Bachelor's thesis

Information and Communications Technology

2018

Jussi Jokela

PERSON COUNTER USING REAL-TIME OBJECT DETECTION AND A SMALL NEURAL NETWORK

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2018 | 38 pages

Jussi Jokela

PERSON COUNTER USING REAL-TIME OBJECT DETECTION AND A SMALL NEURAL NETWORK

Machine learning is a trend in artificial intelligence, and deep learning is one of the fields in machine learning. In short, machine learning means that the computer learns to accomplish given complex tasks independently, while developing itself at the same time. Deep learning is inspired by how human brain works, which makes it more complex but, at the same time, more efficient learning algorithm.

The main goal of this thesis was to research deep learning and its different frameworks, and after that create a person counter which utilizes deep learning and neural networks for object detection. Also, the requirements were that the counter was supposed to be run on affordable hardware.

During the coding process, many different deep learning frameworks were tested and reviewed. Also, the training of own model and object tracking were tested. During testing, the only available hardware was Intel NUC. The NUC was able to run the test code with tracking enabled at around 4 – 6 frames per second and without tracking at around 6 – 8 frames per second. These speeds are barely adequate for real-time object detection or tracking, so some more optimization or slightly better hardware would be required.

The end result of the project was a functional deep learning pedestrian counter, which would need further optimization for improved counting accuracy and speed.

KEYWORDS:

object detection, object tracking, computer vision, machine learning, deep learning, neural network, Tensorflow

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2018 | 38 sivua

Jussi Jokela

HENKILÖLASKURI HYÖDYNTÄEN OBJEKTIN TUNNISTUSTA JA SUPPEAA NEUROVERKKOA

Koneoppiminen on yksi tekoälyn suuntauksista, ja syväoppiminen on yksi koneoppimisen osa-alueista. Koneoppiminen tarkoittaa, että kone oppii itsenäisesti suorittamaan monimutkaiset tehtävät ja kehittämään itseänsä koko ajan. Syväoppiminen taas pohjautuu lähinnä samaan tapaan, miten ihminen oppii asioita, ja sen takia se onkin paljon monimutkaisempi, mutta tehokkaampi oppimistapa.

Tämän opinnäytetyön tarkoituksena oli tutustua syväoppimiseen ja sen eri kehitysympäristöihin, ja sen jälkeen luoda henkilölaskuri, joka hyödyntäisi syväoppimista ja neuroverkkoja henkilön tunnistukseen. Työn vaatimuksiin kuului myös, että laskuri toimisi mahdollisimman halvalla tietokoneella.

Ohjelmoinnin aikana kokeiltiin monia eri syväoppimisen kehitysympäristöjä, oman tunnistusmallin opettamista ja objektin seurantaa. Testauksen aikana ainoa saatavilla oleva testikone Intel NUC pystyi suorittamaan testiohjelmaa seuranta päällä noin 4 – 6 ruutua sekunnissa, kun taas ilman seurantaa noin 6 – 8 ruutua sekunnissa. Nämä nopeudet eivät juurikaan riitä reaaliaikaiseen laskentaan, joten ohjelma vaatisi lisää optimointia tai sitten hieman tehokkaamman tietokoneen.

Työn lopputuloksena oli toimiva ohjelma, joka hyödyntää syväoppimista ja neuroverkkoja henkilölaskentaan. Optimoinnilla voitaisiin saada parempi ruudunpäivitys ja laskentatarkkuus.

ASIASANAT:

objektin tunnistus, objektin seuranta, konenäkö, koneoppiminen, syväoppiminen, neuroverkot, TensorFlow

CONTENTS

FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Explanation of abbreviation
API	Application Programming Interface
Caffe	Convolutional Architecture for Fast Feature Embedding
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CV	Computer Vision
GPU	Graphics Processing Unit
IoT	Internet of Things
IOU	Intersection Over Union
mAP	mean Average Precision
ML	Machine Learning
NDA	Non-disclosure agreement
NMS	Non-Maximum Suppression
NUC	Next Unit of Computing
R-CNN	Region-based Convolutional Neural Network
RPN	Region Proposal Network
SORT	Simple Online and Realtime Tracking
SSD	Single Shot MultiBox Detector
YOLO	You Only Look Once

1 INTRODUCTION

These days, there are video surveillance systems everywhere. Monitoring technologies are common in everyday life but they are also used for military and other purposes.

The goal of this thesis is to examine different algorithms for object detection using neural networks and pick the most suitable one for pedestrian counting on affordable hardware, such as Intel NUCs or NVIDIA Jetsons, which both cost roughly from 400 to 600 euros. These requirements cause some limitations on the detection model because the most accurate models require lots of computing power.

There are several different methods for object detection using computer vision, and some methods are more reliable and robust than others. The most modern method is to use deep learning. In deep learning, a computer learns to perform classification tasks directly from examples and can achieve top-quality accuracy [1].

Deep learning is part of machine learning family, and machine learning is one of the fastest-growing and most exciting fields in artificial intelligence. Deep learning has been around since the 1980's, but has become useful only recently because it requires a great amount of labeled data and computing power [1].

Deep learning architectures have been applied to multiple fields including computer vision, speech recognition and board games, where in some cases these solutions have produced results comparable to human experts, if not even superior.

Most of the references used in this thesis are website articles and blog posts, but all sources should be well-known and popular in the deep learning community.

This thesis is structured so that the first chapters (Chapter 2 – and 3) introduce the reader to the subject and explains what object detection is and how neural networks work. The following chapters (Chapter 4, 5) go through the most famous deep learning algorithms and the tools used in this project. The last chapter (Chapter 6) goes through the development in this project and explains briefly all the steps, However, because the project is built on top of Fideras own code and due to NDA, no important code is shown.

2 WHAT IS OBJECT DETECTION?

Computer vision, as the name suggests, is a field in computer science that works on giving computers the ability to see, identify and process images in the same way that human eyesight does. [2]

In computer vision, object detection means searching for an object in an image or a video. After detection, that object can be classified in multiple categories, such as human or a boat, for instance.

Video is just a sequence of images displayed in rapid succession, so it is obvious that all image processing techniques can be applied to it [3].

Object detection is one of the areas in computer vision that is evolving very rapidly. New algorithms keep outperforming the older ones in terms of speed and accuracy. Historically, object detection emerged in 2001 when Paul Viola and Michael Jones came up with the idea of Haar Cascades.

Haar Cascade is a classifier which is used to detect the object which it has been trained for. Haar Cascade classifier is trained using a set of positive and negative images, where positive images are images of the object and negatives are something else.

With the introduction of convolutional neural networks (CNNs) and their proven success in computer vision, cascade classifiers are now the second-best alternative [4].

Convolutional neural networks work by splitting the input into smaller chunks, and then passing that to the next layer which does the same thing with different rules.

