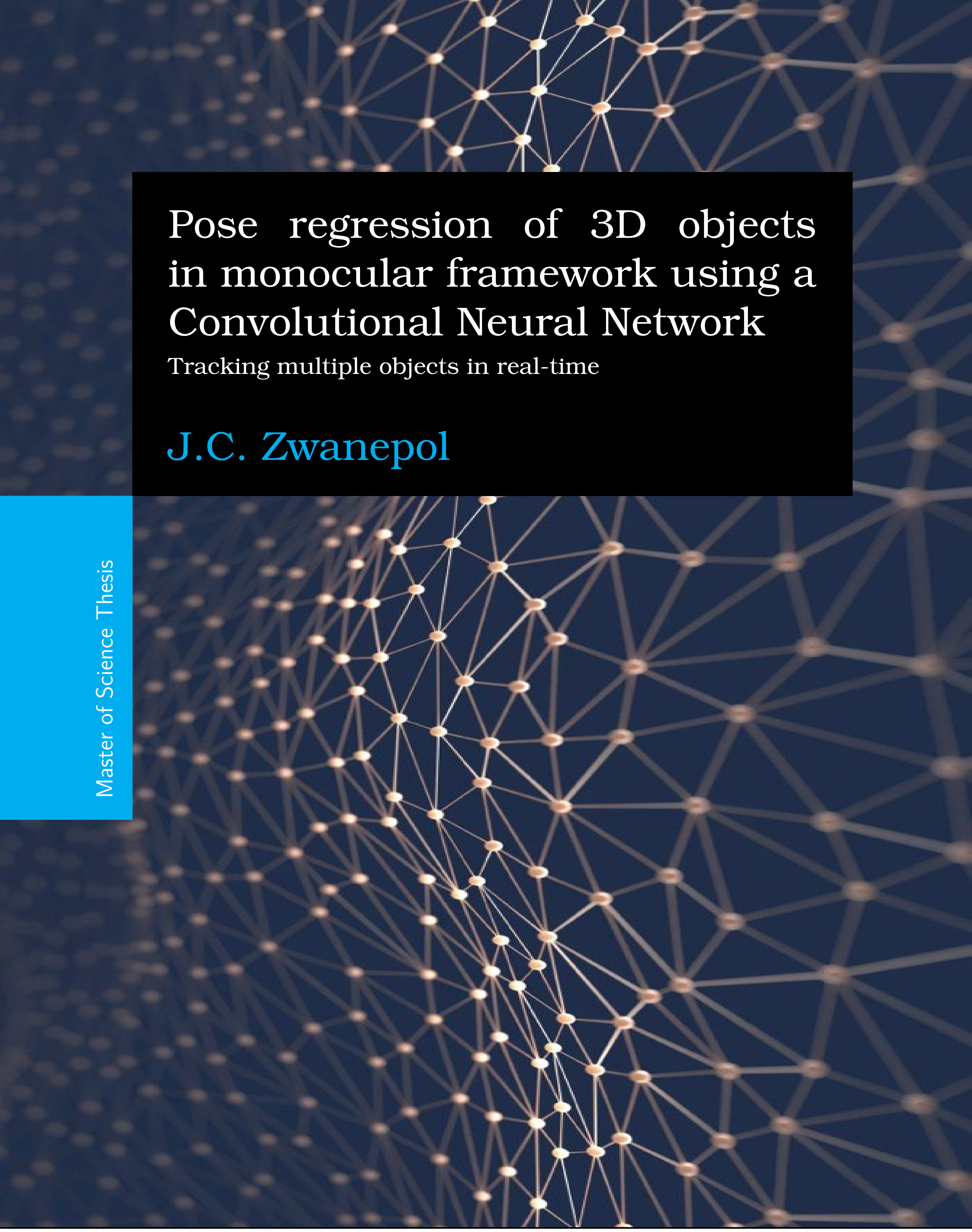
Pose regression of 3D objects in monocular framework using a Convolutional Neural Network

Tracking multiple objects in real-time

J.C. Zwanepol

Master of Science Thesis



Delft University of Technology

Bio-Mechanical Design

Pose regression of 3D objects in monocular framework using a Convolutional Neural Network

Tracking multiple objects in real-time

Master of Science Thesis

For the degree of Master of Science in Bio-Mechanical Design at Delft University of Technology

J.C. Zwanepol

April 30, 2018

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was supported by Mainblades Inspections. Their cooperation is hereby gratefully acknowledged.

Abstract

In computer vision pose estimation of objects in everyday scenes is a basic need for a clear understanding of the surrounding environment, fields of interests include; augmented reality, surveillance, navigation, manipulation, and robotics in general. Pose estimation is a well studied topic, however fast and robust solutions are still hard to obtain. The goal of this research is to robustly and efficiently perform 3D pose estimation of multiple objects within a single RGB image in real-time (>24 **f**rames **p**er **s**econd (fps)).

To achieve this goal an existing CNN is utilized, more specifically the YOLO network is used. This provides a stable platform for object detection and classification, the network is only slightly modified to include pose regression. The YOLOv2 network was originally designed to generate bounding boxes around objects and classify the objects within the bounding boxes. This research shows that using a single confidence value rather than 4 bounding box parameters is sufficient to determine the relative location of objects within the image, limiting the number of parameters that need to be trained. This has allowed to make the network more efficient and to make the network focus more on training the pose parameters (azimuth,elevation and distance) rather than the bounding box parameters.

Using several techniques like data augmentation, data clustering and data selection the state-of-the-art AVP of 50.1% was achieved on the azimuth estimation problem. For the full 3D pose (azimuth, elevation and distance) problem the AVP is limited to 30.1%, with no direct comparison this is still considered to be state-of-the-art. However, normalizing the confidence output of each image in the post-processing step has increased this accuracy further, improving beyond the state-of-the-art results. With a normalization step the AVP score has reached 63.0% and 40.4% respectively. These results show that the pose estimation problem has improved significantly, getting closer to a viable solution for real-world applications.

However, further improvements can be made in the post-processing step, the data augmentation step and the data selection step. The research conducted has shown that accuracy gains are not only achieved through better network architecture but are also highly dependent on the training and processing techniques used. This is evident from the accuracy increase of 38% from an AVP of 25% to an AVP of 63%. Optimizing these techniques specifically for the YOLO$_{pose}$ architecture, might result in a solution that can be used in real-world applications.

Table of Contents

List of Figures

List of Tables

Preface

This master's thesis is written in collaboration with Mainblades Inspections with supervision from the Delft University of Technology. It is part of the Bio Robotics specialization in the Bio Mechanical Design track. The work focuses on pose regression of 3D objects in monocular framework using a Convolutional Neural Network. It builds on the research conducted in the literature survey titled *"Mapping of environments containing reflective 3-D objects for vision-based localization"*.

I would like to thank my supervisors dr. ir. Jan van Gemert at the TU Delft together with dr. ir. Hamdi Dibeklioglu and prof. dr. ir. Pieter Jonker. I would also like to thank ir. Dejan Borota and ir. Jochem Verboom at Mainblades Inspections for their support and guidance during the writing of this thesis.

Delft, University of Technology
April, 2018

Jacco Zwanepol

Chapter 1

Introduction

Visual perception is an important part of the human senses that forms an integral part of how we interact with our environment. Through it we can play sports, drive cars and recognize our friends and family. For this interaction we have to understand what we are looking at, recognize the individual objects and estimate position and orientation of objects of interest. This requires a certain level of signal processing that our brain is particularly good at, to the point where it seems like an almost effortless and instantaneous task. Like humans, robots have similar sensors to interact with their environment. Instead of having eyes, robots use cameras or lasers to observer their environment. However, robots cannot (yet) accurately mimic the functions of the brain. Robots use complex algorithms to interpret sensor data that is often task specific and limited by its accuracy, robustness or required processing power. For a robot to mimic part of the visual perception capabilities of our brain requires a combination of these complex algorithms. This Results in a system that is complex in design and operation, often requiring parameters to be tuned to fit different environments. These complex systems require significant processing power to run in real-time. However, In recent years progress has been made into realizing this goal of human like visual perception capabilities through the introduction of **N**eural **N**etwork (NN) models. Instead of algorithms being designed for specific tasks, NNs can be trained instead of programmed. Like our brain, that learns to do different tasks in a unique way by learning from mistakes and past experiences, similarly a NN learns to see patterns and correlations in a unique way through a machine learning process. This new technique has lead to multiple breakthroughs in computer vision fields, like in classification and object detection.

This thesis will focus on:

- **Object detection:** Detecting multiple objects in single RGB images.
- **Object classification:** Classifying individual objects that have been detected.
- **Pose estimation:** Estimate 3D poses of individual objects that have been detected.
- **Real-time operation:** Ability to Run at 24 **f**rames **p**er **s**econd (fps).
- **Dataset:** Ability to train and perform validation on PASCAL3D+ dataset.

1-1 Basic Principles

1-1-1 Object detection

In the computer vision field, object detection has been around for a long time with related work dating back to 1963 [1]. Until recently the standard techniques for object detection were geometric based methods that extract 2D features, like [2, 3, 4, 5]. These techniques often resulted in case variant results dependent on lighting conditions, object representation and/or object visibility (i.e. occlusion). With the introduction of **C**onvolutional **N**eural **N**etworks (CNNs) a new height was achieved in the precision of object detection and classification. In the 2012 **I**mageNet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge (ILSVRC) [6] the CNN architecture called AlexNet [7] won the image classification competition scoring 11% higher than the runner-up. From then on object detection and classification problems have been dominated by CNNs, resulting in architectures like **R**ecurrent **C**onvolutional **N**eural **N**etwork (R-CNN) [8], **Y**ou **O**nly **L**ook **O**nce (YOLO) [9] and **S**ingle **S**hot **D**etection (SSD) [10] consistently showing improvements in accuracy and detection rate on multiple online datasets.

1-1-2 Pose estimation

With the success of CNN in classification and object detection, with several CNNs passing human level classification accuracy in 2015, the shift has slowly moved to pose estimation of objects. This opened up possibilities for CNN to be used in a range of new applications. Pose estimation of objects in everyday scenes is a basic need to understanding the surrounding world, fields of interests include; augmented reality, surveillance, navigation, manipulation, and robotics in general. Robotics relies heavily on pose estimation to know the location of the robot and to understand its surroundings.

Pose estimation is a well studied topic, however fast and robust solutions are still hard to obtain. Improvement in robustness often means a compromise on computational speed, this particularly applies to geometric based methods. Initially pose estimation of objects was done on single objects with their exact 3D geometries known, like [11, 12]. With the introduction of the PASCAL3D+ dataset this moved to multiple objects pose estimation, with generalized 3D geometries for objects. PASCAL3D+ introduce a way of comparing different algorithms on their performance renewing interest in the pose estimation problem.

The difficulty of pose estimation is that it requires the preservation of geometric information, unlike object detection and classification that only require view-invariant representations of the objects. In real life operating environments there are variations in shape and appearance of objects within their respective categories as well as varying viewing conditions. i.e. a car can be represented by different brands and models each with their own unique design, but the same car can also have a different appearance if doors are open instead of closed. This makes it extremely difficult to build robust models that represents the geometric properties of all objects within a scene. CNNs have the potential to solve this problem. The deep architecture of most state-of-the-art CNN architectures allow to decode large complex non-linear problems.

