Iterative deep aggregation, Hierarchical deep aggregation and DLA-34 which is a modified version of a normal DLA structure used in CenterNet.

Stacked Hourglass is a set of *stacked* Hourglass modules which are laid out in succession. An hourglass module, which can be seen in Figure 19, is designed in a symmetrical fashion with a set of convolutional layers and max pooling layers for down-sampling of image resolution followed by a set of *nearest neighbor upsampling* layers to upscale the images. For each iterative step through the network in the upsampling stage the network is split and combined with an additional convolutional layer. Reference to Figure 19 for a visualization of the splitting from upsampling to downsampling layers. After reaching the middle part of the network (lowest resolution) the network starts up-sampling and combining each feature set (of each resolution scale). The up-sampling is done by combining two resolutions that are adjacent to each other (one down-sampled and one up-sampled) with a *nearest neighbor upsampling* of the down-sampled resolution which then is combined with an element-wise addition. The Stacked Hourglass architecture is a consecutive stack of multiple Hourglass modules, whereas each pass through a module the network is able to apply a loss. This allows the stacked network to create an iterative inference slope to be able to reconsider each feature across images. No modifications were made to the Stacked Hourglass architecture with regards to how it operates as a backbone for CenterNet [37].

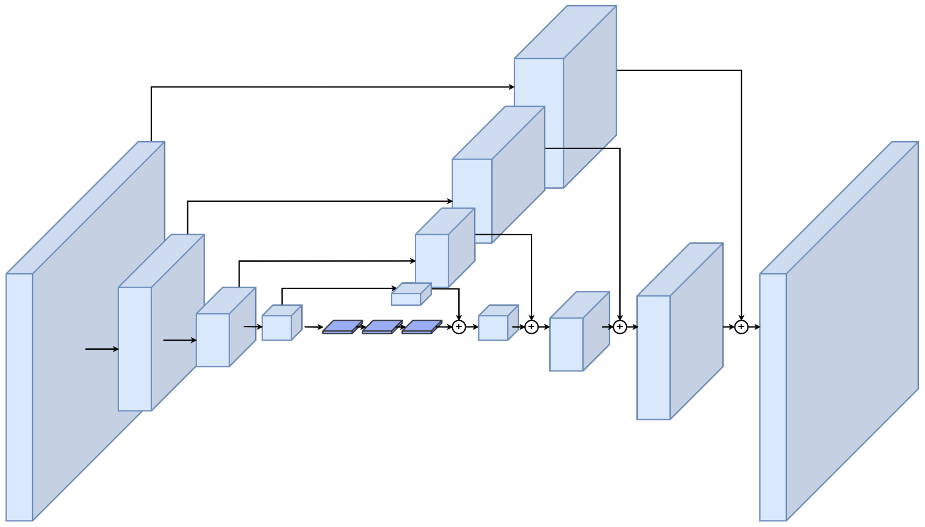


Figure 19: A single *stack* of the stacked hourglass architecture. Each rectangle represents a residual block, similar to that of Figure 20.

Residual Neural Networks: ResNet101 & ResNet18 are a class of DC-NNs with a set of identity mappings (skip-connections) placed in between a certain amount of layers. He et al. brings up the point of the exploding gradient problem when it comes to very deep networks, such as the one in their article that is made up of 56 layers. The author however argues that the use of Batch Normalization helps to alleviate the exploding gradient, the true solution to the problem is the use of *Residual Blocks*. The Residual Block implements the identity mapping which works as a shortcut between one layer to another with no added parameters. To combat the issue with different resolution scales between layers an element-wise addition is used to be able to combine the layers [20]. Figure 20 illustrates the use of an skip-connection between a set amount of layers. With these Residual Blocks ResNet's are able to incorporate a much larger number of layers, such as ResNet-101 in CenterNet that has 101 layers. Modifications made in the ResNet architectures for CenterNet: A deformable convolutional layer is added before each up-sampling phase [7].

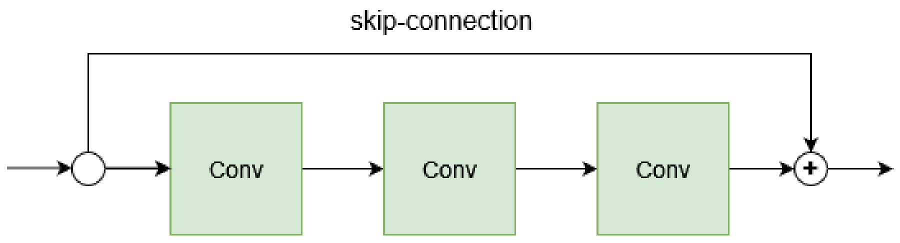


Figure 20: A residual block with a skip-connection.

5.2 ABOships dataset - Exploratory analysis of annotations

ABOships [1] is a dataset consisting of multiple instances of both inshore and offshore imagery with a large set of maritime object classes. The dataset was gathered over a duration of 13 days from a route that extended from the city of Turku (Aura River) all the way to Finnish Archipelago of Ruissalo. The images from which the dataset is derived from were split into frames from 135 videos at 720p videos each of which were filmed at 15 frames per second. The dataset consists of multiple varying environments, such as, urban areas from the Aura River and open sea conditions from Ruissalo both of which were captured in different conditions. These different conditions consist of sunny weather, rainy weather, and clouded weather. All these conditions meant that the dataset was able to be gathered in different lighting conditions. ABOships consists of 9880 images with 41967 annotations roughly equalling an average of four object annotations per image, whereas each annotation can be any one of 11 different maritime classes [1, 34]. The class disparity can be seen in Table 4 with example images of each class in Figure 21.