Object detection and classification are simply preceding steps for object tracking. In object tracking, the goal is to keep track of its motion, location and occlusion. Object tracking is used in many different applications, such as video surveillance, robotics and traffic monitoring.

3 HOW NEURAL NETWORKS WORK?

Similar to how a child learns to recognise objects, an algorithm needs to be shown thousands of pictures before it learns to recognise objects and make predictions for images it has never seen before [5].

Computer handles images as numbers, and every image can be represented as a 2-dimensional matrix full of numbers, known as pixels [5]. The value of each pixel in the matrix ranges from 0 to 255, 0 being black and 255 being white. Figure 1 shows an example of a 2-dimensional matrix and image classification.

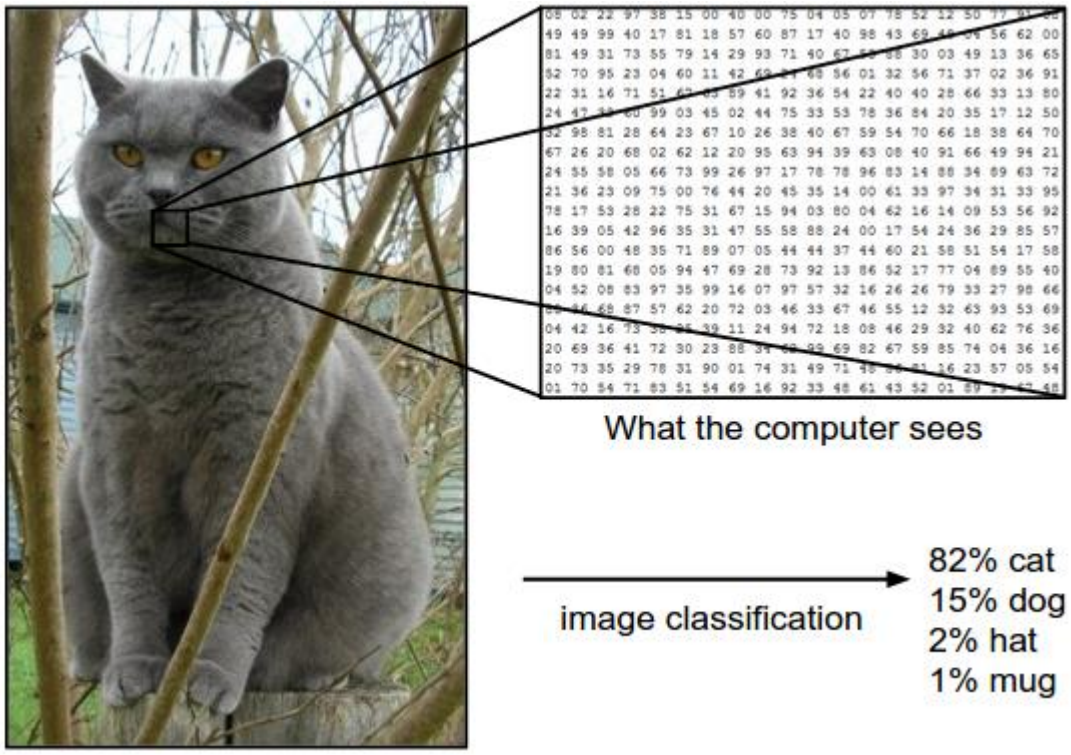

Figure 1. How a computer sees an image. — Source: http://cs231n.github.io/classification/

3.1 Artificial Neural Networks

Our brain uses an enormous connected network of neurons to process all information. A neuron is a cell that receives, processes, and transmits information through electrical and chemical signals.

Artificial Neural Networks is a machine learning technique which is based in our brain structure.

The neurons are arranged in layers: an input layer, one or more ”hidden” layers and an output layer, as seen in Figure 2. For the basic idea how neural networks work can be compared to a factory line. After the raw material (data) is passed from the input layer, each ”hidden” layer extracts different sets of high-level features. The first layer might analyze the color or brightness of the pixels. The next one might look for edges, based on lines of similar pixels. Third one might be looking for shapes or textures. [6]

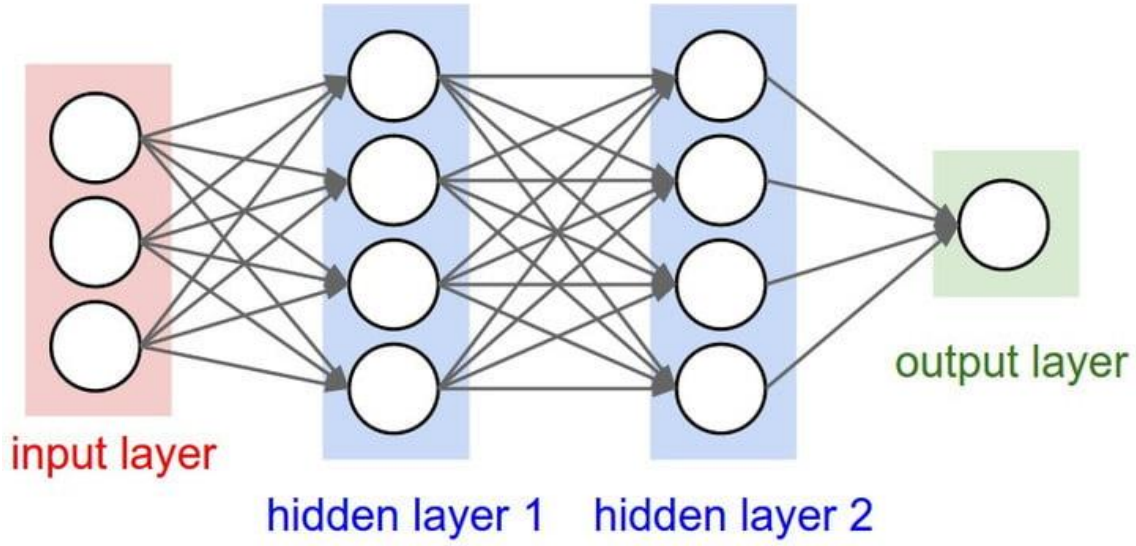

Figure 2. A neural network with two hidden layers. — Source: https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/

After a certain number of layers has been processed, the network will have created complex feature detectors. These feature detectors can figure out that certain features (like nose, eyes, mouth) are commonly found together. [6]

After the detection has been carried out, the results can be labeled and any errors that the detector made can be corrected by using backpropagation. After enough training and corrections, the network can work on its own without any human assistance. [6]

3.2 Convolutional Neural Networks

So how does convolutional neural networks differ from normal neural networks? They have different architecture. Normal networks transform an input through hidden layers.