In this thesis the second version of the YOLO architecture [13] is used as it shows state-of-the-art result in object detection and classification on both PASCAL VOC [14] and COCO

[15] dataset. This architecture has a unique structure that allows for single shot detection of multiple objects at a fast rate. By directly integrating the pose estimation into the object detection framework, rather than using a second stage for classification, the resulting system is fast and able to train both classification and pose simultaneously.

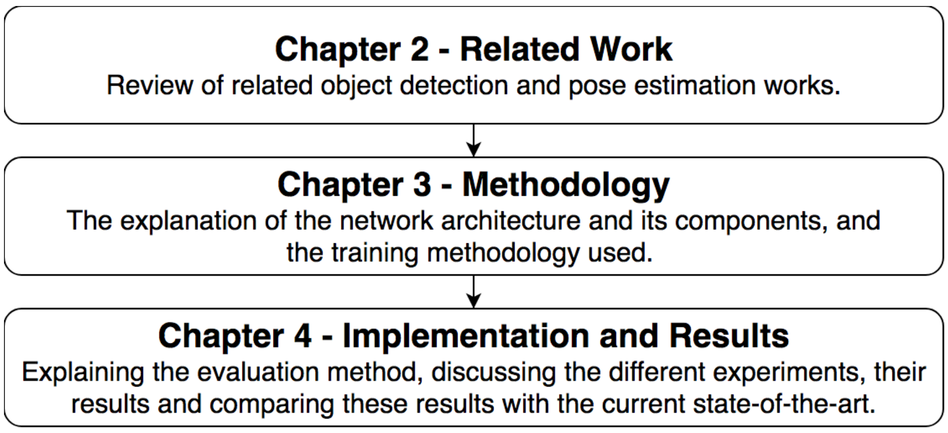
1-2 Overview

Unlike previous works the aim is to directly regress the pose of all objects from a single 2D image, using data provided by PASCAL3D+ [16]. Because CNNs are essentially black-boxes, the main focus of this paper will not be on the architecture of the CNN, rather the focus will fall on the pre-processing, training, and post-processing steps that are implemented to solve the research problem. The state-of-the-art CNN architecture YOLO [13] is used as basis to perform the pose estimation task. The YOLO architecture is design to detect and classify objects, the output of the network therefore has to be modified to allow for pose estimation. Two pose representation were tested in this research; Using x,y,z coordinates and four quaternion values to describe the pose and a spherical coordinates (azimuth, elevation and distance) to describe the pose. In the training phase it became evident that the first pose representation method did not work, for this reason the results and discussion are moved to Appendix A. The rest of the research will focus on the spherical coordinate system to desribe the pose.

The goal of this research is to design a state-of-the-art model that can accurately predict poses at real-time speeds of 24 fps. The methods that contribute to this solution are the main contributions of this paper, and can be summed up by the following:

- Adding elevation and distance to the pose, where other works only look at azimuth.
- How the pose is structured in bins, using opposite poses in each bin instead of sequential poses.
- Normalizing the confidence (objectness) scores for each image in the post-processing step.

The research is divided into 3 parts:



Chapter 2 - Related Work
Review of related object detection and pose estimation works.

Chapter 3 - Methodology
The explanation of the network architecture and its components, and the training methodology used.

Chapter 4 - Implementation and Results
Explaining the evaluation method, discussing the different experiments, their results and comparing these results with the current state-of-the-art.

Chapter 2

Related Work

In this chapter an overview of the available techniques in image based object detection and pose estimation will be reviewed. The work related to object detection will be reviewed first, then works combining both object detection and pose estimation will be introduced and discussed. Both approaches will include information on techniques that are most common and techniques that are considered state-of-the-art. Decisions made for the model architecture and related components of the model are based on these works discussed in this chapter.

2-1 Object Detection

Object detection is a core problem in computer vision with a rich history. A common approach to detection is to start with feature extraction using robust features like; Haar [17], SIFT [18], HOG [19] and convolutional features [20]. Then, using classifiers to identify objects from these features, using techniques like [21, 22, 23, 5] or making use of a localizer like [24, 25]. These techniques are run either in sliding window fashion over the whole image or on some subset of regions in the image.

A common approach that has been implemented in multiple different cases is the **D**iscriminative **P**art based **M**odel (DPM). DPM is a broader term for detection algorithms that determine where objects exist in an image by separately analyzing various parts of the image, rather than analyzing the whole. The DPM is a decoupled pipeline, extracting features first, then classifying the regions and finally predicting the bounding boxes. A popular DPM method is the constellation model that detects a small number of features, determines their relative positions and then decides whether an object is inside the region. The idea behind this technique was first introduced in [2]. However the speed, accuracy and robustness of DPM remains an issue. With the introduction of **C**onvolutional **N**eural **N**etwork (CNN) these techniques have become less common, even obsolete.

Ever since the CNN AlexNet [7] won **I**mageNet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge (ILSVRC) [6] in 2012, CNNs have become the gold standard for image classification. CNNs have improved to the point where they now outperform humans on ILSVRC. Figure 2-1 shows the results of ILSVRC over the period 2010-2014. Note the sudden change in deeplearning techniques vs traditional **C**omputer **V**ision (CV) after 2012, where CNNs have dominated the field since.

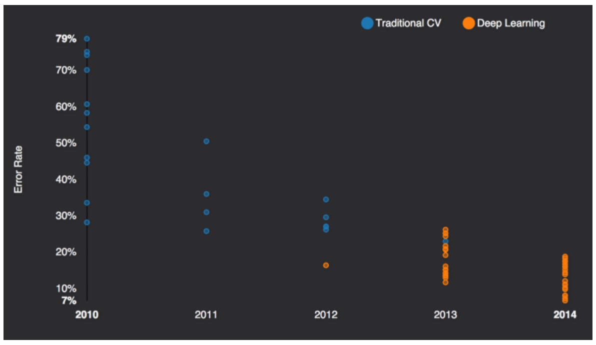


Figure 2-1: ILSVRC from 2010-2014. Sowing the results of traditional CV and deeplearning techniques for the years 2010-2014.[1]

In real world cases multiple objects can co-exist, they can overlap and backgrounds can change. CNNs are a useful tool that can help with these complex tasks. Given a more complicated image, CNNs can be designed to go as far as to identifying the different objects, determine their boundaries and identify the classes of each object. With the development of CNN architectures in combination with improved computing power and expanding training data, a CNN can now surpass human level performance in object detection. Albeit that this is for specific tasks, such as face recognition.

2-1-1 R-CNN

One of the first techniques taking advantage of CNNs, that also showed promising results, is **R**ecurrent **C**onvolutional **N**eural **N**etwork (R-CNN) [8]. R-CNN takes an image as input, proposes a bunch of regions in the image where an object might exist and then classifies and generates bounding boxes around these objects. In this pipeline R-CNN creates a bunch of region proposals using a technique referred to as **S**elective **S**earch (SS). Selective search analyses an image using different sized windows, grouping together pixels for each window size based on their texture, color, or intensity to detect relevant objects. Figure 2-2 shows an illustration of the R-CNN architecture.

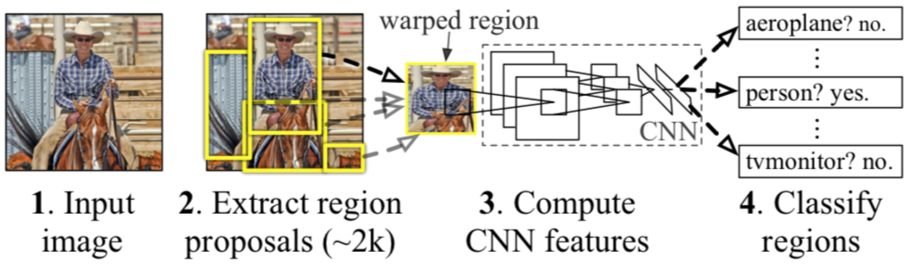


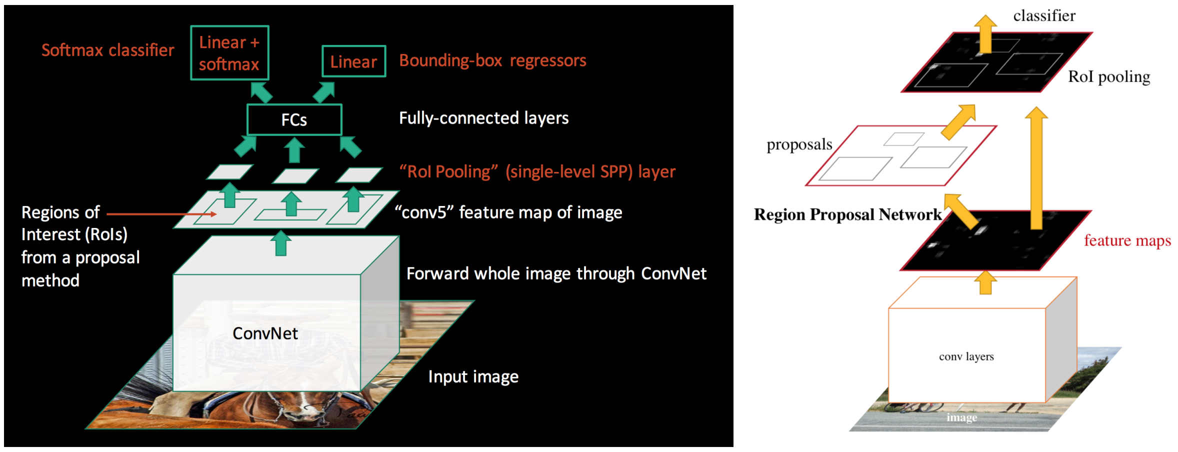
Figure 2-2: Illustration of the R-CNN architecture

[1]https://cdn-images-1.medium.com/max/800/1*k0b39xf47de-Bqr9KcK9hw.png

For the generated region proposals, the proposed regions are warped to a fixed squared size and passed on to a modified version of AlexNet. A **S**upport **V**ector **M**achine (SVM) is added to the last layer that performs the classification. The SVM determines whether the region contains an object, if so it also determines the type of object. The final step is an optimization step that re-sizes the bounding box to better match the boundaries of the object. This process is computationally expensive and is not yet capable of running in real time, iteration on this concept have eventually led to the development of Faster R-CNN [26].

In R-CNN the region proposal step generates 2000 potential **R**egions **o**f **I**nterest (RoIs), each of these proposals is converted into corresponding features maps using the AlexNet network, this is computationally very expensive. In Fast R-CNN [27] a break through was achieved by realizing that region proposals depend on the same features that are generated in the forward pass of the CNN. So in R-CNN the image is passed through the network once, generating the necessary features maps, using these feature maps the relevant RoIs can be generated. With the use of a **R**egion **o**f **I**nterest (RoI) pooling layer it is then possible to wrap the RoIs into a single layer. From this pooling layer the feature maps are fed into a fully-connected layer and split into two heads; One performs classification using a linear regression and softmax operation, and the other generating bounding boxes using a linear regression. See Figure 2-3A for an illustration of this network.