As it is described in [34], the the 11 classes of the ABOships dataset are as follows: motorboat, sailboat, seamark, miscboat, passengership, militaryship, boat, cruiseship, ferry, cargoship and finally miscellaneous, all of which have a visual description in Table 4.

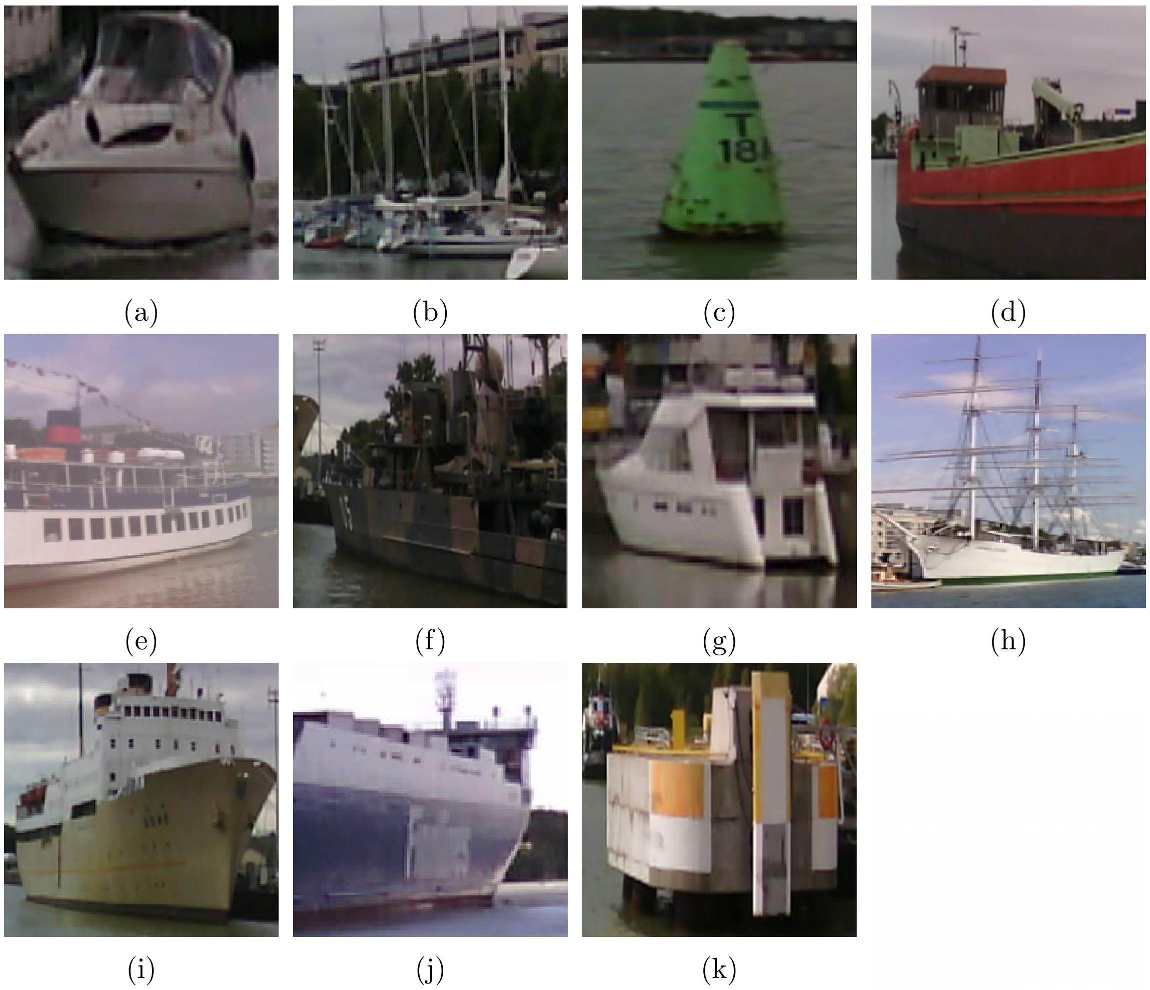


Figure 21: Example images of each class in the ABOships dataset: motorboat (a), sailboat (b), seamark (c), miscboat (d), passengership (e), militaryship (f), boat (h), cruiseship (i), ferry (j), cargoship (k), miscellaneous (l).

Class	**Images**	**Objects**	**Visual description**
Motorboat	4062	7092	Sleek, aerodynamic features
Sailboat	3842	8147	Sails
Seamark	3744	7670	Cone shaped floater or pipe
Miscboat	2797	4642	Generic boat
Passengership	2639	4464	Medium size, multiple lateral windows
Militaryship	2559	4128	Special hull with antennas, dark hue
Boat	2034	2913	Oval shaped, small size
Cruiseship	1347	1504	May contain passengers and/or cars on board, large size
Ferry	945	1046	Medium size, entrance on two sides, cabin in the middle
Cargoship	157	161	Large size, long, cargo containers
Miscellaneous	129	200	Random floaters

Table 4: Amount of images and objects in each class type with a visual description from the ABOships paper.