CNNs layers are organised in 3 dimensions: width, height and depth, as displayed in one of the boxes in Figure 3. Also, the neurons in one layer do not connect to all neurons in the next layer, only a small part of it. Once everything is complete, the ouput will be reduced to a single vector of probability scores, organized along the depth dimension. [5]

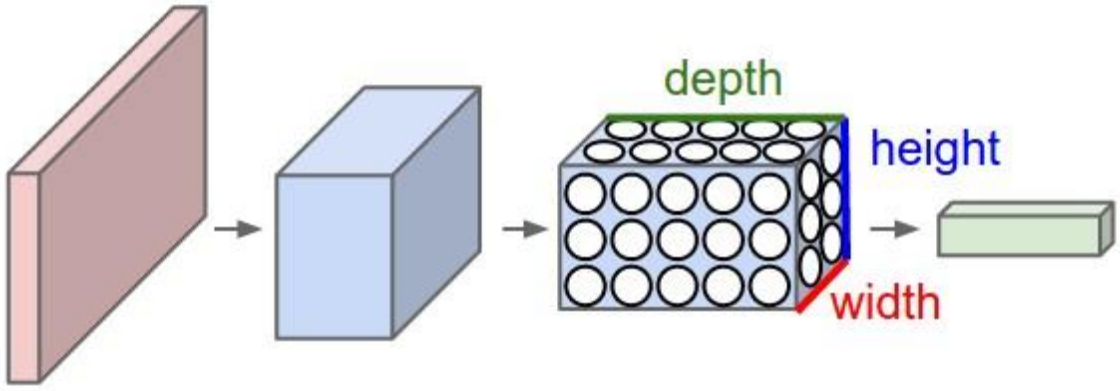

Figure 3. ”The red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels)”. — Source: http://cs231n.github.io/convolutional-networks/

3.2.1 Convolution

Convolution is one the main parts in CNN. Convolution is a mathematical term and refers to a combination of two functions to produce a third function, merging two sets of information [5].

As previously mentioned, every image can be considered as a matrix of pixel values. In reality, convolutions are performed in 3D, but for clarity this example operation is now performed in 2D. Think of a 5 × 5 matrix and a 3 × 3 matrix that both only have values 0 or 1, as in figure 4.

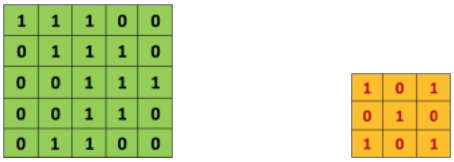

Figure 4. Convolution matrixes. — Source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Now, when we slide the 3 × 3 matrix over the 5 × 5 matrix, starting from the top-left corner, we acquire the result displayed in Figure 5 below:

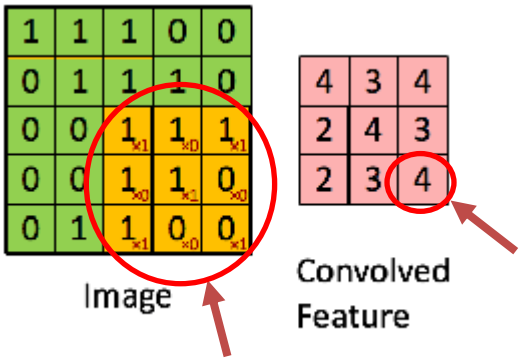

Figure 5. Convolution result. — Source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

In CNN terminology, the yellow square is called a ”filter” or a ”kernel”. Filters act as feature detectors from the original input image. [7]

3.2.2 Feature map

Numerous convolutions are performed on the input, where each operations uses a different filter. This results in different feature maps. After, all the feature maps are put together as a final output of the convolution layer. [7]

The size of the feature map is controlled by three attributes that are defined before the convolution step is performed:

- **Depth** is the number of filters used for the convolution operation.
- **Stride** is the size of the step the convolution filter moves each time.
- **Padding** adds a zero-value pixel layer around the input borders to prevent feature map from shrinking.

There is not any real set standard for these parameters. This is because the network heavily depends on the type of data and data can vary.

3.2.3 Pooling layer

To reduce training time and control overfitting, it is common to add pooling layer between CNN layers. There are several pooling layer options, with max pooling being the most popular. As show in Figure 6, max pooling takes the maximum value in each window. This decreases the feature map size while keeping the significant information. [5]

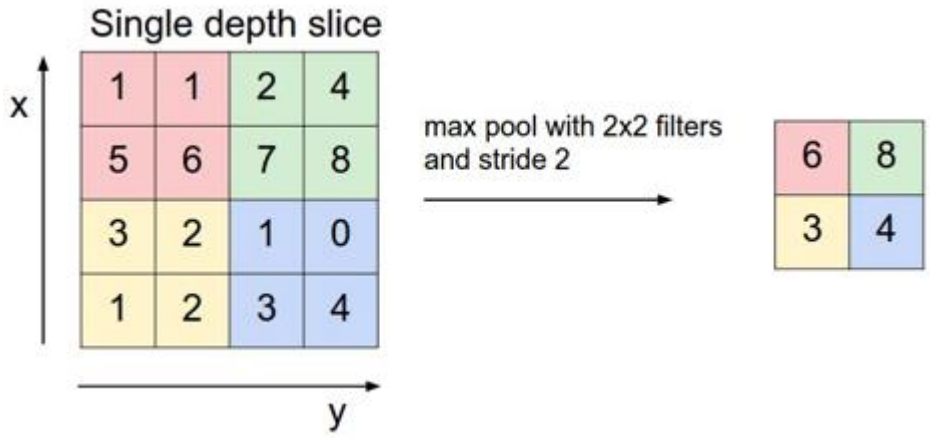

Figure 6. Max pooling takes the largest value from each window. — Source: http://cs231n.github.io/convolutional-networks/

3.2.4 Output

The output from convolutional and pooling layers represent high-level features of the input image. After feature extraction the data is classified into various classes. This can be done using a fully connected layer. Fully connected layers act the same way as a normal neural network, they have full connection to all the activations in the previous layer [5]. Adding a fully connected layers is also a cheap way of learning non-linear combinations of these features. Most of the features from convolutional and pooling layers may be good for the classification task, but combinations of those features might be even better. [7]

Training a CNN is done using backpropagation or gradient descent. Convolution and pooling layers act as feature extractors and fully connected layers as a classifier. Training steps are shown in Figure 7.

