TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

EMIL KATTAINEN

OBJECT DETECTION FOR CONTAINER CORNER DETECTION

Bachelor's thesis

Examiner: Professor Heikki Huttunen

ABSTRACT

EMIL KATTAINEN: Object detection for container corner detection
Tampere University of Technology
Bachelor of Science Thesis, 18 pages
December 2018
Bachelor's Degree Programme in Science and Engineering
Major: Machine Learning
Examiner: Professor Heikki Huttunen

Keywords: thesis, object detection

In this bachelor's thesis different deep neural networks for object detection are studied. The networks are used to detect corners of containers in harbor. The studied network types are Single Shot Detector, Faster Region based CNN and RetinaNet with MobileNet and ResNet as base networks. The dataset used is built as part of the study and the networks are implemented using the Tensorflow Object Detection API. The detection performance of the models are compared using average precision without K-fold validation on a single test set.

From the results we can see that the that using a pretrained network reasonable detection performance can be achieved using SSD or Faster R-CNN. RetinaNet however performs noticeably worse than the other methods.

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

BN	Batch Normalization
CNN	Convolutional Neural Network
FP	False Positive
FPN	Feature Pyramid Network
IoU	Intersection over Union
mAP	Mean Average Precision
R-CNN	Region based Convolutional Neural Network
ReLU	Rectified Linear Unit
RPN	Region Proposal Network
GPU	Graphics Processing Unit
SSD	Single Shot Multibox Detector
TN	True Negative
TP	True Positive

B_p	Predicted bounding box
B_{gt}	Ground truth bounding box
$\mathbf{x}$	Input vector
$\mathbf{y}$	Output vector
$\mathbf{W}$	Weight matrix
b	Bias value

1. INTRODUCTION

Machine learning has changed the world with the rapid development of deep learning. Deep learning based methods have been used to achieve state of the art performance in image recognition tasks such as classification and object detection. The previously used methods relied often on handcrafted feature extractors and couldn't be transferred to other problems trivially because of that. Deep convolutional neural networks learn the feature extractors from the data so they only require the data to be specific for the task. [4]

Anomaly detection is an important subtopic of machine learning. In anomaly detection the problem is to find patterns which don't conform to expected behavior. These patterns are called outliers. For example anomaly detection is used when detecting frauds from credit card transactions and when detecting cancer from MRI scans.

Harbors have thousands of containers passing through every day. Finding out that the containers are damaged before sending them overseas would be useful so the containers wouldn't get compromised during the trip. Having a human physically inspect the containers isn't feasible because it would hammer the throughput of containers harbors would have. Not checking the containers leads to people noticing the damage in the containers at arrival. This creates problems when we have to find out which part of the transportation pipeline is responsible for the damage because of insurance reasons. Having an automated system to find out about the damages ahead of time would save us from all that trouble.

To find out whether a container is damaged we can compare it to a reference image of a similar container which for sure isn't broken. To compare the pictures they need to be lined up. The pictures which are taken of the containers at harbor aren't exactly lined up so we need to line them up somehow. For this we detect the corners of the container and use an affine transformation to get the picture to match the reference picture. The main point of this study is to try out different methods for finding the corners and measure the performance of those methods with some specified metrics in finding the corners. There are older methods which have been used to detect the corner previously. These older methods use handcrafted features to find the corners. In this study we use object detectors based on convolutional neural networks to detect position of the corners.

In the following chapter the theory behind the used methods is described. In the third chapter the implementation of these methods is shown. The fourth chapter presents and discusses the results and the final chapter contains conclusions drawn from the results and some suggestions for future research.

2. THEORY

This chapter specifies the theoretical concepts that are used in this study. We define what neural networks are and how they work. We show how detecting objects is done and how the quality of detections is assessed.

2.1 Machine Learning

Machine learning is data analysis method for automatically building analytical models. A subfield of machine learning is supervised learning. In supervised learning algorithms learn from data given to them to increase their performance on a specific task. The data consists of examples which are pairs of a data sample and an annotation. The task for a supervised learning algorithm is to make predictions of the data which corresponds as closely to the given annotation in dataset. The method should also be able to make predictions for data that hasn't been used to tune the method and not only perform well on the data used for tuning. If performance with the data used for tuning is a lot better than with data which isn't used for tuning the method is considered overfitted. [4]

To tell how well an algorithm works task specific metrics are used. The dataset is often split into train, validation and test sets. The training set is used to train the model while the validation set is used to track the performance during training to know when to stop. After training the test set is used to test the final performance of the models. To more accurately compare different models often k-fold validation is used. In k-fold cross validation the data is split into k folds. The models are then trained k time by using one of the folds as validation set and the others as the train set changing which set is used as the validation set after each run. The metrics on different validation sets are then averaged so the variance of the results is reduced and the reliability of the metric is higher. K-fold cross validation takes a lot of time so it isn't necessarily always used for neural networks. [4]