After a descriptive analysis over the dataset it was evident that there was a high number of very small bounding box areas ($area < 16^2$) that were annotated in the ABOships dataset, specifically there were 8740 annotations with a bounding box area of smaller than 16^2 or only 256 pixels occupied by any given bounding box. The amount of annotations in each bounding box area can be seen Table 5. For reference each bounding box area is set accordingly:

- extra small = $area < 16^2$
- small = $16^2 < area < 32^2$
- medium = $32^2 > area > 96^2$
- large = $area > 96^2$

Bbox area	**Annotations**
extra small	8740
small	10061
medium	15959
large	7207

Table 5: Amount of annotations per bounding box area in the ABOships dataset.

Given the very small size of objects, every image with a bounding box area less than 16^2 was removed and consequently removed images which only contained small bounding box areas with no other bounding box area size annotations in the image. This reduced the dataset from 9880 images and 41967 annotations to 8377 images and 33227 annotations. Another change that was done was the aggregation of classes into larger subsets of classes. Specifically, each class were more or less divided into bounding box area sizes forming the following new classes: powerboat, sailboat, ship and stationary which is a combination of the classes seamark and miscellaneous. The following combinations of classes were done:

- boat + motorboat = powerboat
- passengership + cargoship + ferry + militaryship + cruiseship + miscboat = ship
- seamark + miscellaneous = stationary
- sailboat stayed the same

Table 6 shows the new distribution of the amount of images as well as objects per class type.

Class	**Images**	**Percentage**	**Objects**	**Percentage**
Powerboat	4044	48%	7244	22%
Sailboat	3756	45%	8029	24%
Ship	5887	70%	15272	46%
Stationary	2151	26%	2682	8%

Table 6: Amount of images and objects in each class type for the new dataset as well as their respective percentages.

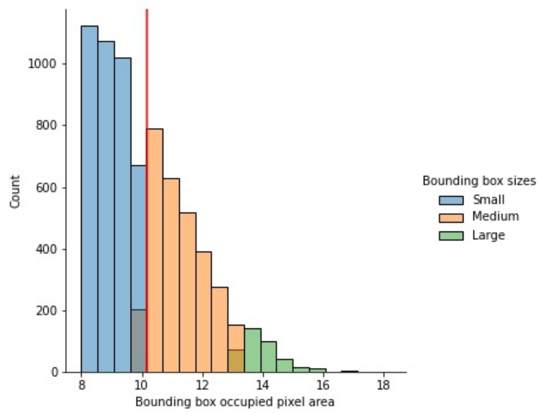
The following table, Table 7 serves to show the amount of bounding boxes per bounding box area size per object class type. This shows that there is a great deal of variety in the bounding box sizes both between and withing object classes. For example, most of the bounding boxes for the class *powerboat* falls within the bounding box sizes small to medium while sailboat is skewed mostly from medium to large bounding box sizes. It is also worth noticing that most of the bounding boxes for the object class *stationary* is of the bounding box area size small, since most of the objects in that class are of smaller seamarks.

Class	**Small**	**Medium**	**Large**
powerboat	3887	2959	398
sailboat	662	4055	3312
ship	3564	8247	3461
stationary	1948	698	36

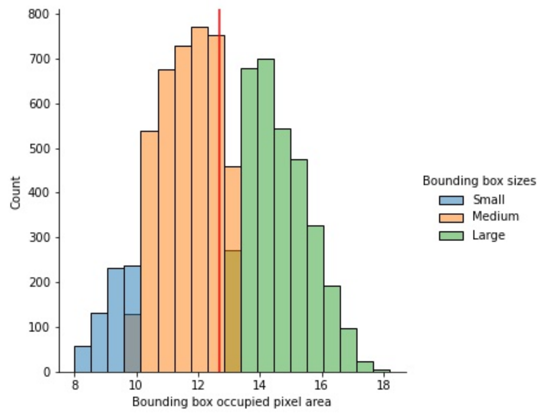
Table 7: Amount of bounding boxes per bounding box area size per class in ABO-ships.

Figure 22 shows the bounding box occupied pixel area distribution between the four new classes in ABOships in $log_2 - scale$. The figure shows each individual object class with bounding box occupied pixel areas grouped into three distinct groups: small, medium and large. As each occupied pixel area in the figure has been transformed to a logarithmic distribution, the areas fall into a distribution between 8 and 18 occupied pixels. The bounding box areas in the figure are as follows:

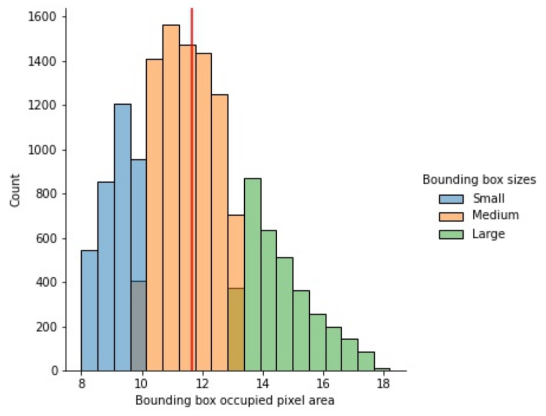
- small = $log_2(area) < 10$
- medium = $10 < log_2(area) < 13.16$
- large = $log_2(area) > 13.16$