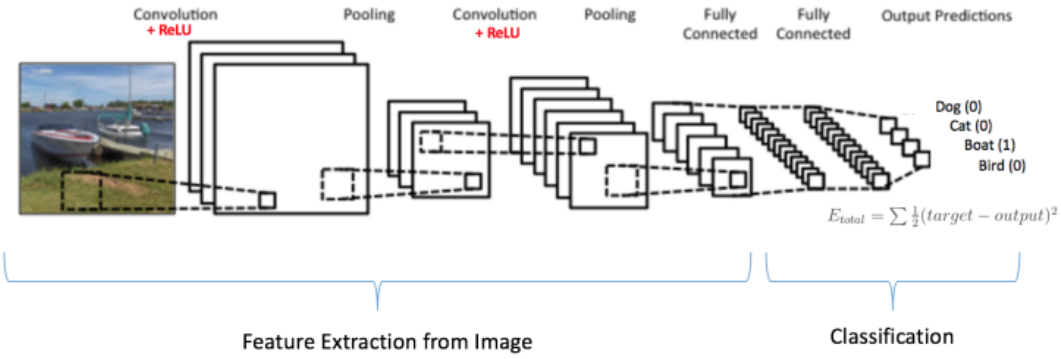

Figure 7. Training the CNN. — Source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

4 DEEP LEARNING ALGORITHMS FOR OBJECT DETECTION

Today, there is a huge amount of deep learning based object detection methods. The most widely used methods include Faster R-CNN, You Only Look Once (YOLO) and Single Shot MultiBox Detector (SSD) [8]. This chapter compares the differences between these three methods.

4.1 Faster R-CNN

Faster R-CNN was published in 2015 by Girshick et al. [9]. The ”R” stands for ”Region-based”. It is a third iteration of the R-CNN, the previous ones being R-CNN and Fast R-CNN.

Faster R-CNN uses Region Proposal Network (RPN) to generate regions of interests. Faster R-CNN also introduces anchor boxes to handle variations in aspect ratio and scale of objects. Pipeline can be seen in figure 8.

The default configuration of Faster R-CNN contains 9 anchors at a position of an image, which predicts the probability of it being background or foreground. Based on the results found [12], Faster R-CNN processes about 7 FPS (frames per second) for PASCAL VOC 2007 testing set, which is not enough for real-time object detection.

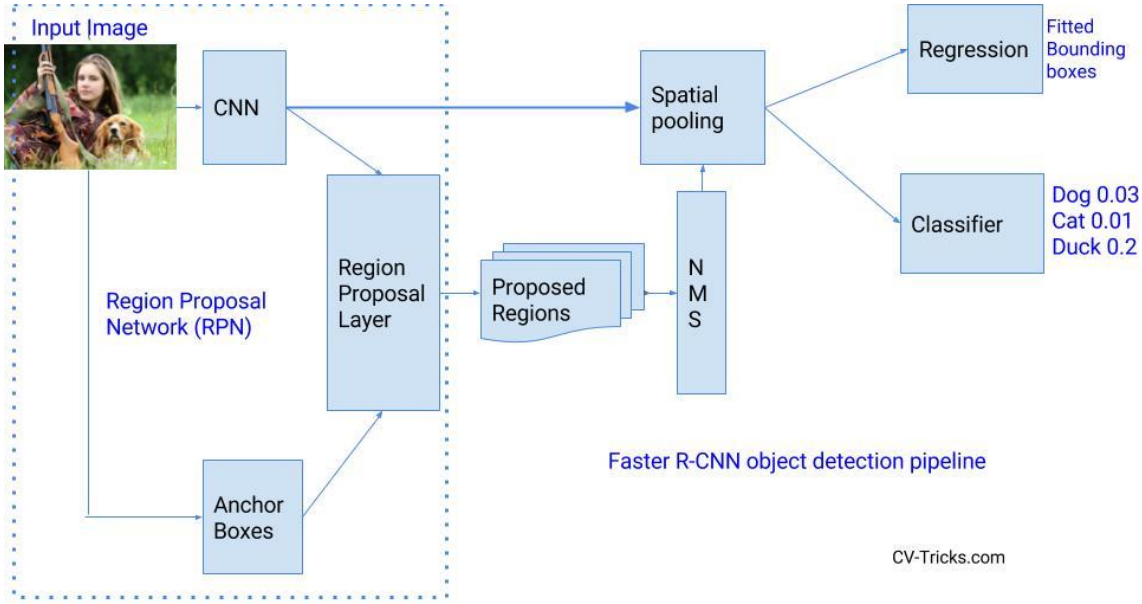

Figure 8. Faster R-CNN pipeline. — Source: http://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/

4.2 You Only Look Once (YOLO)

You Only Look Once (YOLO) was published in 2015 by Redmon et al. [10]. Third version of YOLO, called YOLOv3 was released in May, 2018.

YOLO and SSD are both Single Shot Detectors. So what is the difference with Faster R-CNN? Faster R-CNN performs region proposal and region classification in two separate steps. Single Shot Detectors do both in a ”single shot”, simultaneously predicting the bounding box and the class as it handles the image.

YOLO divides each image into a S × S grid and each grid predicts N bounding boxes and confidence. The confidence reflects the accuracy of the bounding box and whether the bounding box actually contains an objects, regardless of class. [13]

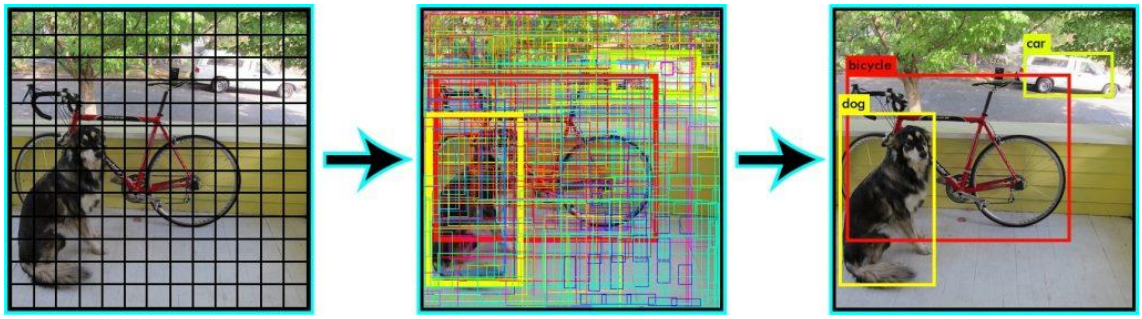

Figure 9. How YOLO handles bounding boxes. — Source: http://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/

But as displayed in Figure 9, most of the bounding boxes have low confidence score. Low confidence bounding boxes can be eliminated by setting a threshold, which is set to 30% in Figure 9. Also notice that in runtime, the image was run on CNN only once, which makes YOLO alot faster than Faster R-CNN and can be run in real-time. [13]

YOLOv3 is an updated version of YOLO9000, also known as YOLOv2.

YOLOv2 had many improvements compared to YOLOv1, including anchor boxes and a new classification model called Darknet-19. Darknet-19 has 19 convolutional layers and 5 maxpooling layers, as displayed in figure 10. YOLOv2 runs at 45 FPS on Titan X.