The latest iteration of the R-CNN framework is called faster R-CNN. Faster R-CNN addresses only the SS method that creates the region proposals. This part is the bottle neck in the fast R-CNN architecture, significantly effecting the efficiency. Instead of the SS method a second small CNN is used to generate RoIs, getting region proposals for little computational effort. This small network is also referred to as a **R**egion **P**roposal **N**etwork (RPN). The RPN is trained to generate RoIs from the feature maps in the last layer of the CNN, it is based on the MultiBox approach [28, 29]. The RPN outputs a set of rectangular object proposals, each with an objectness score. These proposals are then feed into the RoI pooling layer the same as in Fast R-CNN. See Figure 2-3B for an illustration of the faster R-CNN architecture.



A – Fast R-CNN **B – Faster R-CNN**

Figure 2-3: Illustrations of the fast and faster R-CNN architecture.[2]

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

The faster R-CNN framework uses a combined loss function of bounding box regression, bounding box confidence scoring, and object classification. This combined loss allows both networks to be trained simultaneously. Faster R-CNN also introduces anchor boxes for the first time. Rather than predicting each box directly, offsets are predicted for hand-picked priors. The reasoning behind bounding boxes was the for the bounding boxes of objects their exist certain common shapes and sizes, these common shapes and sizes are referred to as the anchor boxes. Predicting offsets to bounding boxes of several standard shapes and sizes helps simplify the problem for the network, it essentially gives the network several options to choose from that are similar in either shape or size to the ground truth.

2-1-2 Single forward pass architectures

The next step in object detection is combining the RPN and classification networks into a single CNN, where bounding boxes and class predictions are predicted from a single forward pass of the network. The first to achieve this was the **Y**ou **O**nly **L**ook **O**nce (YOLO) [9] architecture, achieving much faster detection rates. **S**ingle **S**hot **D**etection (SSD) builds on this principle, leverages the Faster R-CNN's RPN, using it to directly classify objects inside each prior box instead of just scoring the object confidence. It improves the diversity of prior boxes' resolutions by running the RPN on multiple convolutional layers at different depth levels. Figure 2-4 compares the architectures of both YOLO and SSD.

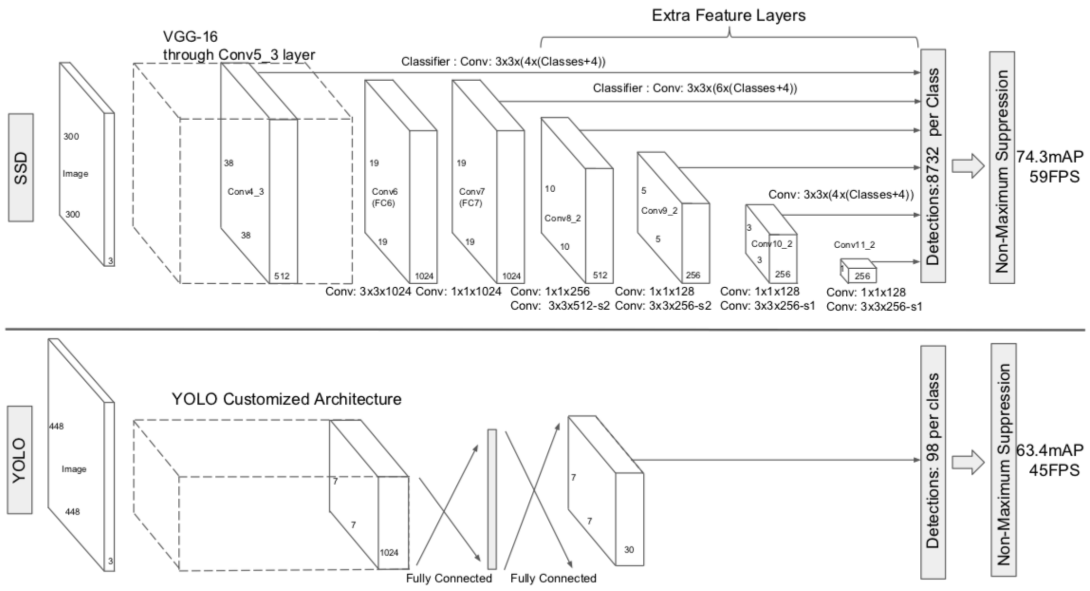


Figure 2-4: A comparison between two single shot detection models, SSD and YOLO. The SSD model adds several feature layers to the end of a base network, which predict the offsets to default boxes of different scales and aspect ratios and their associated confidences.

The YOLO network generates 1024 feature maps into a 7x7 grid, using fully connected layers this is restructured into a 7x7 grid with a depth of 30. Each cell in the grid now contains n predictions of bounding boxes and a confidence score for each bounding box. The confidence score gives an indication of the accuracy of the bounding box prediction and whether an object is present in the bounding box. Similarly YOLO generates a class probability score for all classes in each grid cell. A big limitation of this method is that it only generates one prediction per grid cell. This can result in smaller object not being detected that are close to other objects in the scene.

The SSD network has a good balance between efficiency and accuracy. The SSD architecture makes use of a modified version of the VGG network. SSD adds a small 3x3 sized convolutional layer to the outputs of several layers at different scales, using their feature maps to predict the bounding boxes and classify objects. Like the faster R-CNN framework SSD takes advantage of the anchor box method, only learning the off-set rather then directly determining the box dimensions. By predicting bounding boxes at different layers in the network it helps the network to better predict different scales of objects. SSD with a 300x300 input size significantly outperforms its 448x448 YOLO counterpart in accuracy on VOC2007 test while also improving the speed, see Table 2-1.

Finally after the initial YOLO model, a new version of the model was introduced that further increased the efficiency and accuracy, obtaining similar sate-of-the-art accuracy as SSD but being even more efficient. Unlike Faster R-CNN and SSD that generate bounding boxes for several scales, YOLOv2 takes a different approach. Instead of generating a prediction for several layers at different levels, YOLOv2 simply adds a pass-through layer to the network. The pass-through layer brings features from an earlier layer at a resolution of 26x26 to a layer further down the pipeline at a resolution of 13x13, see Section 3-2. Like SSD, YOLOv2 uses anchor boxes to help the network predict bounding boxes. Both SSD and YOLOv2 form the current state-of-the-art, achieving fast and accurate object detection. Table 2-1 shows the performance of these algorithms on the PASCAL VOC2012 dataset.

Framework	mAP - 2007	mAP - 2012	fps
Fast R-CNN	70.0	68.4	0.5
Faster R-CNN	76.4	70.4	7
YOLO	63.4	57.9	45
SSD-512	76.8	**74.9**	19
YOLOv2-544	**78.6**	73.4	40

Table 2-1: Object detection results on PASCAL VOC 2007 and 2012 dataset. YOLOv2 performs on par with state-of-the-art detectors like Faster R-CNN and SSD-512 while running 2-6x faster, All timing information is measured on a Geforce GTX Titan X.[13]

2-2 Object Detection and Pose estimation

With these advancements in object detection and classification and the introduction of PASCAL3D+ [16], a slow shift toward object detection and pose estimation using CNN has occurred. The problem of pose estimation of objects in an image however, was initially considered a **P**erspective **n**-point **P**roblem (PnP) [30], where the problem is seen as purely geometric. Some methods looked at correlation between keypoints in a 2D image and a 3D model representations of the object [31]. Other techniques focused on constructing a 3D model of the object and using this to find the 3D pose that best match the ground truth model [32]. All these cases involve simple datasets in controlled environments with single objects. With the introduction of more complex dataset containing multiple objects, new techniques using either extensions of DPM [5, 33, 34], parametric models [35, 36] or large 3D instances collections [37, 38] were introduced.

With the introduction of Pascal3D+ dataset, which extends Pascal **V**isual **O**bject **C**lasses (VOC) dataset by aligning a set of 3D CAD models for 12 rigid object classes, learning-based approaches became possible. CNN-based approaches, that until the availability of Pascal3D+ were limited to special cases such as faces [39] and smaller datasets [40], now begin to be applied to this problem at a larger scale.

2-2-1 Classification of the pose

Most techniques formulate the pose estimation problem as a classification problem rather than a regression problem. Instead of directly determining the pose, the pose is split into bins of equal ranges. i.e. the azimuth is split into 8 bins ($360/8 = 45°$ per bin), which can be viewed as 8 separate classes. This simplifies the problem but provides limited accuracy, increasing the number of bins also increases the computational complexity.

The work done in **V**iew**p**oint**s** & **K**ey**p**oint**s** (VpKps)[41] describe the same approaches as discussed above, treating pose estimation as a classification problem and using a two stage architecture. Using R-CNN framework to detect objects and suggest regions in which the objects can be found. These suggested regions are then used as input for the pose estimation network, which in this case is the VGG network [42] trained and tested using PASCAL3D+.

In RCNN+Alex[37] the same formulation is used as the earlier work of VpKps for pose estimation. However, by generating more than 2 million synthetic images with ground truth pose annotations the network is better suited at estimating viewpoint accuracy. Like VpKps the network is tested using PASCAL3D+. Both techniques use a disjoint architecture, showing slow operating speeds. As was mentioned in the previous section, to get faster operation speeds a single CNN is suggested.

2-2-2 State-of-the-art

With the recent development of single shot architectures YOLO and SSD a significant speed increase has been achieved. In SSD$_{POSE}$[43] the SSD pipeline is used as the basis. This architecture allows to jointly perform the detection and pose estimation in a single forward pas of the network reducing the required computational power. However, it still makes a discrete estimate of the object's pose, classifying the pose in bins rather than regressing the pose directly. The techniques that are considered the current state-of-the-art on Pascal3D+are are MT-CNN[44] and SSD$_{POSE}$. Both approaches show higher accuracy than previous approach with MT-CNN outperforming even SSD$_{POSE}$, but the SSD$_{POSE}$ approach is computationally much more efficient, capable of running in real-time.

Framework	AVP
VDPM[16]	12.1
DPM-VOC+VP[33]	13.6
RCNN+Alex[37]	19.8
VpKps[41]	31.1
MT-CNN[44]	**36.1**
SSD$_{POSE}$[43]	28.8

Table 2-2: Joint object detection and azimuth pose estimation results on PASCAL3D+ (24 bins AVP).[3]

2-3 Chapter Overview

With the improvements in hard-ware, deep CNNs and available training data, CNNs have grown to tackling more and more real-world problems. This also applies to vision related problems in robotics. Works like R-CNN, YOLO and SSD have shown promising results in object detection and are getting increasingly better at pose estimation. However there are some limitations when it comes to pose estimation, as these networks only look at azimuth angles, classify these angles rather then regress them and often are limited by their operating speed. However, YOLO and SSD show promising results as both models show that it is possible to have a good balance between accuracy and efficiency, running in real-time (>24 fps). These networks also show that it is possible to learn to detect multiple objects in an image and estimate their pose, showing that geometric information can be preserved in the feature maps in both models. These related techniques show that pose estimation including azimuth, elevation and distance should be possible using an efficient single shot CNN architecture. Below a list is given on what is possible with current techniques and what this research aims to improve on.