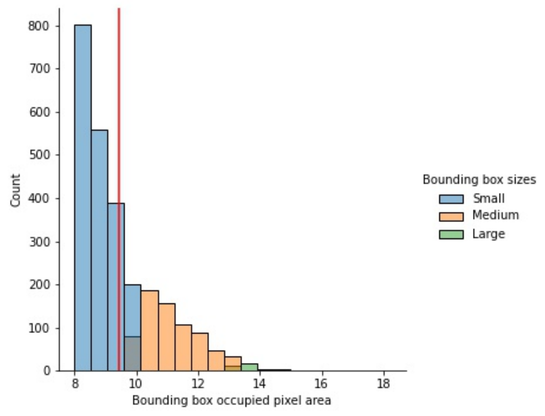
(a) Powerboat



(b) Sailboat



(c) Ship



(d) Stationary

Figure 22: Occupied pixel area distribution in $log_2 - scale$ of bounding-boxes in each object class in ABOships (includes aforementioned superclasses: powerboat, sailboat, ship and stationary). Each bounding box area size is divided into three different colors: small colored blue, medium colored orange and, finally, large colored green. The red line represents the mean distribution of the bounding-box occupied pixel area for each specific class.

Table 8 shows the disparity of the amount of bounding boxes (of each bounding box area) per image. There is a very large number of images that have zero bounding box areas of any one of the bounding box sizes; small medium or large. This does however not mean that these images without are without any bounding boxes whatsoever, this is just meant to illustrate that there are images which do not contain any one of the three bounding box area sizes within them, while still containing any of the other two bounding box areas.

Finally, the ABOships dataset was adapted to the COCO standard format for

Bbox nr	**Small**	**Medium**	**Large**
0	2810	1990	4443
1	2866	2322	2172
2	1578	1694	955
3	714	1024	438
4	253	598	207
5	93	299	89
6	39	200	29
7	13	93	19
8	6	73	9
9	4	37	8
10	0	26	4
11	1	9	2
12	0	4	1
13	0	2	1
14	0	2	0
15	0	3	0
16	0	1	0

Table 8: Amount of images with 0 to 16 bounding boxes within them depending on the bounding box size.

datasets. This was done by a custom script that first randomly splits all the images in the ABOships dataset into a training (70%) validation (15%) and test (15%) set. After the split the annotation file of the original dataset was parsed in correlation to the images that were placed in their respective sets and produced in a similar split. Then the dataset annotation files were converted to JSON from CSV to work with CenterNet.

5.3 Evaluation methods

To evaluate the performance of CenterNet on the dataset two metric were employed, Intersection over Union (IoU) and Average Precision (AP) both of which complement each other (as they can be used in combination to get a more detailed evaluation of a given dataset). IoU is the area of which the ground truth labels, and the predicted labels overlap over each other. The higher magnitude of overlap equals a higher IoU [38]. Figure 23 illustrates how the intersection over union is gathered from the division of the area of intersection and of the union of both the precision and ground truth.

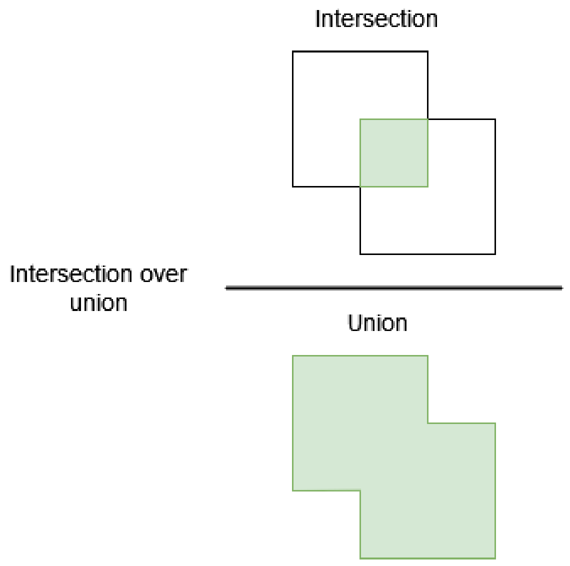


Figure 23: Intersection over Union illustration

Equation 1 explains the above image in a more formal definition, whereas the intersection of the predictions bounding box B_p and the ground truth bounding box B_{gt} taken as an absolute value is divided by the union of the same bounding boxes in their respective absolute values [38].

$$IoU = \frac{|B_p \cap B_{gt}|}{|B_p \cup B_{gt}|} \tag{1}$$

AP can be calculated given that the precision and recall is calculated beforehand and is then done with the following formulae [38]:

$$AP_{11} = \frac{1}{11} \sum (R \in \{0, 0.1, ..., 0.9, 1\}) P_i(R) \tag{2}$$

Whereas precision is

$$P = \frac{TP}{TP + FP} \tag{3}$$

And recall is

$$R = \frac{TP}{TP + FN} \tag{4}$$

With the maximum precision $P_i(R)$ over a recall value of over R

$$P_i(R) = max(R * |R* \geq R) P_i(R*) \tag{5}$$

5.4 Evaluation of COCO-pretrained CenterNet

To examine the out-of-the-box performance of CenterNet on ABOships (pre-trained on the COCO dataset), I re-labeled all maritime vessel annotations in the dataset into one super-class ”boat”, and calculated the average precision with an IoU of 50 for each backbone variant, see Table 9.