2.2 Neural networks

Neural network is a machine learning algorithm inspired by the human brain. The network uses thousands to millions of artificial neurons to make predictions based the data by using a stochastic gradient descent like loss minimization algorithm. Each neuron has multiple inputs and a single output. Neurons calculate their output by multiplying the inputs with their corresponding weights, summing the products and a bias value together and using an activation function on the sum. [4]

Neural networks are built from different kind of layers. The layer types used in this study are convolutional, fully connected, batch normalization and depthwise separable convolution.

Different neural network architectures are defined by how these different layers are used. The architectures used in this study are deep residual networks and MobileNets.

Activations

Activation functions act as nonlinearities between the layers. This makes networks capable of approximating more complex functions. Typical activation functions are rectified linear unit (ReLU)

$$y = max(x, 0) \tag{2.1}$$

or different sigmoid functions like the logistic function

$$y = \frac{1}{1 + e^{-x}}. \tag{2.2}$$

Sigmoids were used before in shallow networks but ReLUs and variants of ReLUs are used in deeper networks because of their properties. The main advantages of using ReLUs in deep neural networks is the fact that the gradient won't vanish like they do with sigmoids and the computational speed of ReLUs being faster than sigmoids. [1]

Convolutional layers

Convolutional filters are often used in image processing. They work by having a sliding window go over an image and calculating weighted sum over all the values in the sliding window for each single pixel. Convolutional layers use convolutional filters with learnable weights to produce the output of the layer. Each convolutional layer has multiple filters with same size to produce multiple feature maps. Formally the output is calculated as

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n), \tag{2.3}$$

where I is 2d input image, K is a 2d kernel with the weights.

Convolutional layers are often used as the first layers in image recognition tasks to extract features from the image. Traditional methods often used handcrafted kernels to get the wanted features from the pictures as a stage in image recognition pipelines. [5]

Fully connected layers

In fully connected layers all the neurons take all the outputs from the previous layer as inputs. These are often used as the few final layers of the network to reduce the number of outputs of the whole network. Fully connected networks consist of only this kind of layers stacked on top of each other. Figure 2.1 shows a single neuron with a single row of weights multiplied with the input vector and the bias. A fully connected layer is built from many of these. The output for full layer is calculated with

$$\mathbf{y} = h(\mathbf{W} \cdot \mathbf{x} + \mathbf{b}), \tag{2.4}$$

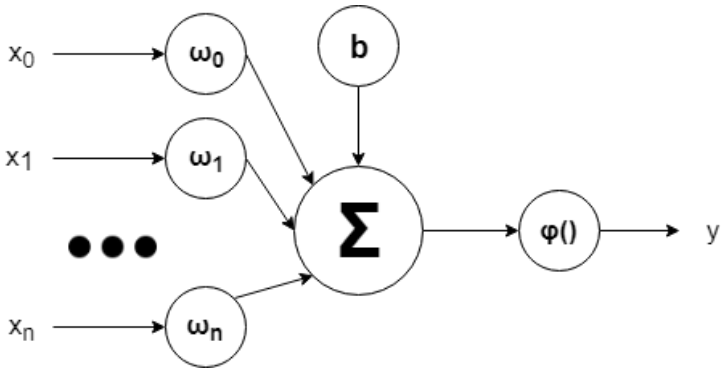

Figure 2.1. *Image of an artificial neuron. X0-xn are the input vector which are multiplied by the corresponding ω. These are summed together with bias b at the Σ. Final output of the neuron is value of the activation function at the sum.*

where h is the activation function, **W** is the weight matrix with size of **x** as width and size of **y** as height, **x** is the input vector, **b** is the bias vector and **y** is the output vector. [4]

2.3 Neural network architectures

Network architectures define how different layers are combined. Different kind of network architectures are better for different use cases. We are going to talk about two popular kinds of architectures ResNets and MobileNets. ResNets are good when the computational time isn't as big of a concern and maximum performance in metrics is preferred. On the other hand MobileNets are useful when evaluating the model has to be fast and performance in metrics doesn't have to be as high as possible.