Current techniques

- **Object detection:** done using a bounding box.
- **Object classification:** classify each object detected.
- **Pose estimation:** only looking at the azimuth and structuring poses into bins, classifying which bin best represents the pose.
- **Real-time operation:** running at >24 fps.
- **Dataset:** able to train and test on PASCAL3D+ dataset.

[3]http://cvgl.stanford.edu/projects/pascal3d.html

Further improvements

- **Object detection:** using only objectness function to detect objects in the scene rather than using bounding boxes, limiting the number of parameters needed to be learned.
- **Pose estimation:** including elevation and distance into the pose definition and regressing the pose rather than just classifying the pose.
- **Performance:** increase the overall **A**verage **V**iewpoint **P**recision (AVP) of the pose estimation problem.

Chapter 3

Methodology

This chapter will address the design of an end-to-end trainable network that predicts a 3D pose in real-time, using a single shot 2D object detector. The network is designed to take a single RGB image as input and perform classification and 3D pose estimation of objects in the image. The network is a variant of the YOLO network with a modified output layer for pose estimation, this modification refers to a change in dimension of the output layer as explained in Section 3-2-1. In this chapter the model architecture will be explained and the choices made for each section of the model architecture will be addressed in further detail. The model architecture will be covered in three parts:

1. Pre-processing of PASCAL3D+ data
2. YOLO network architecture
3. Post-processing of network output

The main part of the architecture is the **C**onvolutional **N**eural **N**etwork (CNN), therefore the CNN is discussed first. The pre-processing step and the post-processing step are discussed later referring to parts discussed in the CNN section. The chapter will conclude with an explanation of the training procedure. However, first a quick introduction of the PASCAL3D+ dataset will be given.

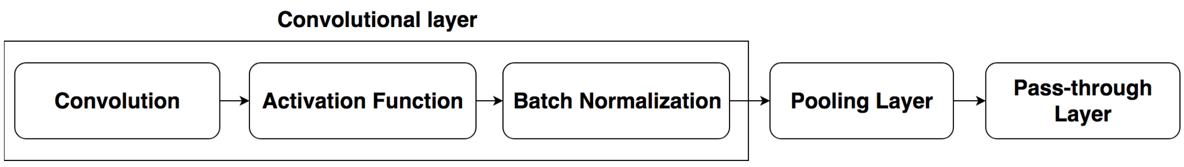
3-1 PASCAL3D+

Pascal3D+ dataset, re-lease 1.1 [16] is a dataset that provides 3D pose annotations for 12 common categories: aeroplane, bicycle, boat, bottle, bus, car, chair, dining-table, motorbike, sofa, train, and tv-monitor. The annotations are made for images selected from PASCAL VOC 2012 [14] and ImageNet [6]. For training both the ImageNet and Pascal VOC data is used and for evaluation only the validation dataset provided by VOC Pascal 2012 is used. The bottle category will be ignored as this is often the case in other works using PASCAL3D+. This is due to the bottle category not having a distinct reference for the azimuth angle to be determined, as the object is cylindrical in shape.

Other dataset like KITTI[45] and ICCV2015 Occluded Object Challenge[46, 47] are similar in that they also provide 3D pose annotations of objects. However, both dataset are task specific datasets, KITTI is a dataset that is specific to cars driving on a road and ICCV2015 Occluded Object Challenge only has one type of scene (table with a bunch of objects on it). PASCAL3D+ has a larger range of different environments making it harder and more applicable for different tasks.

3-2 YOLO network architecture

In this section the network architecture will be discussed together with the network inputs and outputs. The individual building blocks of the model will be reviewed with relevant basic theory of each building block. The building blocks consist of the following, in order of dependency:



3-2-1 Architecture

The architecture, as stated at the beginning of this chapter, is based on the YOLOv2 single shot object detector architecture that is modified to include pose estimation. The decision was between either the SSD or YOLOv2 architecture as both are capable of running in real-time and both show state-of-the-art accuracy's. In the end **Y**ou **O**nly **L**ook **O**nce (YOLO)v2 architecture was chosen because the ratio between accuracy and efficiency was better. YOLOv2 being 1.8% more accurate than SSD on the PASCAL VOC2007 dataset and only 1.5% worse on the PASCAL VOC2012 dataset, while running twice the amount of **f**rames **p**er **s**econd (fps) (see Table 2-1).

The network described in this paper, from this point on, will be referred to as the YOLO$_{POSE}$ architecture. The complete architecture is shown in Figure 3-1.

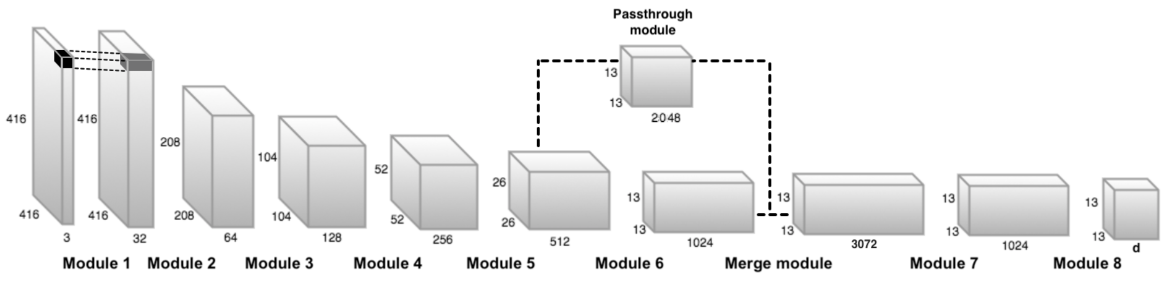


Figure 3-1: Illustration of the YOLO architecture – pass-through module passes information from a higher dimensional layer to a lower dimensional layer.

The architecture consists of 10 different modules, 8 modules with convolutional layers, a single pass-through module and a merge module. These modules fit in-between the boxes shown in the Figure 3-1, these boxes represent the dimensions of each input an output of the individual modules. Table 3-1 gives a more detailed representation of the different modules and their components.

Modules	Type	Filters	Size/Stride	Output
module 1	Convolution 1	32	3 x 3	416 x 416
module 2	Maxpool		2 x 2/2	208 x 208
	Convolution 2	64	3 x 3	208 x 208
module 3	Maxpool		2 x 2/2	104 x 104
	Convolution 3	128	3 x 3	104 x 104
	Convolution 4	64	1 x 1	104 x 104
	Convolution 5	128	3 x 3	104 x 104
module 4	Maxpool		2 x 2/2	52 x 52
	Convolution 6	256	3 x 3	52 x 52
	Convolution 7	128	1 x 1	52 x 52
	Convolution 8	256	3 x 3	52 x 52
module 5	Maxpool		2 x 2/2	26 x 26
	Convolution 9	512	3 x 3	26 x 26
	Convolution 10	256	1 x 1	26 x 26
	Convolution 11	512	3 x 3	26 x 26
	Convolution 12	256	1 x 1	26 x 26
	Convolution 13	512	3 x 3	26 x 26
Pass-through module	High Res.	2048		13 x 13
module 6	Maxpool		2 x 2/2	13 x 13
	Convolution 14	1024	3 x 3	13 x 13
	Convolution 15	512	1 x 1	13 x 13
	Convolution 16	1024	3 x 3	13 x 13
	Convolution 17	512	1 x 1	13 x 13
	Convolution 18	1024	3 x 3	13 x 13
	Convolution 19	1024	3 x 3	13 x 13
	Convolution 20	1024	3 x 3	13 x 13
Merge module	Concatinate	3072		13 x 13
module 7	Convolution 21	1024	3 x 3	13 x 13
module 8	Convolution 22	d	3 x 3	13 x 13

Table 3-1: CNN model information. Layer's input size, output size, number of Filters, kernel size and stride.

The table shows that there are 5 max-pooling operations with a stride of 2 in the network. These pooling layers determine the overall size reduction of the image, reducing the image by half with each pooling operation. Because there are 5 max-pooling layers the resulting image reduction equals 32 (2^5). The image goes from an input size of 416x416 to an output size of 13x13.

The size of the input image is determined by the following:

- **Has to be a multiple of 32** – (224, 256, 288, ...).
- **Spatial dimension of output has to be an odd number** – The center of a picture is often occupied by a large object. With an odd number grid there exists a single cell in the center, allowing the network to be more certain on the location of the object.
- **Number of objects that can be detected close to one another** – A higher spatial dimension of the output results in more objects being detected close to one another.
- **Computational complexity** – Spatial dimension of the input determines the number of weights in the network, higher dimension results in a higher complexity.

The first two points are fixed constraints on the input image size, the last two points are performance related. Increasing the spatial dimension of the output increases the accuracy of the network but subsequently also increases the complexity, decreasing the number of fps. Table 3-2 shows several image scales, comparing accuracy and fps between scales. Figure 3-2 shows the accuracy plotted against the fps, after image scale of 416 the accuracy increase starts leveling out, while the number of fps decreases at a continues rate. Therefore, the chosen input image size is set to 416x416 being an optimal ratio between efficiency and accuracy.

Framework	mAP	fps
YOLOv2-288	69.0	91
YOLOv2-352	73.7	81
YOLOv2-416	76.8	67
YOLOv2-480	77.8	59
YOLOv2-544	78.6	40

Table 3-2: YOLOv2 accuracy and efficiency results on PASCAL VOC2007 dataset for different image scales, all timing information is measured on a Geforce GTX Titan X.[13]

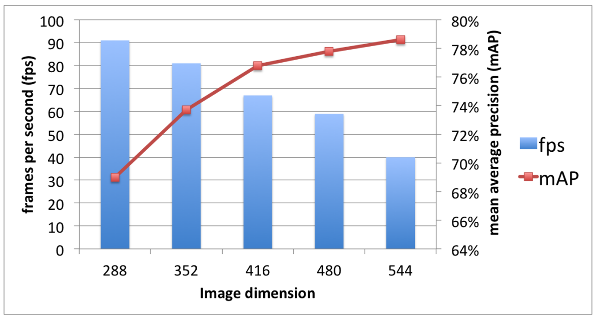


Figure 3-2: line plot of data in Table 3-2

Like VGG [42] the size of the kernel for each convolution layer is set to 3 x 3, they discovered that a network with a large depth of 16-19 weighted layers performs significantly better when only using small (3 x 3) kernels, rather than large kernels (>5). In a similar manner to GoogLeNet [48] layers with kernels size 1 x 1 are used between layers with kernel size 3 x 3 to reduce layer dimensions (number of feature maps) in the network. This results in more efficient and accurate features.

The output of the network is defined by a tensor of 13 x 13 x d. d is the number of predictions per grid cell (d $= n(P + V + C)$) where, n is the number of bins, C the number of confidence (objectness) values for the predictions, V the number of parameters defining the 3D pose and P the number of classes. In this work the azimuth data is divided in 4 bins ($n = 4$), the PASCAL3D+ dataset has 11 classes (excluding the bottle class as mentioned in section 3-1), there are 3 parameters related to the pose ($V = 3$; azimuth, elevation and distance) and there is 1 confidence prediction for the object pose ($C = 1$). Figure 3-3B illustrates the output structure of the network. In Figure 3-3A the original YOLOv2 network output is also visualized. The difference between the two outputs is that YOLOv2 was designed to work on the PASCAL VOC dataset which has 20 class categories and instead of estimating the pose, four bounding box parameters are estimated. In the network used in this research $S = 13$ and there are a total of 4 predictions per cell ($n = 4$).