Feature Extractor	**Small AP50**	**Medium AP50**	**Large AP50**	**All AP50**
DLA	0.184	0.258	0.429	0.346
Hourglass	0.256	0.322	0.452	0.375
ResNet101	0.16	0.267	0.444	0.309
ResnNet18	0.197	0.251	0.362	0.264

Table 9: CenterNet trained on the COCO dataset [6] with each backbone.

5.5 Training

The ABOships dataset was used to train each backbone of CenterNet. The dataset was randomly split into a ratio of 70% training data, 15% validation data and 15% test data. For ABOships this meant that there were 5864 images for training the network, 1257 images for validation and 1257 images for testing out of a total of 8377 images. The dataset was, as previously stated in the thesis, converted to the COCO format to be able to use the CenterNet. The number of annotations for the dataset can be found in the overview chapter, which have previously been discussed. The dataset was trained on a single RTX 2080 Ti with 11GB of VRAM (memory) and 4352 CUDA cores with CUDA Version: 11.3 and PyTorch version 1.1.0 running on Ubuntu 20.10 in a custom docker container (Docker version 20.10.5, build 55c4c88). The models were trained on ABOships from scratch without freezing any layer and without using transfer learning as well as using transfer learning to load in pre-trained weights from models trained on COCO, there were also scenarios where parts of the network were frozen, however these results were not significant enough to be discussed in the thesis. Training each backbone with ABOships in total took 99 hours to complete, Hourglass took by far the longest out of all the backbones, about 55 hours to train, i.e., over 50% of the total training time. All training was done with the following parameters: input resolution of 512x512, training for 200 epochs, at epoch 80, 100 and 125 the learning rate reduced by 10% with an initial learning rate of 1.25e-4 with the Adam algorithm as optimizer.

5.6 Results

Results are gathered from eight (8) different models, i.e., DLA, Hourglass, ResNet101, ResNet18, using both training from scratch and transfer learning for ABOships. Each sub-chapter presents the results either from the models trained from scratch or trained with transfer learning. Furthermore, each sub-chapter illustrates three unique tables, one with AP for all object classes enclosing different augmentation types during testing, one based on occupied pixel area of bounding-boxes and finally AP results for each individual object class. This means that there are a total of six (6) tables of results worth discussing. Description of the table containing AP for all object classes is as follows:

1. Average precision and its target IoU: AP (IoU between 0.50-0.95), AP50 (IoU 0.50) and AP75 (IoU 0.75)

2. Feature extractors: DLA, Hourglass, ResNet101, ResNet18

3. The AP results: N.A (Non-augmented, meaning that the images have been left untouched), F (Flipped, meaning that the images are flipped) and MS (Multiscaled, meaning that the test is ran multiple times each with a different user defined scaling of the images).

The second table is similar in the manner that it is shows the AP and IoU threshold each column comes under, further clarify how the bounding box area specific tables are structured:

1. Average precision and its target IoU: AP50 (IoU 0.50) and AP75 (IoU 0.75).

2. Feature extractors: DLA, Hourglass, ResNet101, ResNet18.

3. Bounding box area size: small, medium and large (these were discussed in detail in Chapter 5.2).

Finally the object class specific table is formatted in a similar manner as the tables before with added granularity in the form of class specific results:

1. Average precision and its target IoU: AP50 (IoU 0.50).
2. Feature extractors: DLA, Hourglass, ResNet101, ResNet18.
3. Object class: powerboat, sailboat, ship and stationary.

5.6.1 Evaluation of training with transfer learning

The following six tables show the results for CenterNet trained on ABOships with transfer learning where pre-trained weights from Microsoft's COCO dataset [6] were used. The tables are grouped as follows: first ABOships AP for all object classes with different augmentations, then object bounding box area specific table and finally AP per object class.

As each backbone were first pre-trained on the COCO dataset [6] they were able to produce respectable AP results, especially for Table 11 where large objects in the ABOships dataset had an AP of 81.7% for DLA and up to 82.5% for Hourglass and ResNet101 with an IoU threshold of 0.50. However, we can see a clear difference between the other sizes in the table, in AP50 small and medium have worse AP and large. This is even more pronounced in AP75 where the AP between small and large is 42.45%. This illustrates that the occupied pixel areas of different objects plays a significant role in the average precision that the network can provide. Table 10 shows clear improvements in the testing when using different augmentations, this is more prominent when using both flipped images as well as multiscaling. Overall, CenterNet with DLA backbone shows the most promise out of all the backbones when trained on ABOships with an AP (IoU 0.50-0.95) of 36.3%, this is interesting as it took much longer to train Hourglass while still showing similar AP results.