Deep Residual Networks

Deep Residual Networks are an modification to traditional convolutional neural networks to improve their trainability at higher depths. ResNets use multiple residual blocks to build a deep network. Each residual block has two convolutional layers and a skip connection. The output of the residual block is sum of the output of the convolutions and the input as shown in 2.2. [6]

The skip connection changes the layers so that they don't have to calculate the y value directly but have to calculate only the difference between x and y as function of x. This makes the networks easier to train and makes very deep networks possible to train. The state of the art deep neural network previous to ResNets was VGG-19 which used normal convolutional layers and was 19 layers deep. With residual connections ResNet was scaled to 152 layers before the performance gains started to diminish. Other depths used for ResNets are 34, 50 and 101. [6]

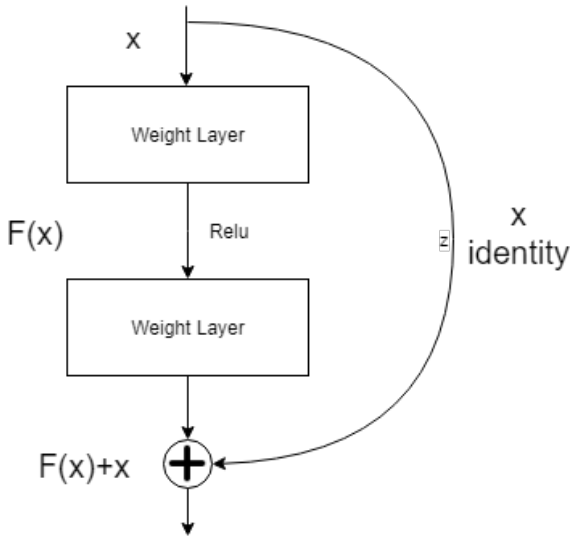

Figure 2.2. *A residual block showing the convolutional layers as F(x) and the skip connection as x beside it.*

MobileNet

MobileNets are designed to be small and low latency to meet the needs of mobile platforms. MobileNets use depthwise separable convolutions to achieve this. Depthwise separable layers use depthwise convolutions and pointwise convolutions. Normal convolutional layers do these in one step but splitting it to two stages reduces computation and model size drastically. Depthwise convolution applies convolution filter to each filter separately. Pointwise convolution is equivalent to a 1x1 normal convolution which then combines the output of the different filters. The combination of these layers are shown in 2.3 with comparison to normal convolutional layer. These blocks are then used to build the MobileNet. The MobileNet consists of one normal convolutional layer as the first layer and 13 of these depthwise separable convolution blocks stacked on top each other. The original MobileNet was used as a classifier so there was also a single fully connected layer on top of the convolutions.

Computational cost of normal convolution is

$$D_K * D_K * M * N * D_F * D_F \tag{2.5}$$

where M is the number of input channels, N is the number of output channels, D_K is width and height of the kernel and D_F is the width and height of the feature map. In comparison the cost of depthwise separable convolution is

$$D_K * D_K * M * D_F * D_F + M * N * D_F * D_F. \tag{2.6}$$

Therefore the reduction in computation is

$$\frac{D_K * D_K * M * D_F * D_F + M * N * D_F * D_F}{D_K * D_K * M * N * D_F * D_F} = \frac{1}{N} + \frac{1}{D_K^2}. \tag{2.7}$$

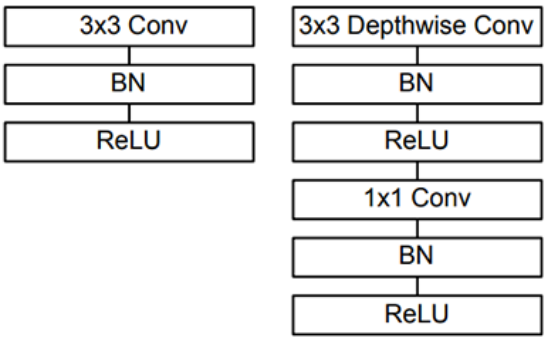

Figure 2.3. *Comparison of how layers are stacked in regular CNNs and MobileNets. Instead of a regular convolutional layer MobileNet uses 3x3 depthwise separable and 1x1 pointwise convolution. [7]*