Type	Filters	Size/Stride	Output
Convolutional	32	3 × 3	224 × 224
Maxpool		2 × 2/2	112 × 112
Convolutional	64	3 × 3	112 × 112
Maxpool		2 × 2/2	56 × 56
Convolutional	128	3 × 3	56 × 56
Convolutional	64	1 × 1	56 × 56
Convolutional	128	3 × 3	56 × 56
Maxpool		2 × 2/2	28 × 28
Convolutional	256	3 × 3	28 × 28
Convolutional	128	1 × 1	28 × 28
Convolutional	256	3 × 3	28 × 28
Maxpool		2 × 2/2	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Maxpool		2 × 2/2	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	1000	1 × 1	7 × 7
Avgpool		Global	1000
Softmax			

Figure 10. Darknet-19 convolutional layers. (YOLO9000: Better, Faster, Stronger, 2016)

As mentioned above, YOLOv3 is the latest version of YOLO. Compared to YOLOv2, YOLOv3 is slower but more accurate. YOLOv3 has a new network for feature extraction, called Darknet-53, and because detecting smaller objects was a problem with YOLOv2, it also makes detections at three different scales. Darknet-53 layers can be seen in Figure 11. The scales are given by downsampling the dimensions of the input image by 32, 16, and 8 respectively. YOLOv3 runs at 30 FPS on Titan X. [14]

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 11. Darknet-53 convolutional layers. (YOLOv3: An Incremental Improvement, 2018)

4.3 Single Shot MultiBox Detector (SSD)

Single Shot MultiBox Detector (SSD) was published in 2015 by Liu et al. [11].

Single Shot Detector, originally developed by Google, is a combination of speed and accuracy. SSD runs a CNN on input image only once and outputs a feature map. After, it runs a small 3 × 3 sized convolutional kernel on the feature map to predict bounding boxes and classification probability. SSD also has anchor boxes similar to Faster R-CNN, but SSD learns the off-set and class probability rather than the box. And to handle object scale of various sizes, SSD predicts bounding boxes after multiple convolutional layers, since each convolutional layer acts at a different scale. [13] Scaling can be seen in figure 12.

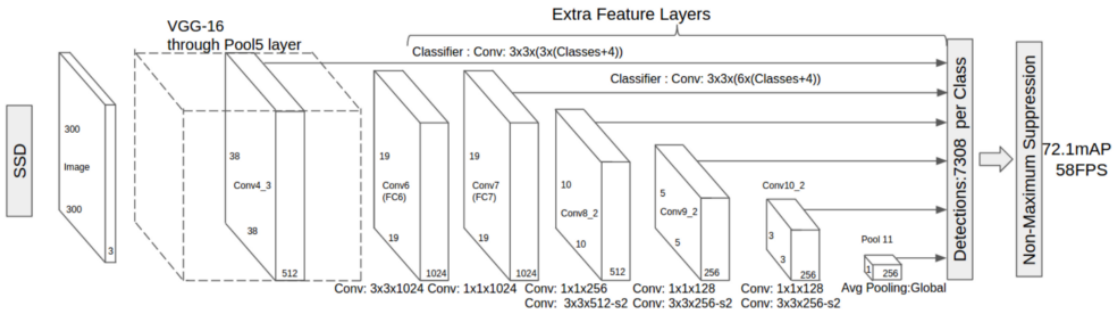

Figure 12. SSD convolutional layer scaling. — Source: https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9

Finally, SSD uses a method called non-maximum suppression to group together overlapping bounding boxes into a single box. Non-maximum suppression looks through the boxes that contains the same object, finds the one with the highest confidence and discards the rest.

5 DATASETS AND TOOLS

For real-life applications, decisions between speed, accuracy and resources has to be made. Some methods are more suitable for the application than others.

There is already a great amount of different results between different methods found from the internet. Most tests and training are done with datasets, and results are measured in mean Average Precision (mAP) at Intersection over Union (IoU) threshold.

IoU measures the overlap between two regions. This means that how good is the prediction in the object detector with the ground truth, the real object boundary.

5.1 Datasets

A dataset is a collection of data. Image datasets are used to train and benchmark object detection algorithms.

5.1.1 PASCAL Visual Object Classification

The PASCAL Visual Object Classification (PASCAL VOC) dataset is very popular for building and evaluating algorithms for image classification, object detection and segmentation. It contains 8 different challenges spanning from 2005 to 2012, each having their own specifications [15]. The 2012 version contains 11 530 images and 20 different classes [16].

5.1.2 ImageNet Large Scale Visual Recognition Challenge

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) was introduced in 2013. The 2014 dataset contains roughly 500 000 images only for training, 40 000 for testing and 200 classes. Due to the size of the dataset and number of classes, the dataset requires a great amount of computing power and that is why it is rarely used. [15]

5.1.3 Common Objects in Context

Common Objects in Context (COCO) is developed by Microsoft and was introduced in 2015. The 2017 dataset contains over 120 000 images for training and validation, over 40 000 images for testing and 80 classes. [15]

5.2 Frameworks

A framework can be defined as a set of libraries and tools to work with. It is a skeleton, which the user can build the application on top of.

Frameworks are extremely important these days, because they help save time. Frameworks take care of the low-level functionality.

There are many deep learning frameworks and most of them are viable tools. Some of the most popular frameworks include TensorFlow, Torch/PyTorch and Caffe.

TensorFlow, PyTorch and Caffe all have model zoos. Model zoo is a collection of pre-trained models trained on a dataset. Model zoo usually also lists the models accuracy in mAP and speed per image on the dataset, as seen on figure 13.

5.2.1 TensorFlow

TensorFlow is an open source software library, developed by Google Brain team and used by several giants such as Airbnb, Twitter, Snapchat, NVIDIA and Dropbox [17].

Probably the most well known use of TensorFlow is Google Translate. Google Translate includes natural language processing, text classification/summarization and speech/image/handwriting recognition.

5.2.2 Torch/PyTorch

Torch is a Lua-based machine learning library, a scientific computing framework. It provides a wide range of algorithms for deep learning.

PyTorch is an open source machine learning library for Python, based on Torch. PyTorch is primarily developed by Facebook.

PyTorch is used and developed by, for example Facebook, Twitter, NVIDIA, Stanford University and University of Oxford [18].

5.2.3 Caffe

Caffe (Convolutional Architecture for Fast Feature Embedding) is a deep learning framework, originally developed at UC Berkeley.

Caffes biggest advantage is speed. Caffe can process over 60 million images in a day with a single NVIDIA K40 GPU [19].

Caffe2 is new version of Caffe and is aimed towards mobile phones and other relatively computationally constrained platforms. Caffe2 is used by Facebook for fast style transfer on their mobile application.