Table 12 shows the class specific AP results per class object in the ABOships dataset, these results were tested with an IoU threshold of 50 with no test augmentations done, i.e., the images were non-augmented. The results show a fairly even spread of AP throughout each class, with sailboat having the highest overall average. Interestingly, CenterNet with DLA as feature extractor was able to distinguish between the ship class much better than any other backbone, while also having the worst performance for the stationary class.

	AP			**AP50**			**AP75**		
Feature Extractor	**N.A**	**F**	**MS**	**N.A**	**F**	**MS**	**N.A**	**F**	**MS**
DLA	0.320	0.335	0.363	0.674	0.686	0.731	0.249	0.272	0.307
Hourglass	0.328	0.342	0.359	0.679	0.693	0.712	0.277	0.295	0.311
ResNet101	0.316	0.331	0.349	0.674	0.698	0.723	0.241	0.263	0.289
ResNet18	0.286	0.303	0.326	0.638	0.662	0.698	0.206	0.222	0.250

Table 10: AP results from CenterNet pre-trained on COCO and then trained on ABOships. N.A. (non-augmented), F (flipping), MS (multiscale augmentation) measured at IoU 0.50:0.95, IoU 50 and IoU 75.

	AP50			**AP75**		
Feature Extractor	**Small**	**Medium**	**Large**	**Small**	**Medium**	**Large**
DLA	0.504	0.652	0.817	0.122	0.324	0.582
Hourglass	0.477	0.580	0.825	0.123	0.298	0.560
ResNet101	0.428	0.604	0.825	0.089	0.293	0.560
ResNet18	0.426	0.577	0.822	0.196	0.234	0.526

Table 11: AP results from CenterNet pre-trained on COCO and then trained on ABOships. Measured at three different bounding box area sizes: small, medium and large at IoU 50 and IoU 75.

	AP50			
Feature Extractor	**Powerboat**	**Sailboat**	**Ship**	**Stationary**
DLA	0.695	0.719	0.784	0.641
Hourglass	0.702	0.713	0.663	0.738
ResNet101	0.665	0.717	0.617	0.731
ResNet18	0.594	0.714	0.717	0.655

Table 12: Class-specific AP results from CenterNet pre-trained on COCO and then trained on ABOships. Measured for each specific object class in ABOships, i.e., powerboat, sailboat, ship and stationary at IoU 50.

5.6.2 Evaluation of training from scratch

The following three tables show the results for CenterNet trained on ABOships without transfer learning, meaning that it was trained from scratch with each backbone. The tables are grouped by AP for all object classes with different augmentations and then occupied pixel area of bounding boxes.

When training CenterNet on ABOships from scratch and looking at the results from all object classes, Table 13 illustrates that DLA backbone is able to extract features better than when training the network with pre-trained weights (transfer learning). This contrasts with the rest of the backbones showing worse AP on average across the board. It would be expected that using a much larger dataset and model to pre-train the network would yield significantly better results in every scenario, which clearly is not the case. In Table 14 we can see somewhat similar results in AP50 as we saw in Table 11 for larger objects. However, what is very surprising is the fact that DLA-34 in AP75 was able to double its AP when trained from scratch in comparison to being trained with pre-trained weights, this is most likely the reason for why DLA yields and overall better AP when trained from scratch. Otherwise, the backbones seem to show worse results on average when trained from scratch (other than DLA-34).

Table 15 shows a fairly even distribution between each backbone with DLA performing the best on average across all object classes. Interestingly the object classes ship and stationary gave better results when trained from scratch than with transfer learning, this could be that the network was able to learn better features from an early stage when trained from scratch. This could also be because COCO (the dataset CenterNet was pre-trained on) mostly contains smaller boats such as motorboats. CenterNet is also most likely able to utilise more generic object features such that would benefit the stationary object class. This is likely the reason as to why CenterNet is able to perform better with DLA as a backbone when trained from scratch.

	AP			**AP50**			**AP75**		
Feature Extractor	**N.A**	**F**	**MS**	**N.A**	**F**	**MS**	**N.A**	**F**	**MS**
DLA	0.324	0.340	0.371	0.676	0.692	0.732	0.261	0.285	0.334
Hourglass	0.293	0.306	0.336	0.637	0.653	0.682	0.225	0.244	0.289
ResNet101	0.291	0.305	0.336	0.638	0.652	0.698	0.212	0.230	0.274
ResNet18	0.273	0.290	0.328	0.611	0.641	0.689	0.190	0.211	0.263

Table 13: AP results from CenterNet trained on ABOships from scratch. N.A. (non-augmented), F (flipping), MS (multiscale augmentation) measured at IoU 0.50:0.95, IoU 50 and IoU 75.