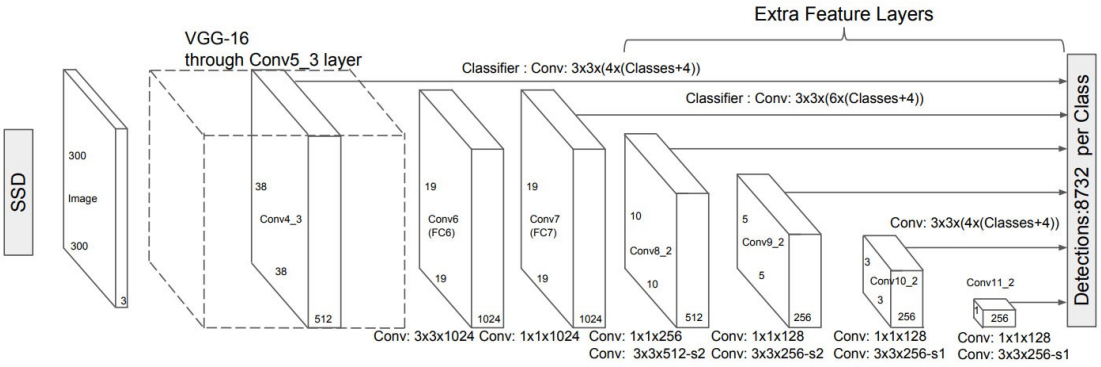

Figure 2.4. *Image showing Single Shot MultiBox Detector with VGG-16 as basenet. The image shows how there are additional convolutional layers as classifiers to make predictions based on different scaled features.[13]*

The width and the height of the kernels in MobileNet are 3 so the depthwise separable convolutions in MobileNet use 8 to 9 times less computation than normal convolutions. [7]

2.4 Object detection

In object detection the challenge for the algorithm is to predict bounding boxes for an image. Each predicted bounding box (B_p) has a location (x, y), 2d size (w, h), confidence (c) and class (cls). In the dataset there are ground truth bounding boxes (B_{gt}) which have the same properties as predicted bounding boxes besides not having a confidence associated with them. The predictions and ground truths can be compared to assess performance of the object detection algorithm. [3]

Single Shot Detector

Single Shot MultiBox Detector (SSD) uses a base network with added convolutional layers to generate predictions for a frame. It is called ”single shot” because of the fact that it is

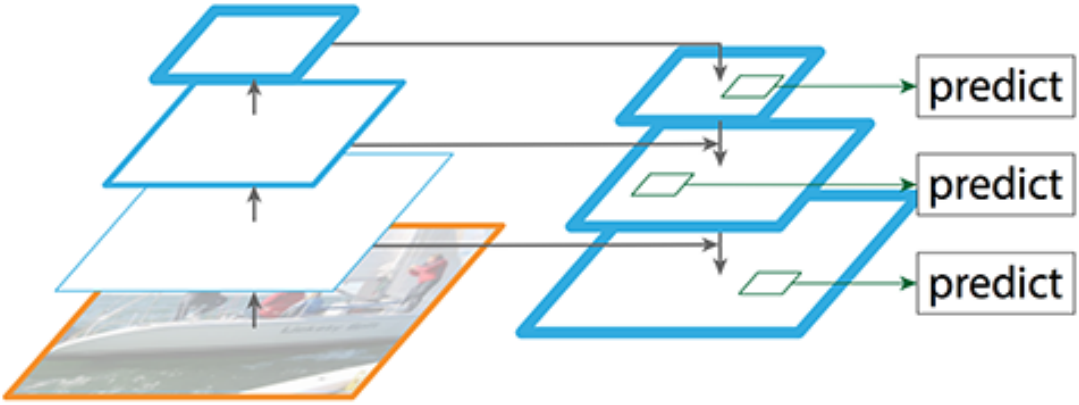

Figure 2.5. *Image of feature pyramid. Shows how the semantic information from the top layers is combined with the higher resolution features from lower layers. [10]*

evaluated only once when predicting the bounding boxes of a single image. The added convolutional layers use feature maps from intermediate layers as input as seen on figure 2.4. These added layers output box coordinate offsets and class probabilities for each pixel in the feature map to make the predictions. Non-maximum suppression is used on the output of the added convolutional layers to make the final predictions. [13]

Faster Region based CNN

Faster R-CNN consists two modules instead of a single one used in SSD. The first one is a fully convolutional network called region proposal network (RPN) which outputs possible regions for objects to be in. The second one is the Fast R-CNN detector which uses the proposed regions to tell whether there actually is an object and which class it is. Both of the networks share their first few layers as a base feature extractor. The region proposals consist of a rectangular object proposals with an objectness score. The region proposals are generated by sliding a small network over the output of the last shared layer. This network has a box regression output and a box-classification output. The box regression output encodes coordinates for k bounding boxes and the box-classification output estimates probability of an object being there. The k bounding boxes each have their own aspect ratio-scale pair. The Fast R-CNN model then predicts classes for the boxes to get the final output. [14]

RetinaNet

RetinaNet combines a novel loss function with a feature pyramid network (FPN) to increase performance for the Single Shot Detector. RetinaNet's loss function takes into account the low amount of positive samples and weighs those more than the easy negative outputs. The FPN is an augmentation to a standard fully connected network to get multi-scale feature pyramid from a single scale input image.