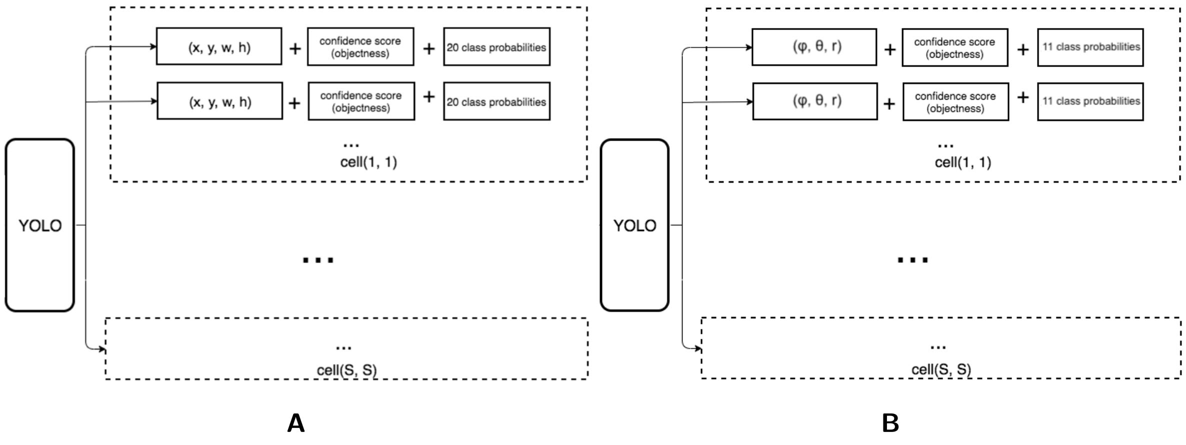


Figure 3-3: Output dimension structure of YOLO network.[1] A – is the output of the original YOLOv2 network. B – is the output of the modified YOLO$_{POSE}$ network.

As was shown in Table 3-1 there are several types of layers, each of these layers consists of specific building blocks. For the convolutional layer there is the convolution operation itself, an activation function and a batch normalization operation. For the max pooling layer it is just the pooling operation. The pass-through layer has a reorganization operation and a merging operation. The subsequent sections will discuss these building blocks respectively.

3-2-2 Convolutional layer

convolution operation

The convolution operation is the main component of a CNN, this is also where most of the computational costs lie. In the convolution operation learnable weights (a.k.a. features) are applied to the input of the convolutional layer to generate feature maps. The size of the features can be freely selected, but often is set to a size of 3x3. In the YOLO$_{POSE}$ architecture the convolution has 2D features not taking the depth into account. i.e. a 3D feature when applied to the input image would allow to generate a feature that incorporates all three RGB values, whereas 2D feature only look at one color value. However, 3D feature are computationally more expensive and often don't show better results. The reason for a convolution operation is to reduce the number of free parameters and improve generalization, 3D features introduce more free parameters making them less common in CNN. The 2D features are applied on the whole image in a similar manner as a sliding window going across the image. The convolution operation can mathematically be defined as an operation performed on two functions to produce a third, the equation is show in Eq. (3-1).

$$(\mathbf{K} * \mathbf{X})_{i,j} = \sum_m \sum_n \mathbf{X}_{i-m,j-n}\mathbf{K}_{m,n} \tag{3-1}$$

[1] https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

The convolution operation $*$ performs an operation on the kernel $\mathbf{K}$ (or feature) and on the layer input $\mathbf{X}$, the resulting output is defined as the feature maps. The input of the convolutional layer can either be the RGB images (only for the first layer) or feature maps generated in a previous layer. In the resulting function the terms m and n represent the location of the kernel and i and j represent the points within the kernel, Figure 3-4 illustrates this in detail for a single kernel.

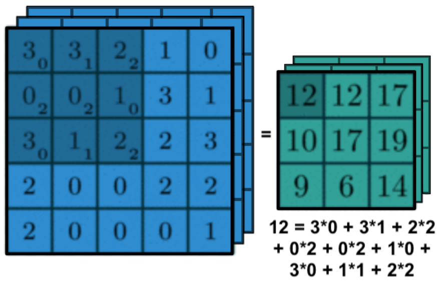


Figure 3-4: Illustration of the convolution operation, applying a feature to the input data **X** of size 3 x 3. The **output grid** is the resulting feature map after convolution, the depth is defined by the number of feature maps.

In the training process weights are adjusted after each propagation of the network. To limit the number of weights that need to be trained, a feature map consist of only one feature that is applied to all regions of the input instead of multiple features applied to different regions. The reasoning behind this is that if a feature is useful for a region i it will also be useful for a region j. For this reason the size of the output of the convolutional layer can be determined by four hyper-parameters:

- **Depth** corresponds to number of features being applied to the input.
- **Filter size** corresponds to the size of the feature.
- **Stride** is the shift of the feature over the input. With a stride of 1 the feature is shifted 1 pixel at a time. With a larger stride the output becomes smaller spatially (width and height) as more pixels are skipped.
- **Zero-padding** is padding the input with zero values around the border. Increasing the padding on the input also increases the size of the output spatially. This can be used to make the output size the same spatially as the input size, if stride is set to 1.

In Table 3-1 these hyper-parameters are specified together with the output size of each layer.

Activation Function

The activation function of a node (individual component within a layer) determines the activation range of the node, in other words it determines whether the node plays any role in predicting the class and pose of an object. Every activation function takes an input x and performs a fixed mathematical operation on it. There are several activation functions that can be used, each with its own properties. There are three standard activation functions often used in CNNs:

- **Sigmoid:** Generates a non-linear output between 0 and 1 from a given input x.

$$\phi(x) = \frac{1}{1 + \exp(-x)} \tag{3-2}$$

- **Tanh:** Generates a non linear output between -1 and 1 from a given input x.

$$\phi(x) = \tanh(x) \tag{3-3}$$

- **Rectified Linear Unit (ReLU):** Replaces negative input values with 0 and thus thresholds the input at 0.

$$\phi(x) = \max(0, x) \tag{3-4}$$

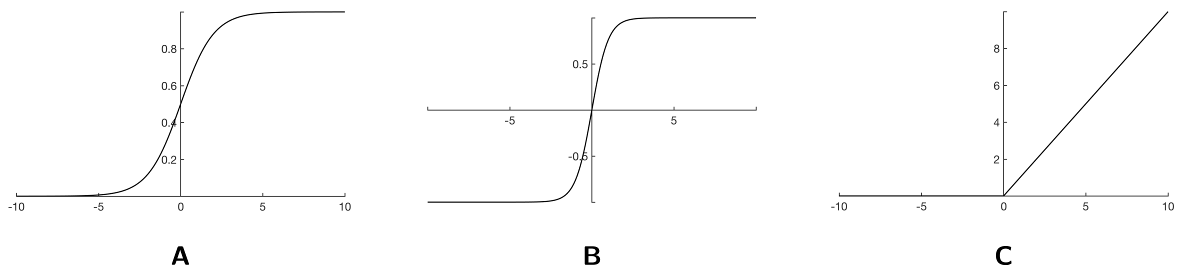


Figure 3-5: Graphs of three common activation functions. A – Sigmoid function. B – Tanh function. C – ReLU

In machine learning ReLU is the most common activation function, it is found that it significantly accelerates the convergence of **S**tochastic **G**radient **D**escent (SGD). Likewise it is computationally less complex than its counterparts. It is suggested that this is due to its linear and non-saturating structure [7]. Because ReLU implements a true zero cut off, neurons are actually cut off, unlike a non-linear activation where zero is never truly reached. This is similar to how brains of humans and animals work, neurons are neglecting if they don't play a role in the decision making [49]. In [50] they showed that this improved the overall accuracy of the network.

However, with ReLU functions there is a risk of neurons becoming permanently inactive. In some cases a large gradient can cause the ReLU to update the weights to a point ($\phi(x) = 0$) where it does not activate on any other data point again. Setting the learning rate lower can prevent this from happening, a learning rate that is too high can cause a number of neurons to become inactive. Another solution is to use a new type of ReLU, also known as the Leaky ReLU activation [51]. The difference being that instead of the activation function having a threshold at zero when $x < 0$, the Leaky ReLU has a small negative slope p.

$$\phi(x) = \begin{cases} x, & if\ x > 0 \\ px, & otherwise \end{cases} \tag{3-5}$$

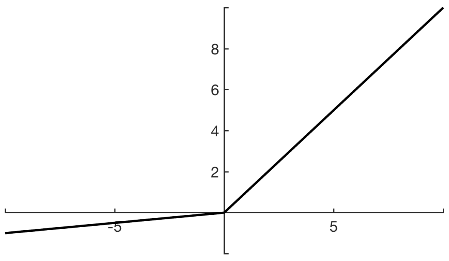


Figure 3-6: Function graph of Leaky ReLU activation function

In this case p can be a fixed value or it can be an adaptive value that is learning during training. Based on research done in [52] adaptive learning of p has not yet shown any consistent improvements. In the YOLO$_{POSE}$ network the p is set to 0.1 as this is a standard value for p.

Batch Normalization

To increase the stability of a neural network batch normalization can be used. Batch normalization essentially works by adding noise to the layer's activation in order to reduces the effects of overfitting[2]. As the name suggests, the activation is normalized by subtracting the batch mean and dividing it by the batch standard deviation.

The problem with batch normalization is that the activation output can changes with each batch, the weights of the next layer become hard to optimize if the normalization is based on the random selection of a batch. During training, SGD might be forced to undo the normalization of the previous layer, if it means that the loss is kept at a minimum. For this reason in the batch normalization step two trainable parameters are added, standard deviation γ and mean β. This allows the SGD algorithm to denormalize the batch normalization without having to change all the weights in the network, this helps preventing instability. The algorithm of the batch normalization is shown in Algorithm 1.

[2]Refers to a model that models the training data too well, negatively impacting the performance on new data

Algorithm 1: Batch Normalization, applied to activation x over a mini-batch [53]

```
input  : Values of $x$ over a mini-batch: $\beta = \{x_{1...m}\}$;
         Parameters to be learned: $\gamma, \beta$
output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$;

$\mu_\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$                          // mini-batch mean

$\sigma_\beta^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2$      // mini-batch variance

$\hat{x}_i \leftarrow \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}}$    // normalize

$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$        // scale and shift
```

3-2-3 Pooling layer

A standard operation after a convolution layer is to implement a pooling layer. This step reduces the spacial size of the feature map by replacing regions of the input with summations of the values within these regions. Similar to a convolutional layer, pooling is performed by sliding a kernel over the input and computing one single value (maximum, average or other) from the kernel (see Figure 3-7). The desired effect of pooling is to transform the representation of the feature map discarding irrelevant information while retaining important information.