	AP50			**AP75**		
Feature Extractor	**Small**	**Medium**	**Large**	**Small**	**Medium**	**Large**
DLA	0.480	0.604	0.816	0.227	0.315	0.558
Hourglass	0.528	0.648	0.809	0.138	0.322	0.577
ResNet101	0.290	0.574	0.838	0.136	0.197	0.537
ResNet18	0.365	0.582	0.830	0.182	0.191	0.525

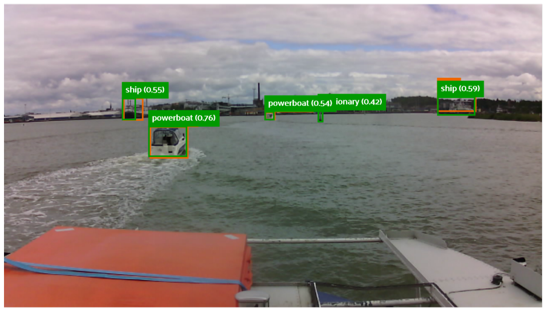
Table 14: AP results from CenterNet train on ABOships from scratch. Measured at three different bounding box area sizes: small, medium and large at IoU 50 and IoU 75.

	AP50			
Feature Extractor	**Powerboat**	**Sailboat**	**Ship**	**Stationary**
DLA	0.681	0.712	0.796	0.726
Hourglass	0.677	0.715	0.722	0.784
ResNet101	0.598	0.718	0.728	0.778
ResNet18	0.573	0.712	0.718	0.701

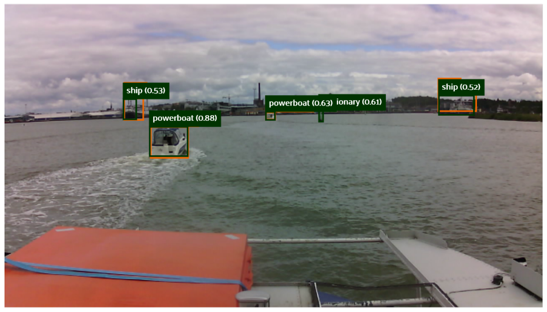
Table 15: Class specific AP results from CenterNet trained on ABOships from scratch. Measured for each specific object class in ABOships, i.e., powerboat, sailboat, ship and stationary at IoU 50.

Figure 24 shows qualitative results over each backbone and either being pre-trained with COCO or trained from scratch respectively. In both figures we can see that there is an annotation error in ABOships, a stationery object has been detected by all pre-trained models. This is an issue that is often encountered in object detection/classification datasets, see [39].

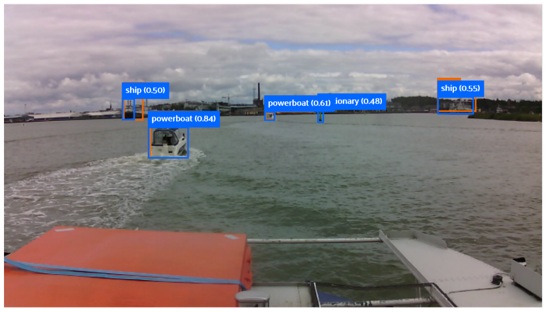
In this particular image used for testing, the pre-trained model that was able to perform the best was Hourglass. All of the backbones were able to detect the object labelled as *stationary* even though it was not annotated in ABOships. In Figure 24 the worst performing backbone was ResNet18, which is not surprising as this backbone has also the overall worst AP out of all the backbones.



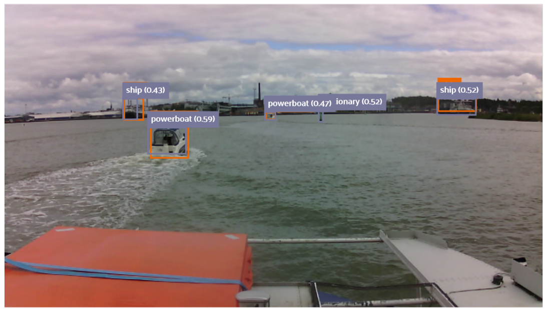
(a) DLA-34



(b) Hourglass



(c) ResNet101



(d) ResNet18

Figure 24: Qualitative detection results on the ABOships dataset using CenterNet and transfer learning starting from COCO-pretrained weights. Each image shows the performance of CenterNet with a distinct feature extractor each with its own color: (a) DLA colored in green, (b) Hourglass colored in dark green, (c) ResNet101 colored in blue, and (d) ResNet18 colored in purple. Each bounding box shows the corresponding object class and confidence score within the interval [0, 1].

6 Conclusion

This thesis investigates the benefits of deep learning and transfer learning for maritime ship detection from inshore and offshore imagery. The main dataset used in this thesis, ABOships, was trained with CenterNet on its four distinct backbones. The goal was to find investigate the performance of a key-point detector like CenterNet on a maritime dataset, as well as to exploit transfer learning and examine the training performance on smaller datasets. Each trained model was evaluated against each other with their respective AP over different IoU thresholds.

Both deep learning and transfer learning show promising results in the maritime vessel sphere. However, the improvements seen with transfer learning were not as drastic as I would have thought especially for CenterNet trained with DLA as feature extractor with an IoU of 50, as this showed worse results when trained on pre-trained weights.

The biggest improvement can be seen when comparing a network trained on a generic dataset such as COCO in Table 9 versus the results when the network is trained specifically for maritime objects in Tables 10 and 13. Overall, transfer learning yielded the best results when applying the pre-trained weights from CenterNet trained on COCO to CenterNet being trained on ABOships.