5.2.4 Pre-trained model

A pre-trained model is a model created by someone else. Instead of building and training a model from a scratch, a pre-trained model can be useful for out-of-the-box inference or used as a starting point. A pre-trained model may not be 100% accurate, but it saves a lot of time.

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes
mask_rcnn_inception_resnet_v2_atrous_coco	771	36	Masks
mask_rcnn_inception_v2_coco	79	25	Masks
mask_rcnn_resnet101_atrous_coco	470	33	Masks
mask_rcnn_resnet50_atrous_coco	343	29	Masks

Figure 13. COCO-trained TensorFlow models. — Source: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

5.3 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library, originally developed by Intel. OpenCV is mainly aimed at real-time computer vision.

The library has more than 2500 optimized algorithms, which includes both the classic and state-of-the-art computer vision and machine learning algorithms.

OpenCV is written in C++ and it is also its primary interface, but there are bindings and wrappers in Python, Java, MATLAB/OCTAVE, C#, Perl, Ch, Haskell and Ruby.

OpenCV supports the deep learning frameworks TensorFlow, Torch/PyTorch and Caffe.

6 DEVELOPMENT

The coding for this project was implemented in Python language, OpenCV library and Tensorflow framework.

During the process, different frameworks and pre-trained models were tested, including TensorFlow, Caffe and PyTorch.

Due to the limitations in computing power, the model had to be small and fast.

Tensorflow was chosen as a framework because it was easy to implement and the pre-trained models were easy to use due to freeze graphs.

Training of a model was also tested, hoping to acquire better accuracy in pedestrians from a bird's eye view.

6.1 Training

The training was mostly a test, and it was carried using only around 200 pictures, as training usually consists of thousands of images. To save time, the images were downloaded using Bing Search API and they were manually labeled using LabelImg image annotation tool as shown in Figure 14, and ssd_mobilenet_v1_coco pre-trained model was used as a training checkpoint. Checkpoints are versions of the model created during training and they are used so that the training has some ”basic knowledge” of the trained object which in turn saves time.

Figure 14**.** LabelImg outputs an .xml file that includes the label and coordinates of bounding boxes in the image. Bounding boxes are manually selected.

Because no training capable hardware was available, training was performed using ML Engine and Storage on the Google Cloud Platform. The Google Cloud Platform is a set of cloud computing services and it runs on the same infrastructure that Google uses for its own products, such as Google Search and YouTube [20].

Google Cloud Platform is not free to use but at registration the user is granted 250 euros in credits, which was sufficient for this test.

When starting a training job in ML Engine, the user needs to choose the needed machine type. The list of available machine types is shown in Figure 15.

For this training test, complex_model_m_gpu was used because all smaller ones ran out of memory during the first steps of training, probably because the size of the images were not considered when using Bing Search API and some of them ended up being quite large.

Machine type	Compute Engine machine name	Virtual CPUs	GPU / TPU	Memory (GB)
standard	n1-standard-4	4	-	15
large_model	n1-highmem-8	8	-	52
complex_model_s	n1-highcpu-8	8	-	7.20
complex_model_m	n1-highcpu-16	16	-	14.4
complex_model_l	n1-highcpu-32	32	-	28.8
standard_gpu	n1-standard-8	8	1 (K80 GPU)	30
complex_model_m_gpu	n1-standard-16	16	4 (K80 GPU)	60
complex_model_l_gpu	n1-standard-32	32	8 (K80 GPU)	120
standard_p100 *(Beta)*	n1-standard-8	8	1 (P100 GPU)	30
complex_model_m_p100 *(Beta)*	n1-standard-16	16	4 (P100 GPU)	60
cloud_tpu *(Beta)*	custom	50	8 (TPU cores)	128 or 256, depending on allocated zone

Figure 15. The list of available machine types in ML Engine. — Source: https://cloud.google.com/ml-engine/docs/tensorflow/machine-types

The training was monitored using Tensorboard, a visualization tool.

As the training went on, the accuracy kept decreasing (as shown in Figure 16.) and so the training was terminated at around 4 hours.

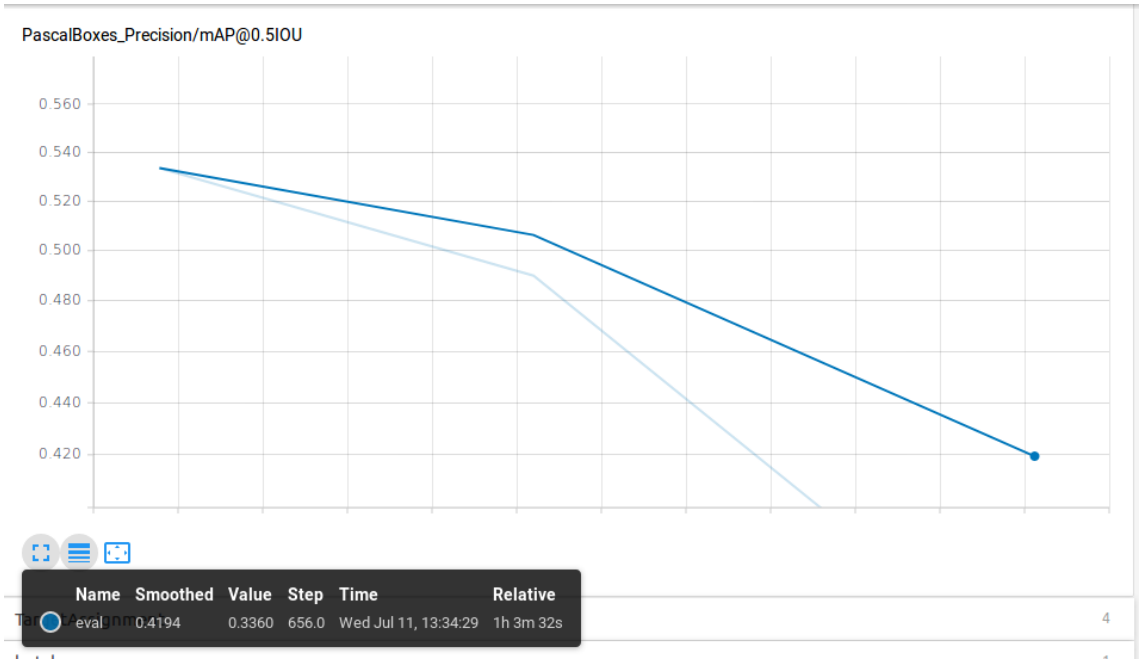

Figure 16. As the graph displays, precision kept decreasing during training.

This test showed that 200 labeled images for a complex class like a pedestrian is not even nearly enough, even with a checkpoint, which are versions of the model created during training. Labeling more, even a thousand images and then training the model is so time consuming and expensive while running on cloud platform without any quarantee of success, so the idea was unsuitable for this project.