The loss in RetinaNet is called focal loss. The loss function is

$$FL(p_t) = -(1 - p_t)^{\gamma} log(p_t) \tag{2.8}$$

where γ is hyper parameter for adjusting the effect of focal loss and

$$p_t = \begin{cases} p & if\, y = 1 \\ 1 - p & otherwise \end{cases} \tag{2.9}$$

where p is the model's probability given for class $y = 1$. The addition of the term in front of the log loss keeps the loss functioning similarly to normal cross entropy loss when an example is misclassified and the model is giving big probability for $y = 0$ or small probability for $y = 1$. This helps the model to learn from the rarer cases of $y = 1$. [11]

The feature pyramid network helps the model detect objects at multiple scales. It is constructed by having a top-down pathway in addition to the base network. The top down path combines the semantic knowledge of the input with the feature maps at lower levels with higher resolutions. This way instead of making predictions based on only some of the middle layers of the network the predictions can be made based on all the layers of the network and at different resolutions. [10]

2.5 Metrics

Metrics are used to measure how well different algorithms work to solve the problem. To assess how well the algorithm works for unseen data the dataset is split into training and validation datasets. The training dataset is used for training and the metrics calculated on this set don't necessarily reflect how well the algorithm will do on unseen data. The validation set images are used only to calculate metrics and performance on this data is more likely to be what one could expect for other unseen data.

When calculating metrics for detection algorithms detections are often classified as true positive (TP), false positive (FP), true negative (TN) and false negative (FN). The metrics are then calculated using the amount of different kind of detections. The detections are classified by comparing models predictions to the ground truths in the dataset. If the model predicts a bounding box on a ground truth so that the intersection over union (IoU) is over the threshold value, the prediction is considered a true positive. In Pascal VOC challenge the threshold for true positive is 0.5 IoU [2]. In COCO used metrics are calculated using thresholds from 0.5 to 0.95 with 0.05 intervals in between the thresholds [12]. IoU is calculated as intersection of the detection and the ground truth divided by the union of the two as in

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_t \cup B_{gt})}. \tag{2.10}$$

If the prediction isn't matched with any ground truth bounding box, the prediction is considered a false positive. If a ground truth bounding box isn't matched with any prediction

the bounding box is considered a false negative. From these counts we can calculate multiple different metrics. The ones which are most relevant in this task are precision, recall and mean average precision. Precision measures what percentage of all the detections are true positives as shown in 2.11.

$$precision = \frac{TP}{TP + FP} \tag{2.11}$$

Recall measures what percentage of all the objects are detected as shown in 2.12.

$$recall = \frac{TP}{TP + FN} \tag{2.12}$$

These two measures are used to calculate mean Average Precision (mAP) which is often used to compare different object detection algorithms. To calculate mAP first the precision-recall-curve should be calculated. The curve is calculated by calculating precision and recall values at different confidence thresholds. AP is the area under this curve. The area can be approximated in a few different ways. In Pascal VOC challenge the area is approximated by interpolating the precision value at 11 equally spaced recall levels from the calculated precision recall pairs. Mean AP is mean of AP for all classes. [2]

3. IMPLEMENTATION

In this chapter it is explained how the previously mentioned algorithms were implemented. The Tensorflow Object Detection API is talked here [8]. Also the way the dataset for training and evaluation was built is described.

3.1 Dataset

Visy Oy is a Finnish company providing systems for automated access and area control used most notably in different harbors and terminals around the world. They provided the images from their harbor systems. The images were scanned from the trucks carrying containers at entering the harbor. There were artificial lights setup so that the conditions would be good at all times but there still were sometimes artifacts in the images which makes the corners hard to recognize even for human labeler. There were 512 images altogether.

The images were annotated using an open source tool in Dlib library called Imglab [9]. Imglab offers functionality for drawing bounding boxes in the images with a class and changing image. Figure 3.1 shows the Imglab GUI. Imglab saves the drawn bounding boxes in an xml file. [9] The xml file and folder containing the labeled images can then be parsed into tfrecord and pbtxt format for data and labels respectively. These are the data formats required by Tensorflow object detection API. The pbtxt format is file format used to save ProtoBuf objects in text format. Tfrecord format is used in Tensorflow for storing data. [15]

3.2 Training

Training the models was done with Tensorflow object detection API. [1] The object detection API offers an easy way to build and train object detection models on top of Tensorflow. In their Github repository they also offer pretrained models to use for finetuning for a specific task. These models have been pretrained on COCO object detection dataset [12]. This dataset has 80 different object classes ranging from ties to animals. We tried the models shown in table 3.1. The inference times are provided by Tensorflow detections API and are measured on a machine with an Nvidia GeForce GTX TITAN X. [15]

Data augmentation

Data augmentation is a method used in deep learning to generate more data. Images are often augmented by using some kind of affine transformations on them. Augmentation often

[1] $https://github.com/tensorflow/models/tree/master/research/object\_detection$