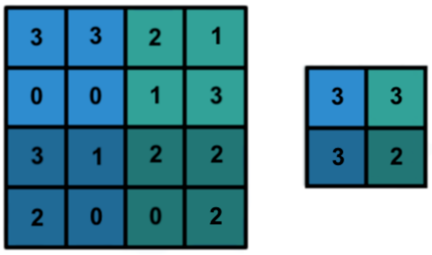


Figure 3-7: Illustration of max pooling operation, 2 x 2 kernel with stride = 2

The most common is max and average pooling [54] that generate as output the maximum or average value of the rectangular region. In the YOLO$_{POSE}$ network only the max pooling operation is used, this is similar to [13].

3-2-4 Pass-Through layer

In CNN, the resolution of the feature map decreases as the layers go deeper. A higher resolution feature map can be useful in predicting more precise poses as it gives more accurate spatial information. To incorporate a higher resolution feature map into the resolution of the output feature map, a pass-through layer is needed. Two channels cannot be combined if the size of the feature maps do not match, therefore the pass-through layer changes the size of the higher resolution feature map to match that of the output feature map. This is done by reorganizing the higher resolution feature map into multiple lower resolution feature maps. Figure 3-8 shows this process where a 4x4 feature map is cut into 4 separate 2x2 feature maps. These smaller resolution feature maps are then concatenated onto the channel of the lower resolution layer, see Figure 3-1 for the placement of the pass-through module and merge module.

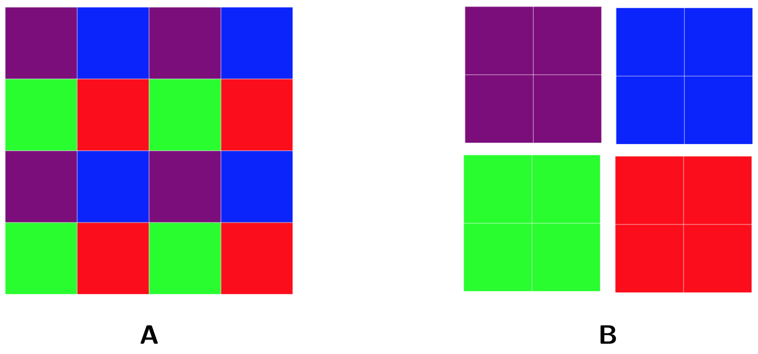


Figure 3-8: Illustration of how pass-through layer reorganizes the higher resolution channel to 4 lower resolution channels. A – 1 channel of a 4 x 4 feature map B – 4 channels of 2 x 2 feature maps.

3-3 Pre-processing

The first stage in the YOLO$_{POSE}$ architecture is the pre-processing stage, where data is modified to fit the CNN. The network is design to only work with square images of a predefined dimension, more specifically RGB images of 416 x 416 pixels (Section 3-2 explains further why the input has to be square and how the pixel size is chosen). The images need to be reshaped, as the data can range from images sized 1600 x 800 pixels to images size 230 x 430 pixels. To preserve information when re-sizing images, a geometric transform is applied instead of a crop. When cropping an image information is lost, objects can be cut out of the image frame unwillingly. Rather scaling and compressing an images keeps all the information in the image frame, Figure 3-9 illustrates this, where Figure 3-9-A shows the image when applying a Geometrical transform and Figure 3-9-B shows the image when a Square crop is applied. It shows that when cropping the image the bike can get cropped out, resulting in an image that does not have an objects to be detected.

The warp caused by the geometric transform (elongation or squeezing) can be seen as a side-effect. However, the network should be capable of learning around this problem, missing objects due to cropping is a much bigger issue. Another technique that can be used is to fill the sides of the image with black pixels to make the image square, see Figure 3-9-C. The problem with this is that network can learn to associate the the black pixels as part of the image and generate feature that are not relevant for global pose estimation. Because of these issues the chosen technique is to Geometrically transform the images, this is the same technique used by the original YOLO architecture.

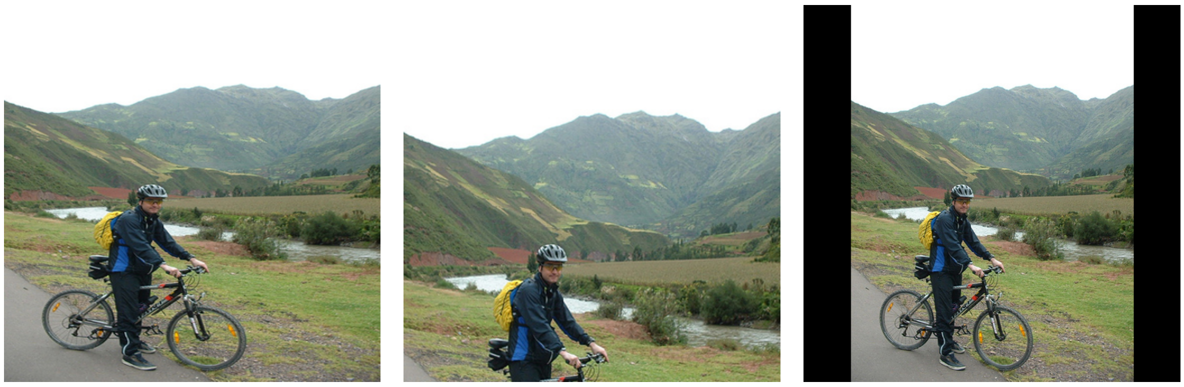


Figure 3-9: A – Geometrical transform of image to 416 x 416 px. B – Square crop of image re-sized to 416 x 416 px. C – Pixel filling to square image.

In the pre-processing stage images are not only re-sized to fit the network, data is also augmented to increase the amount of data available for training. The PASCAL3D+ dataset provides only a small amount of data, with a deep **N**eural **N**etworks (NNs) requiring a significant amount of data to properly train all weights without causing overfitting. The images are augmented in three different ways (*random crops*, *horizontal reflection* and *color distortion*) and are randomly generated for each Epoch[3].

3-3-1 Random Crop

Cropping an image is similar to zooming in on a specific region of the image. It increases the number of viewpoints for each image by augmenting a change in the distance from the camera to the objects. Because cropping an image changes the image frame and the distance of the objects it is necessary to change the annotations accordingly, see Figure 3-10. For the bounding box this is done using equations in Eq. (3-6).

$$\begin{aligned}
\hat{C}_x &= \frac{C_x - x_{shift}}{S} \\
\hat{C}_y &= \frac{C_y - y_{shift}}{S} \\
\hat{w}_{box} &= \frac{w_{box}}{S} \\
\hat{h}_{box} &= \frac{h_{box}}{S},
\end{aligned} \tag{3-6}$$

[3]A full pass through of the the entire training set

Where, C_X and C_y are x- and y-coordinates of the center point of the bounding box, x_{shift} and x_{shift} are shifts in the x and y direction of the cropped region, S is the scale of the cropped region ($\frac{\text{SIZE}_{original}}{\text{SIZE}_{crop}}$) and w_{box} and h_{box} are respectively the width and height of the bounding box. See Figure 3-10-A for an illustration of these parameters.

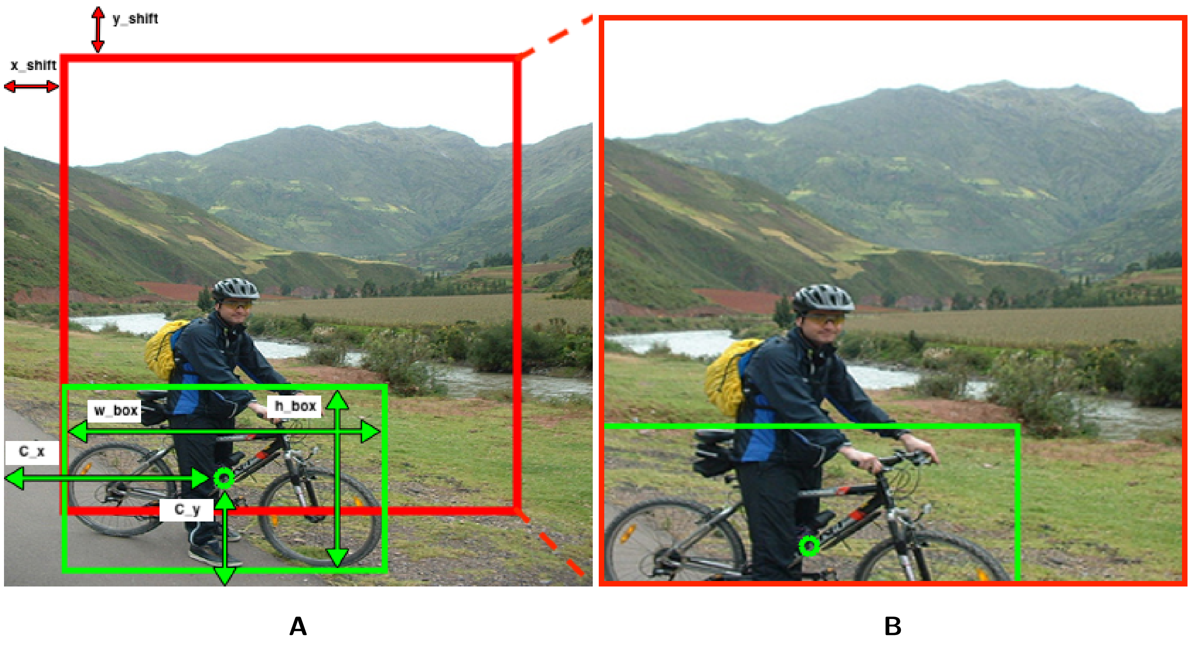


Figure 3-10: Illustration of random crop with the **green** box representing the bounding box of the object and **red** box the cropped region. A – Original image before crop, re-sized to 416 x 416 px. Illustrating the x_{shift}, y_{shift}, x_{crop} and y_{crop} parameters used in Eq. (3-6) B – Image after random crop, re-sized to 416 x 416 px.

To transform from the annotated distance before the random crop to the distance after the crop we look at the pinhole model shown in Figure 3-11.

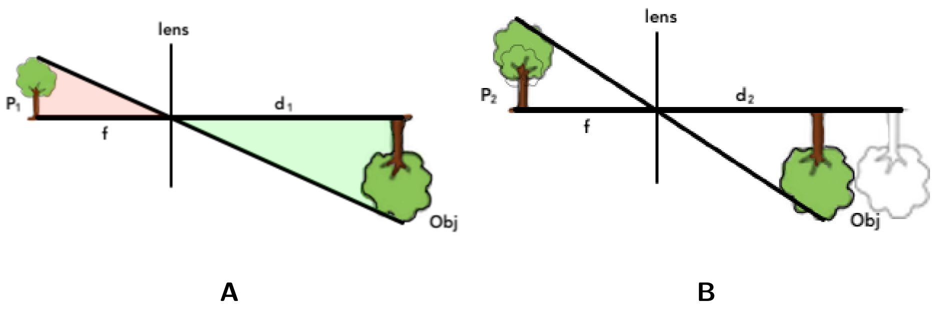


Figure 3-11: Pinhole model, where *f* is the focal length, *Obj* is the object size, p_i is the projection (or image frame) and d_i is the distance from camera to object. A – before crop B – after crop, notice how object in the image frame becomes larger and distance smaller.