6.2 Loading the model

Because the goal of this project was to be able to run the application on affordable hardware and in real-time, ssdlite_mobilenet_v2_coco was chosen as a pre-trained Tensorflow detection model after numerous tests and comparisons.

To load the model with Tensorflow, the user needs a frozen GraphDef file, usually ending with a .pb or .pbtxt extension. Frozen graph proto includes all the necessary weights needed for inference as constants but discards all the data needed for training.

To acquire the real names of the predictions, a label map is also needed. Label maps map indices to category names, so when a neural network for example predicts 3, we know that this corresponds to ”car” if the model was trained using COCO dataset. Most

of the well-known datasets label maps are already supplied within Tensorflow as separate files.

6.3 Detection

After the neural network has finished the predictions based on the input image and given model, it outputs the detections as tensors in an array. These tensors are detection boxes, scores, classes and number of detections. Each score represents the level of confidence for each object, and detection boxes represent the bounding box location around each detected object. An example of the output is shown in Figure 17.

Results can be filtered based on confidence score or class, for example if the score is less than 40% the result is dropped.

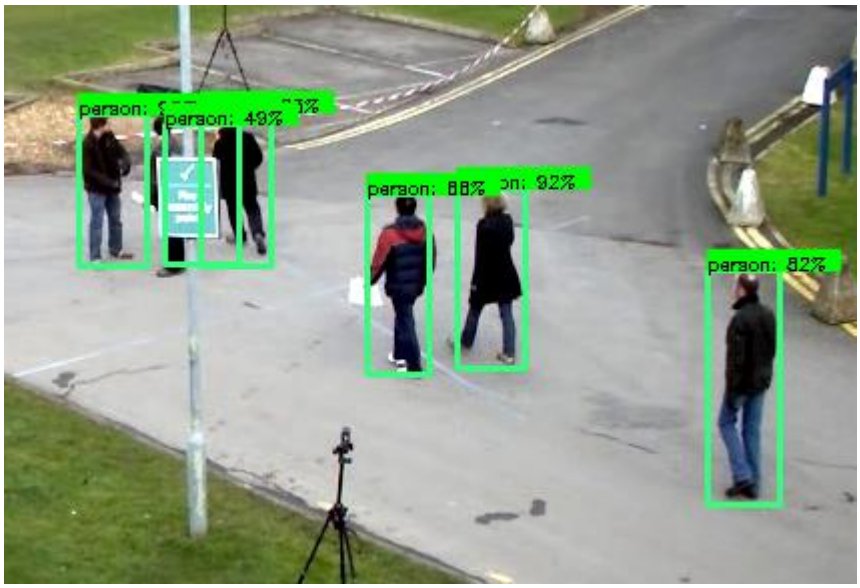

Figure 17. An example of the detection output.

And since a video is just a sequence of images, the detector does this phase for each frame if the input is a video file or a camera stream.

6.4 Tracking

To acquire better accuracy in counting, an object tracking method was implemented.

There are several different tracking methods available, each having their own algorithms for tracking and most are able to track only one object at once.

6.4.1 Deep SORT

Simple Online and Realtime Tracking (SORT) is a pragmatic approach to multiple object tracking with a focus on simple, effective algorithms. [21]

Simple Online and Realtime Tracking with a Deep Association Metric, also known as Deep SORT is a new and improved version of SORT, implemented in Tensorflow.

Online tracking means that only detections from the previous and the current frame are presented to the tracker and the tracker produces object identities on the fly.

The difference with Deep SORT compared to other tracking methods is that it uses a pre-trained CNN descriptor to reduce identity switches. The CNN available at Deep SORTs Github was trained on a dataset that contains over million images of 1 261 pedestrians [21], making it well suited for this project.

The first steps of Deep SORT work very much the same way as tensor detection. First, it extracts features from the given frame and bounding boxes. Then it generates detections from the found features and boxes. After the detections have been generated, Deep SORT can run non-maximum suppression through the detections to reduce the number of boxes by merging boxes that reside inside one another. After NMS the tracker ”predicts” the objects point in next frame and updates the true location of the object.

Program 1. The tracking process of Deep SORT

```
        self.features = self.encoder(self.pure_frame, boxs)


        # Load image and generate detections.
        self.detections = [
            Detection(bbox, self.nms_max_overlap, feature) for bbox,
feature in zip(boxs, self.features)]
```

```
# Run non-maxima suppression.
boxes = np.array([d.tlwh for d in self.detections])
scores = np.array([d.confidence for d in self.detections])
indices = preprocessing.non_max_suppression(
    boxes, self.nms_max_overlap, scores)
self.detections = [self.detections[i] for i in indices]


# Update tracker.
self.deep_mot_tracker.predict()
self.deep_mot_tracker.update(self.detections)
```

6.5 Counting

Counting is done with tripwires. Tracking gives the ability to assign id numbers for all tracked objects. When the tripwire notices that an object has crossed its perimeter, it checks which way the object was coming from based on the bounding box center point and if the objects id number has already passed that direction to avoid double counting. So an object can go in and out only once. But if an object (person) goes out of frame for long enough and is later detected again, it is considered as a new object and is assigned new id.

As demonstrated in Figure 18, the person with id number 1 comes from right, crosses the tripwire and ”out” count is incremented by 1. Even though the person is lost right after the tripwire is tripped, it does not affect the result. If the person would go back right, ”in” count would be incremented by 1 and that id would not be counted ever again for that tripwire.

Figure 18. An object crossing the tripwire.

All trips and their direction are sent as ZeroMQ messages for further processing.

6.6 Testing

Testing was mostly done using pre-recorded video footage, so it would be easy to track what is happening and adjust code accordingly.

During testing the only hardware available was Intel NUC, which was running 5th generation Intel® Core™ i3-5010U processor. The NUC was able to run test code with tracking enabled at around 4-6 frames per second and without tracking at around 6-8 frames per second. These speeds are barely enough for real-time object detection/tracking, so some more optimization or slightly better hardware would be required.

In testing, tripwires were drawn over the frame and the ideal scenario would have been that when a person found by object detection crossed the line, in/out counter would be incremented by 1, depending on the persons direction.

The previous version of Fideras person counter only allowed to draw tripwires in horizontally straight lines, which made it alot easier to count trips. In this new version, tripwires can be drawn in any direction.

Testing was done with two different direction tripwires, 4 different detection thresholds and with tracking enabled and disabled to acquire a good idea how tracking actually effects the accuracy. Trips were also manually counted to acquire the real trip count.

In Figure 19, the ”up + down” count means the real trip count that was manually counted and ”Tripped” the detected trip count.