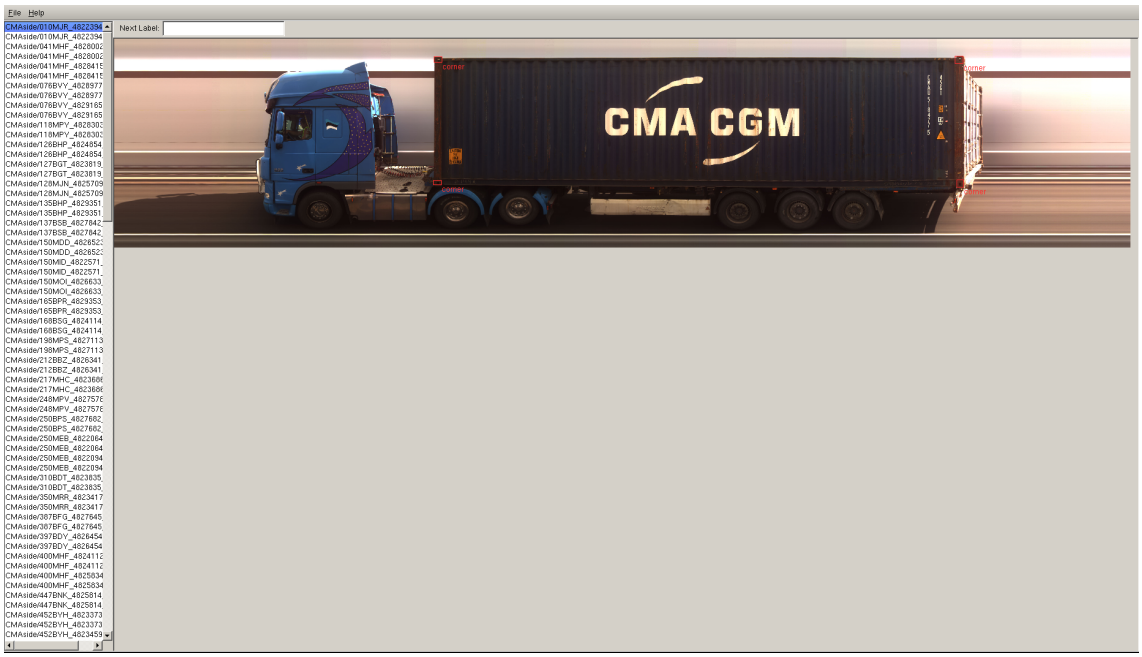

Figure 3.1. *Image of the annotating tool Imglab from Dlib. On the left it shows the list of images in the dataset and in the middle the image to be annotated. The small red boxes in the image are the annotated ground truth bounding boxes.*

Model	mAP	Speed(ms)
ssd_mobilenet_v1	21	30
ssd_mobilenet_v1_fpn	32	56
ssd_resnet_50_fpn	35	76
faster_rcnn_resnet_50_fpn	30	89

Table 3.1. *Results for pretraining on COCO object detection dataset in mAP and inference time for different models on an Nvidia GeForce GTX TITAN X.*

helps increase the performance of deep learning models. Tensorflow Object Detection API provides online data augmentation for training. During training only random horizontal flipping of the images was used. [16]

3.3 Environment

The platform used for training was Tampere University of Techology's own cluster. It contains many nodes with multiple Nvidia GPUs and is used mainly by people at TuT Signal Processing group. The allocated hardware for a single training instance on a node was NVidia Tesla P100 or K80 for GPU and a single core of the processor on the node. The following list shows the most important software and the versions used for building the dataset, training and evaluation.

- Python 3.6
- Tensorflow 1.11

- Open CV 3.4
- CUDA 9.0
- cuDNN 7.3.1
- Tensorflow Object Detection API library
- Imglab annotation tool from Dlib 19.16

4. EVALUATIONS

In this chapter the results are discussed. Different models are compared based on the mean average precision scores. Examples of the detections on images are shown. Different kind of cases and performance in these cases are studied in case study style.

4.1 Results

The results were a little different from what could've been expected. Besides the best model the mAP correlated very positively with FPS. This is very counterintuitive because one would think that the more the model takes time to calculate its output the better the results. This anomaly was possibly due to the loss function of RetinaNets not performing as well on this data set or due to the feature pyramids being unnecessarily large for this task. Also for anchor boxes the default sizes from Tensorflow object detection API were used. Optimizing the anchor-boxes would probably increase the performance significantly.

From 4.1 we can see that the precision drops almost instantly from one to zero after certain recall value specific to the model. This means that most of the objects are either easy for the model or really hard for the model and there aren't too many of objects which are in the middle.