Figure 3-11-A shows that there are two similar triangles of different scales, the triangle **before** and **after** the *lens*. The correlation between these triangles is formulated in Eq. (3-7).

$$\frac{P_i}{f} = \frac{Obj}{d_i} \tag{3-7}$$

Figure 3-11-B shows that cropping the image only changes the projection size P and the distance *d*. Object size *Obj* and focal length f stay the same between crops. For the original image we know the size of the image frame P_1, the focal length (*f=1* as used in PASCAL3D+ dataset [16]) and the distance (d_1). The only unknown is the object size *Obj*, which can be defined by Eq. (3-8).

$$Obj = P_1 \cdot d_1 \tag{3-8}$$

If we combine Eq. (3-7) and Eq. (3-8) we can calculate the distance of the object after the random crop, see Eq. (3-9).

$$d_2 = d_1 \cdot \frac{P_1}{P_2} \tag{3-9}$$

d_2 can be described as the distance before the random crop d_1 times the scale of the crop ($\frac{P_1}{P_2}$). During the random crop operation a constraint is placed on the scale of the crop and the x_{shift} and y_{shift} to prevent objects being cut out of the image frame. This is done by limiting the size of the crop to 0.7 of the original size and limiting the x_{shift} and y_{shift} to 0.3 of the width and high of the original image frame. A final check is then performed to see if objects are cut out of the frame when cropped, if so the crop is discarded and the original image is kept.

3-3-2 Horizontal Reflection

Horizontal reflection is the mirroring of the image on the vertical axis. It increases the number of viewpoints for each image by augmenting a change in the azimuth angle, see Figure 3-12.

With horizontal reflection only the x-coordinate of the center point (C_x) of the bounding box and the *azimuth* change in the annotation of the image. For the C_x parameter Eq. (3-10) is used.

$$\hat{C}_x = I_{width} - C_x, \tag{3-10}$$

where the new x-coordinate of the center point ($\hat{C}_x$) is the original width of the image I_{width} minus the old x-coordinate of the center point (C_x)

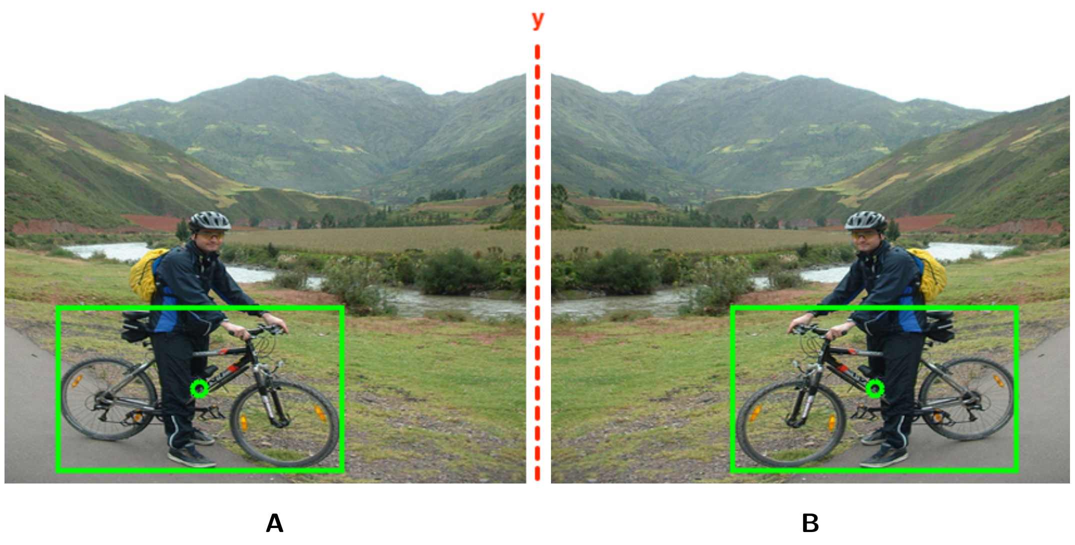


Figure 3-12: A – Original image before horizontal reflection, re-sized to 416 x 416 px. B – Image after horizontal reflection, re-sized to 416 x 416 px.

For the *azimuth* parameter we take a look at the spherical coordinate system as used in the PASCAL3D+ dataset for pose annotation. Figure 3-13 shows an illustration of the spherical coordinate system.

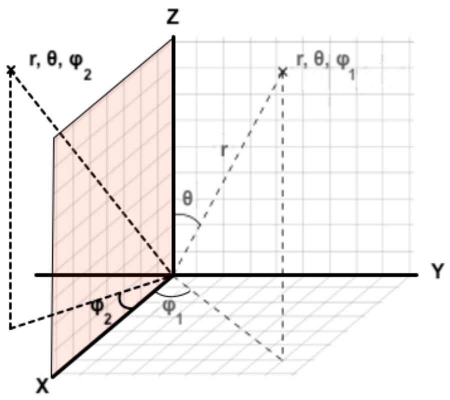


Figure 3-13: Spherical coordinate system. Illustrating the effect of horizontal reflection on the coordinate system. The **red** plane illustrates the reflection plane and r, θ, φ respectively are the *distance*, *elevation* and *azimuth*

.

Horizontal reflection can best be described as the mirroring of an image over the vertical axis, as seen in Figure 3-12. In the spherical coordinate system the mirroring takes place over a 2D plane in the x and z axis. This causes the azimuth to change direction, going from positive to negative. Therefore the new *azimuth* is simply the negative of the old *azimuth* before reflection, $\hat{\varphi} = -\varphi$.

3-3-3 Color Distortion

Color distortion augments changes in image brightness, color representation and color intensity to increase the number of unique images. The augmentation is performed on the brightness, contrast, saturation and hue of the image. Where,

- **Brightness**: The addition of a random offset to the intensity of the image ($\alpha = 1$ and $\beta = Random(-32, 32)$ in Eq. (3-11)).
- **Contrast**: The multiplication of the intensity of the image by a random scale ($\alpha = Random(0.5, 1.5)$ and $\beta = 0$ in Eq. (3-11)).

$$I_{RGB} = I_{RGB} \cdot \alpha + \beta \tag{3-11}$$

- **Hue**: The addition of a random offset to the hue of the image, where the image is first changed from the RGB to the HSV[4] color model ($\alpha = 1$ and $\beta = Random(-18, 18)$ in Eq. (3-12)).

$$H = H \cdot \alpha + \beta \mod 180 \tag{3-12}$$

- **Saturation**: The multiplication of the saturation of the image by a random scale, using the HSV color model ($\alpha = Random(0.5, 1.5)$ and $\beta = 0$ in Eq. (3-13)).

$$S = S \cdot \alpha + \beta \tag{3-13}$$

Figure 3-14 illustrates an example of a color distortion on a image from the PASCAL3D+ dataset.

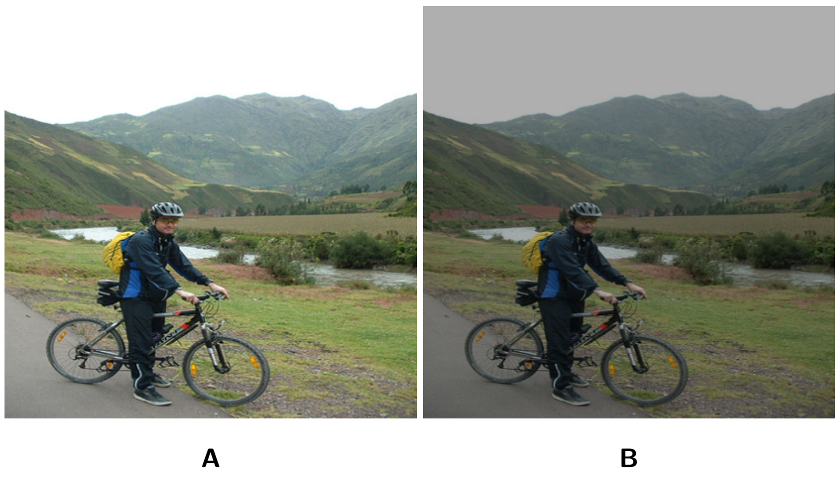
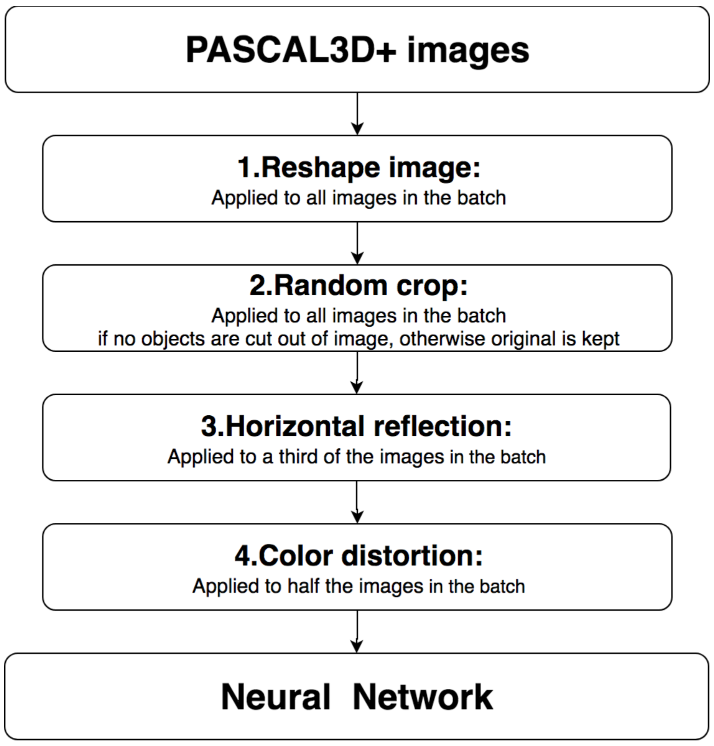


Figure 3-14: A – Original image before color distortion, re-sized to 416 x 416 px. B – Image after color distortion, re-sized to 416 x 416 px.

[4]Alternative representation of the RGB color model

Overview

The steps taken in the pre-processing stage are sequenced as followed:


PASCAL3D+ images
1.Reshape image:
Applied to all images in the batch
2.Random crop:
Applied to all images in the batch
if no objects are cut out of image, otherwise original is kept
3.Horizontal reflection:
Applied to a third of the images in the batch
4.Color distortion:
Applied to half the images in the batch
Neural Network

3-4 Post-processing

The network outputs a tensor of 13 x 13 x 60 as was mention in Section 3-2 of this chapter, depth of the output tensor is determined by $n(P + V + C)$. This tensor is converted into a tensor of size 13 x 13 x n x $(P+V+C)$ that can be used in the post-processing stage. Each cell in the 13 x 13 grid has an array of four separate predictions for Pose, Class and Confidence. These four predictions fall into four separate bins that are classified by the azimuth angle. The reason for this is to help the network with estimating the pose. By classifying the azimuth within four bins the algorithm can first classify within what range the azimuth lies and can then regress the pose for these smaller ranges. See Figure 3-15 for an illustration of the output of the network, similar to Figure 3-3 only differently illustrated.