Vertical tripwire 8 up + 6 down				Horizontal tripwire 17 up + 12 down			
Threshold	Tripped	Tracking?	Accuracy	Threshold	Tripped	Tracking?	Accuracy
40 %	8 + 4	Yes	83 %	40 %	15 + 13	Yes	90 %
	4 + 10	No	55 %		22 + 18	No	72 %
60 %	8 + 5	Yes	92 %	60 %	13 + 11	Yes	84 %
	7 + 5	No	85 %		16 + 22	No	75 %
70 %	7 + 4	Yes	77 %	70 %	12 + 9	Yes	73 %
	7 + 4	No	77 %		16 + 13	No	93 %
80 %	4 + 3	Yes	50 %	80 %	9 + 8	Yes	70 %
	5 + 3	No	56 %		13 + 13	No	84 %

Figure 19. Tripwire test results.

Based on these results, some estimations can be made that tracking helps atleast on lower thresholds. On higher thresholds, without tracking actually has better accuracy.

There are so many components that can affect the counting accuracy, so it is really hard to give a rough estimate of average precision. But to acquire better accuracy, it is easy to just change the detection model, but in this case that was not possible due to the hardware limitations.

6.7 Summary

Summary of all the used and tested methods and tools are listed in figure 20 below.

SUMMARY	
TensorFlow	+ Easy to implement + Lots of support and guides available + Tensorboard for debugging and analysis - Slower than other frameworks
Caffe	+ Somewhat easy to implement - Poor documentation - Dying community
PyTorch	+ Lots of pre-trained models - Poor documentation - Hard to implement
Google Cloud Platform ML Engine and Storage	+ Easy to use + Good documentation + Trial credits - Lots of trial and error when trying to find a working machine type for the current task
Tensorboard	+ Easy to use + Good documentation + Runs in browser - Buggy and required manual restart to update statistics - Not very lightweight
Deep SORT	+ CNN descriptor - Allows only minimal frame skipping for TensorFlows detector, which causes lots of computing power consumption

Figure 20. Summary of all the used and tested methods and tools.

7 CONCLUSION

Object detection using deep learning and neural networks has taken some massive leaps in the past couple of years, and the field is very popular at the moment. Every month someone releases a new research paper, a new algorithm or a new solution for a certain problem.

The purpose of this thesis was to examine different deep learning algorithms and develop a new version of Fideras already existing person counter by using deep learning for detection.

The main part of the goal was successfully implemented, a working application which utilizes neural network model for object detection. However, the application does not run smoothly on the current test hardware, so some changes need to be implemented in the nearby future to acquire more speed. A computer utilizing an NVIDIA GPU would be the best choice, because in deep learning and due to CUDA -architecture, utilizing an NVIDIA GPU can be multiple times faster than a CPU.

This whole project took about two and a half months to complete, which includes researching, documentation, development and testing. Researching and documentation probably took the longest period.

Finding the most suitable framework and tools was fulfilled mostly with trial and error. At first, benchmarking different pre-trained models and frameworks was executed with simple demos found publicly on GitHub. After TensorFlow was chosen as the main framework, coding started and after the code was mostly finished, benchmarking was performed running the main application and testing different models. However, even after testing the most lightweight and smallest models, the test hardware was not able to run the application on sufficiently high framerate.

Before this project the author’s knowledge in deep learning and neural networks was minimal and almost nonexistent. Completing this project has given knowledge, ideas and skills to work in future project involving deep learning and neural networks.

The development of the product will continue but in later time on better test hardware.

REFERENCES

[1] What Is Deep Learning? | How It Works, Techniques & Applications. Available: https://www.mathworks.com/discovery/deep-learning.html [Jun 8, 2018].

[2] What is Computer Vision? - Definition from Techopedia. Available: https://www.techopedia.com/definition/32309/computer-vision [Jun 8, 2018].

[3] HIMANI, S., PAREKH, DARSHAK, G., THAKORE, UDESANG, K. and JALIYA, 2014. A Survey on Object Detection and Tracking Methods. 2. Available: https://pdfs.semanticscholar.org/25a6/c5dff9a7019475daa81cd5a7f1f2dcdb5cf1.pdf [Jun 11, 2018].

[4] Convolutional Neural Networks for Object Detection. 2016. Available: https://www.azoft.com/blog/convolutional-neural-networks/ [Jun 11, 2018].

[5] CORNELISSE, D., An intuitive guide to Convolutional Neural Networks. Available: https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050 [Jun 11, 2018].

[6] DORMEHL, L., What is an artificial neural network? Here's everything you need to know. Available: https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/ [Jun 12, 2018].

[7] An Intuitive Explanation of Convolutional Neural Networks. 2016. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ [Jun 12, 2018].

[8] ROSEBROCK, A., Object detection with deep learning and OpenCV. 2017. Available: https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/ [Jun 13, 2018].

[9] REN, S., HE, K., GIRSHICK, R. and SUN, J., Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. 2015. Available: https://arxiv.org/abs/1506.01497 [Jun 13, 2018].

[10] REDMON, J., DIVVALA, S., GIRSHICK, R. and FARHADI, A., You Only Look Once: Unified, Real-Time Object Detection. 2015. Available: https://arxiv.org/abs/1506.02640 [Jun 13, 2018].

[11] LIU, W., ANGUELOV, D., ERHAN, D., SZEGEDY, C., REED, S., FU, C. and BERG, A.C., SSD: Single Shot MultiBox Detector. 2015. Available: https://arxiv.org/abs/1512.02325 [Jun 13, 2018].

[12] HUI, J., Object detection: speed and accuracy comparison. 2018. Available: https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359 [Jun 14, 2018].

[13] Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSD – CV-Tricks.com. Available: http://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/ [Jun 14, 2018].

[14] KATHURIA, A., What's new in YOLO v3? 2018. Available: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b [Jun 14, 2018].

[15] OUAKNINE, A., Review of Deep Learning Algorithms for Object Detection. 2018. Available: https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852 [Jun 13, 2018].

[16] The PASCAL Visual Object Classes Homepage. Available: http://host.robots.ox.ac.uk/pascal/VOC/ [Jun 13, 2018].

[17] TensorFlow. Available: https://www.tensorflow.org/ [Jul 2, 2018].

[18] PyTorch. Available: https://pytorch.org/ [Jul 2, 2018].

[19] Deep Learning for Computer Vision with Caffe and cuDNN. 2014. Available: https://devblogs.nvidia.com/deep-learning-computer-vision-caffe-cudnn/ [Jul 2, 2018].

[20] Google Cloud Platform. 2018. Available: https://en.wikipedia.org/wiki/Google_Cloud_Platform [Jul 17, 2018].

[21] WOJKE, N., BEWLEY, A., PAULUS D., Simple Online and Realtime Tracking with a Deep Association Metric, 2017. Available: https://arxiv.org/abs/1703.07402 [Jul 30, 2018].