Model	mAP	FPS
ssd_mobilenet_v1	87	8.7
ssd_mobilenet_v1_fpn	73	6.1
ssd_resnet_50_fpn	68	3.9
faster_rcnn_resnet_50	97	3.0

Table 4.1. *Results in mAP and FPS for different models.*

4.2 Examples

The images could be split into 3 or 4 categories. These categories are easy, medium, hard and very hard. Hard and very hard could be combined together but as these categories are very anecdotal it isn't exact science.

The first example in figure 4.2 is portraying a typical easy case. The lighting is good and there is only one fully visible container. The models mostly successfully detected the containers in these kind of images. The second example in figure 4.3 is slightly harder one and even the faster rcnn with resnet failed with this kind of images sometimes. This type is defined by slightly odd lighting with one or two containers fully visible on the screen.

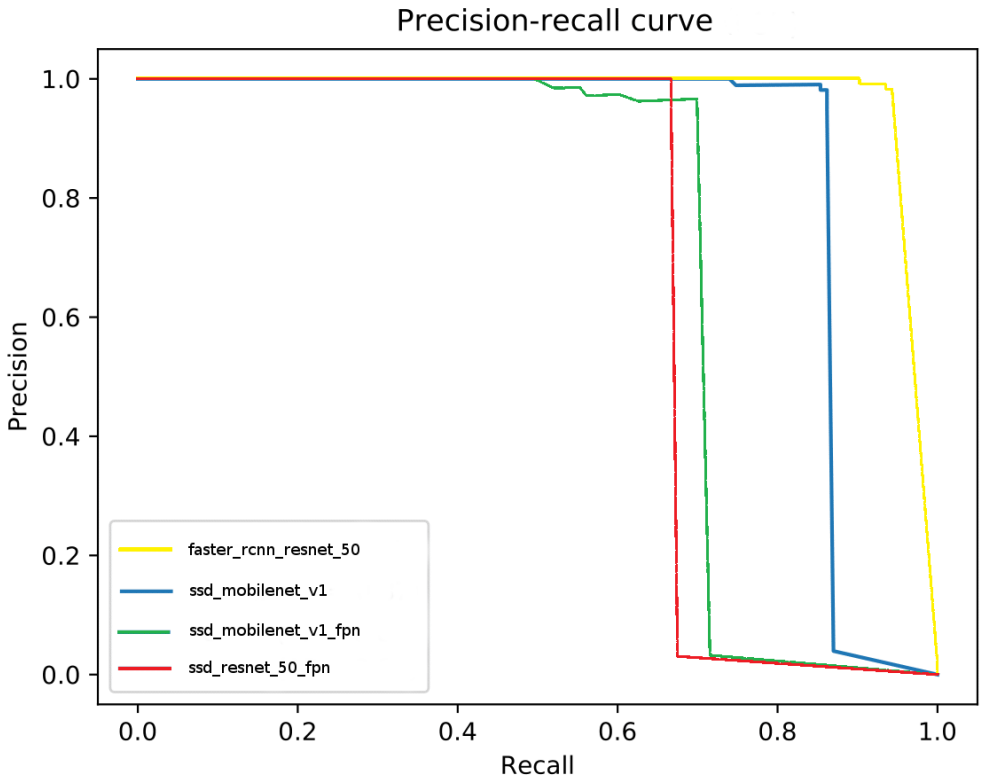

Figure 4.1. *Figure of the precision recall curves of different models. As can be seen for all models the precision stays at almost 1 while recall is incremented before the sudden drop to near zero precision happens.*

The third example in figure 4.4 is a typical hard case. There are two containers and the lighting is very different from normal cases. Models had some problems with differentiating the middle corners and sometimes just predicted there to be one corner in the middle. The fourth example 4.5 is portraying possible case of a really hard image. Even the best model only found one of the corners in this particular image. This image has two containers, the containers have a bit of an angle and the lighting is very bad. Models could probably still learn these harder cases as well but that kind of images were most likely too rare in the small dataset so the models didn't learn them.