Deep learning with key-point detector gives very satisfactory results, especially when considering that the network is not only accurate but also very fast, both in training and inference. This shows that key-point detectors are a very viable solution for real-time problems, such as, traffic monitoring or border control. These kinds of methods and solutions could save costs in the sense that not as much manpower is needed for manual labour, such as, monitor surveillance cameras. Instead of manual labour a layer of automation can be put upon the existing surveillance structure with automatically detecting vessels in the context if this thesis.

6.1 Future work

A set of methods for combining datasets automatically, i.e., ordering images and their respective class objects, as well as correctly ordering and labelling annotation data would be beneficial for future research regarding any form of computer vision problem, not just for object detection. For example, a one click solution for combining ABOships and SMD in the context of this thesis would be highly beneficial to further improve results of deep learning for maritime vessel detection, especially in the case of smaller datasets. This would of course then mean that the datasets that are going to be merged/combined must be of the same format (in this case COCO), which could also be automated depending on if the original format is also known beforehand. Combining datasets could be considered another specific case of transfer learning. Moreover, several other training configurations can be implemented (different optimizers) along with different changes in the network architecture and loss functions.

References

[1] B. Iancu, V. Soloviev, L. Zelioli, and J. Lilius, "Aboships-an inshore and offshore maritime vessel detection dataset with precise annotations," Mar 2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/5/988/htm

[2] Z. Shao, W. Wu, Z. Wang, W. Du, and C. Li, "Seaships: A large-scale precisely annotated dataset for ship detection," *IEEE Transactions on Multimedia*, vol. 20, pp. 2593–2604, 2018.

[3] D. K. Prasad, D. Rajan, L. Rachmawati, E. Rajabally, and C. Quek, "Video processing from electro-optical sensors for object detection and tracking in a maritime environment: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 8, pp. 1993–2016, 2017.

[4] Y. Zheng and S. Zhang, "Mcships: A large-scale ship dataset for detection and fine-grained categorization in the wild," in *2020 IEEE International Conference on Multimedia and Expo (ICME)*, 2020, pp. 1–6.

[5] E. Gundogdu, B. Solmaz, V. Yücesoy, and A. Koç, "Marvel: A large-scale image dataset for maritime vessels," in *Computer Vision – ACCV 2016*, S.-H. Lai, V. Lepetit, K. Nishino, and Y. Sato, Eds. Cham: Springer International Publishing, 2017, pp. 165–180.

[6] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision.* Springer, 2014, pp. 740–755.

[7] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," *arXiv preprint arXiv:1904.07850*, 2019.

[8] X. Wu, D. Sahoo, and S. C. Hoi, "Recent advances in deep learning for object detection," *Neurocomputing*, vol. 396, pp. 39–64, 2020.

[9] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *International journal of computer vision*, vol. 128, no. 2, pp. 261–318, 2020.

[10] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman, "Multiple kernels for object detection," in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 606–613.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 1097–1105.

[12] R. Yamashita, M. Nishio, R. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, vol. 9, 06 2018.

[13] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[15] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[16] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *Lecture Notes in Computer Science*, p. 21–37, 2016.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[18] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[19] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[21] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[22] J. Uijlings, K. Sande, T. Gevers, and A. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, pp. 154–171, 09 2013.

[23] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *Lecture Notes in Computer Science*, p. 346–361, 2014.

[25] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[26] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," *Advances in neural information processing systems*, vol. 29, 2016.

[27] L. Tychsen-Smith and L. Petersson, "Denet: Scalable real-time object detection with directed sparse sampling," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 428–436.

[28] H. Law and J. Deng, "Cornernet: Detecting objects as paired keypoints," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 734–750.

[29] C. Zhu, Y. He, and M. Savvides, "Feature selective anchor-free module for single-shot object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 840–849.

[30] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "Centernet: Keypoint triplets for object detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 6569–6578.

[31] L. Shao, F. Zhu, and X. Li, "Transfer learning for visual categorization: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 5, pp. 1019–1034, 2015.

[32] J. Zhang, W. Li, P. Ogunbona, and D. Xu, "Recent advances in transfer learning for cross-dataset visual recognition: A problem-oriented perspective," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.

[33] H. Venkateswara, S. Chakraborty, and S. Panchanathan, "Deep-learning systems for domain adaptation in computer vision: Learning transferable feature representations," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 117–129, 2017.

[34] B. Iancu, V. Soloviev, L. Zelioli, and J. Lilius, "Aboships-an inshore and offshore maritime vessel detection dataset with precise annotations," *Remote Sensing*, vol. 13, no. 5, p. 988, 2021.

[35] D. K. Prasad, "Singapore maritime dataset." [Online]. Available: https://sites.google.com/site/dilipprasad/home/singapore-maritime-dataset?authuser=0

[36] F. Yu, D. Wang, E. Shelhamer, and T. Darrell, "Deep layer aggregation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2403–2412.

[37] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *European conference on computer vision*. Springer, 2016, pp. 483–499.

[38] R. Padilla, S. L. Netto, and E. A. Da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 international conference on systems, signals and image processing (IWSSIP)*. IEEE, 2020, pp. 237–242.

[39] C. G. Northcutt, A. Athalye, and J. Mueller, "Pervasive label errors in test sets destabilize machine learning benchmarks," *arXiv preprint arXiv:2103.14749*, 2021.