Figure 4.2. *A typical easy case. One container and good lighting. Predicted corners are by SSD MobileNet with RetinaNet.*

Figure 4.3. *A typical case of middling difficulty. One container with not quite perfect lighting. Predicted corners are by SSD Mobilenet.*

Figure 4.4. *A typical case of harder cases. Two containers with odd lighting. Predicted corners are by faster R-CNN ResNet50.*

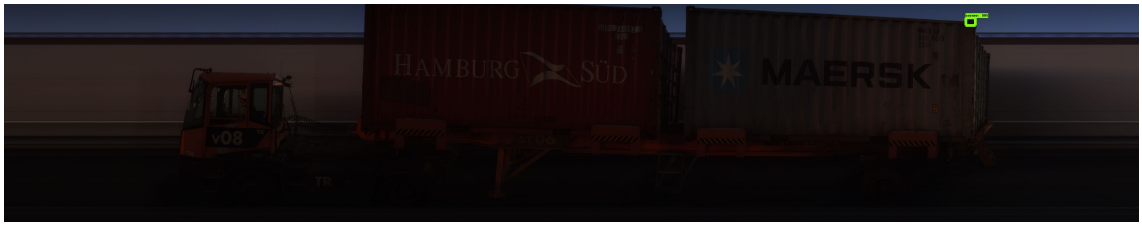

Figure 4.5. *A really hard case. Two containers with some angle so one corner is out of picture and lighting is really bad. Predicted corners are by SSD ResNet50 with RetinaNet.*

5. CONCLUSION

In this thesis we tried out some object detection models with our dataset. Some of the models tested worked reasonably well on the problem and probably with a little bigger dataset a real world application could be built based on this. The object detection models used were faster R-CNN, SSD and RetinaNet with MobileNet and ResNet as the base feature extractors. We found out that faster R-CNN performed the best and also the slowest. Models using RetinaNet performed worst but the reason for this wasn't obvious. SSD was the fastest model and didn't perform too much worse than faster R-CNN so in a case where speed is more important than having the most accurate detections SSD is a viable choice. When detecting corners of containers in harbors the speed isn't that important because there isn't need to infer boxes on multiple images a second so using faster R-CNN would be the best in our case. We can approximate that with the 0.97 mAP with faster R-CNN all the four corners would be predicted right around 90% of the time. This means the alignment wouldn't be correct around 10% of the time. This level of performance would already be helpful in a final product but better would be desirable.

Future Work

This thesis was about testing a few different models for the first stage of a pipeline for detecting damaged containers from images. The next obvious step would be to implement the next steps of this pipeline and use these models with the following steps to test how the whole pipeline would perform.

The used dataset was rather small. Building a bigger dataset and training and testing on it would be likely to increase the performance of the models. Other options could be to try out different anchor-boxes or even some other model architectures. To asses the performance more accurately it would be useful to use k-fold validation if bigger dataset or more models were to be tried out.

REFERENCES

[1] G. Alcantara, Empirical analysis of non-linear activation functions for deep neural networks in classification tasks, 30 Oct, 2017. Available: https://arxiv.org/pdf/1710.11272.pdf

[2] M. Everingham, L.V. Gool, C.K.I. Williams, J. Winn, A. Zisserman, The pascal visual object classes (voc) challenge, 30 Jul, 2008. Available: http://host.robots.ox.ac.uk/pascal/VOC/pubs/everingham10.pdf

[3] A. Godil, R. Bostelman, W. Shackleford, T. Hong, M. Shneier, Performance metrics for evaluating object and human detection and tracking systems, July, 2014. Available: http://dx.doi.org/10.6028/NIST.IR.7972

[4] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016. http://www.deeplearningbook.org.

[5] T. Guo, J. Dong, H. Li, Y. Gao, Simple convolutional neural network on image classification, pp. 721–724.

[6] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, 10 Dec, 2015. Available: https://arxiv.org/abs/1512.03385

[7] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, 17 Apr, 2017. Available: https://arxiv.org/abs/1704.04861

[8] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, K. Murphy, Speed/accuracy trade-offs for modern convolutional object detectors, CoRR, Vol. abs/1611.10012, 2016.

[9] D.E. King, Dlib c++ library. Available: http://dlib.net

[10] T.Y. Lin, P. Dollar´, R. Girshick, K. He, B. Hariharan, S. Belongie, Feature pyramid networks for object detection, 19 Apr, 2017. Available: https://arxiv.org/pdf/1612.03144.pdf

[11] T.Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollar, Focal loss for dense object detection, 7 Feb, 2018. Available: https://arxiv.org/pdf/1708.02002.pdf

[12] T.Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C.L. Zitnick, P. Dollár, Microsoft coco: Common objects in context. Available: https://arxiv.org/abs/1405.0312

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.Y. Fu, A.C. Berg, Ssd: Single shot detector, 8 Dec, 2015. Available: https://arxiv.org/abs/1512.02325

[14] S. Ren, K. He, R. Girshick, J. Sun, Faster r-cnn: Towards real-time object detection with region proposal networks, 4 Jun, 2015.

[15] Tensorflow. Available: https://www.tensorflow.org

[16] S.C. Wong, A. Gatt, V. Stamatescu, M.D. McDonnell, Understanding data augmentation for classification: when to warp?, 26 Nov, 2016. Available: https://arxiv.org/pdf/1609.08764.pdf