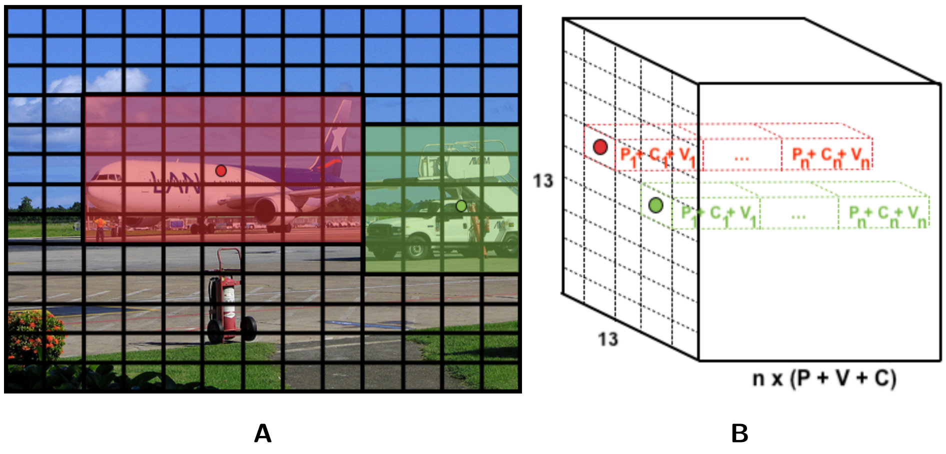


Figure 3-15: Illustration of the output of the network. A – How the output corresponds to the input image, each cell has a set of predictions for that region, red and green represent the individual objects and their center points. B – The model is trained to focus on the prediction of the grid cell where the center point lies, with the prediction (class probability P, pose parameters V and pose confidence C) structured in n number of bins.

For each image the output includes 676 (13 x 13 x 4) predictions concerning the object class, the confidence and the 3 pose parameters (azimuth, elevation and distance). It is therefore necessary to prune the predictions that are incorrect or are not relevant. In the original algorithm, [13] uses a cut-off value for the class probability P and bounding box confidence C to prune the incorrect predictions and uses **I**ntersection **O**ver **U**nion (IoU) to prune predictions that have a certain percentage of overlap.

Like the original, class probability and confidence is used to prune part of the predictions. however, unlike in YOLOv2 the confidence is normalized first before using a cut-off, this ensures that the cut-off better matches the confidence values generated for each image. When the confidence is not normalized some images might not reach the needed confidence threshold, resulting in all predictions being pruned. Normalizing the confidence ensure that for each image at least 1 prediction is generated.

Because YOLO$_{POSE}$ does not predict the bounding boxes the IoU can not be used to prune predictions that are similar. Instead of using the IoU to prune overlapping boxes the pose similarity is used. This simply looks at adjacent cells to determine if there are pose predictions that are similar. If the predictions fall within a certain range of each other, the prediction with the highest confidence is kept and the ones that are lower are pruned. This helps prevent multiple correct answers being used for one object. It is similar to the **N**on-**M**aximum **S**uppression (NMS) technique used in the YOLOv2 architecture. The checklist used is as followed:

- Prune everything with a class probability $P < 0.5$.
- Normalize the confidence (objectness).
- Prune everything with a normalized confidence (objectness) $C < 0.5$.
- Determine if predictions in adjacent cells fall within 10% of each other.
- Prune everything that falls within 10%, only keep the prediction with the highest confidence.

3-5 Training procedure

In this section the training procedure is discussed that is used to train the network to perform object classification and pose estimation. Other training related issues like cost function, confidence function, weight initialization and hyper parameters will also be included in the discussion.

3-5-1 Weight initialization

As stated before, the CNN network has to optimize its weights that form the kernels in convolutional layers. Before the first training iteration, these weights and biases have to be initialized. The choice of initialization strategy can determine the convergence of the training algorithm, how fast it converges and how accurately it converges. It also has an effect on the network's ability to generalize the data.

The common goal of weights initialization is to set them in a way that each neuron produces a different activation. This motivates to initialize the weights in some random way depending on the activation function used for nonlinearity. However transfer learning has shown promising results and significantly cuts the training time. In transfer learning the weights of a pre-trained network are used as the initialization weights of the new network. Or in this case the weights of the trained YOLOv2 network are used as the initialization weights. The weights are trained on the PASCAL VOC 2012 dataset [14] used for object detection and classification and transferred to the YOLO_{POSE} network. The network architectures are similar with an exception of the last layer, thus only requiring the last layer of weights to be randomly set. For ReLU activations modified "Xavier" initialization [55] has been proved to be a good initialization decision in [56]. The randomization is based on a zero mean normal distribution with a standard deviation of $\frac{2}{\sqrt{n}}$, where n is the number of connections of response from the previous layer.

3-5-2 Weight Learning Algorithm

During training of the network multiple images are run through the network simultaneously. An example of such a batch training method is the limited memory **B**royden, **F**letcher, **G**oldfarb, and **S**hanno (BFGS). The limited memory BFGS makes use of the full training set to compute the next update of parameters at each iteration. This method has the ability to converge very well to local optima and is relatively straight forward requiring little tuning as it has few hyper-parameters that need to be tuned. The downside of using the full training set to compute the next update is the computational costs and the memory usage it requires. In SGD this issues is addressed by following the negative gradient of the objective after seeing only a small batch of training examples. The advantage of using SGD is that it only uses small amounts of data rather than an entire training set. Using a batch of training examples in SGD rather than using a single data point is beneficial for the training process as well. Using multiple training examples helps reduce variance in the parameter update, which can lead to a more stable convergence. Likewise this allows to take advantage of highly optimized matrix operations. With SGD the cost of backpropagation can be reduced while still maintaining fast convergence.

In the standard gradient descent algorithm the parameters θ of the objective $J(\theta)$ are updated as followed,

$$\theta = \theta - \alpha \nabla_\theta E[J(\theta)] \tag{3-14}$$

In the standard gradient descent the cost and gradient are still being approximated over the full training dataset. SGD just ignores the expectation (E) in the update and computes the gradient of the parameters using a small batch of training examples. The new update is given by,

$$\theta = \theta - \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)}) \tag{3-15}$$

where $(x^{(i)}, y^{(i)})$ represents a pair from the training set and α represents the learning rate.

Tuning the learning rate can be difficult, a fixed learning rate can be used or a learning schedule can be used that changes with each epoch. Generally a method that works well is choosing a small learning rate that gives stable convergence and then slowly decrease the learning rate by half as convergence slows down. Another approach is to perform an evaluation after each epoch and determine the change in the objective, if the change in objective is below a predetermined threshold the learning rate is decreased. These are relatively common and simple approaches but there are more complex approaches that can also be used, like using a backtracking line search to find the optimal update. Some approaches might be better than others theoretically, this does not mean that they are better for all objectives. Testing is still the best way to determine which method works best.

The chosen algorithm in this research is a variant of the standard SGD implementing a momentum update [57]. The momentum update helps to better find optima in deeper architectures. When an architecture is deeper it often causes the objective to have the shape of a long shallow ravine leading to an optimum with steep walls on the sides. In standard SGD the gradient will point in the direction of the steep walls oscillating at a fixed point in the ravine, not moving along the ravine to the optima. This causes an initial quick convergence, but then slows down resulting in a long convergence time towards the optima. The Momentum update helps to push the objective more quickly along the shallow ravine by giving it, as the name states, a momentum. The update is represented by the following equation,

$$\begin{aligned} v &= \gamma v + \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)}) \\ \theta &= \theta - v \end{aligned} \tag{3-16}$$

Where v is the current velocity vector, θ the parameter vector, α the learning rate α and $\gamma \in (0, 1]$ determines the number of iterations for which the previous gradients are included into the momentum update. It is common to initially set γ to 0.5 and then when learning stabilizes increase it further to 0.9 or higher. In momentum SGD the gradient is generally larger than in standard SGD, requiring the learning rate α to be smaller.

3-5-3 Loss Function

For a SGD algorithm to work a cost (a.k.a. loss) function is required that determines the error of the prediction. The cost function in a supervised training scheme is the error of the prediction compared to the ground truth. Eq. (3-17) shows the complete cost function used to train the network.

$$L_{total}(I) = L_{pose}(I) + L_{conf}(I) + L_{prob}(I) \tag{3-17}$$

The cost function is composed of the pose error L_{pose}, pose confidence error L_{conf} and the class probability error L_{prob}. These errors are determined by the following equations:

$$\begin{aligned} L_{pose}(I) &= \lambda_{pose}^{object} \sum_{i}^{S^2} \sum_{j}^{n} \mathbb{1}_{ij}^{object} \Big(1.5 \cdot (\varphi_{ij} - \hat{\varphi}_{ij})^2 + (\theta_{ij} - \hat{\theta}_{ij})^2 + (r_{ij} - \hat{r}_{ij})^2 \Big) \\ L_{pose}(I) &= \lambda_{pose}^{no-object} \sum_{i}^{S^2} \sum_{j}^{n} \mathbb{1}_{ij}^{no-object} \Big(4 \cdot (0 - \hat{\varphi}_{ij})^2 + (0 - \hat{\theta}_{ij})^2 + (0 - \hat{r}_{ij})^2 \Big) \end{aligned} \tag{3-18}$$

$$\begin{aligned} L_{conf}(I) &= \lambda_{conf}^{object} \sum_{i}^{S^2} \sum_{j}^{n} \mathbb{1}_{ij}^{object} (c_{ij}(\mathbf{x}) - \hat{c}_{ij})^2 \\ L_{conf}(I) &= \sum_{i}^{S^2} \sum_{j}^{n} \lambda_{conf_{ij}}^{no-object} \mathbb{1}_{ij}^{no-object} (0 - \hat{c}_{ij})^2 \end{aligned} \tag{3-19}$$

$$L_{prob}(I) = \sum_{i}^{S^2} \sum_{j}^{n} \mathbb{1}_{ij}^{object} (p_{ij}(c) - \hat{p}_{ij}(c))^2 \tag{3-20}$$

These equations are summations of all the cells in the 13 x 13 ($S = 13$) network output, and all the predictions made for each cell ($n = 4$). The terms $\mathbb{1}_{ij}^{object}$ and $\mathbb{1}_{ij}^{no-object}$ denote whether the objects center is present in cell i and which bin j is responsible for the prediction. The terms λ_{pose}^{object}, $\lambda_{pose}^{no-object}$, λ_{conf}^{object} and $\lambda_{conf}^{no-object}$ are parameters that scale the importance of the error. The confidence is less important then the pose prediction, similarly cells containing no object are less important then cells containing objects. Finally the azimuth (φ) is more important than the elevation (θ) and distance (r) parameters because the azimuth also determines which bin is selected, therefore the loss is multiplied by 1.5. These are hyper-parameters that can be manually tuned to increase performance. The parameters are set to $\lambda_{pose}^{object} = 100$, $\lambda_{pose}^{no-object} = 0.5$, and $\lambda_{conf}^{object} = 10$. For $\lambda_{conf_{ij}}^{no-object}$ this is dependent on the pose predictions,

$$\lambda_{conf}^{no-object} = \begin{cases} 0, & if\ \varphi < 15°, \theta < 7° \textbf{ and } r < 4 \\ 0.1, & otherwise \end{cases} \tag{3-21}$$

The remaining terms are φ_{ij}, θ_{ij}, r_{ij}, c_{ij} and p_{ij} that respectively represent the azimuth, elevation, distance, pose confidence and class probability, the $\hat{}$ represents the ground truth value. Finally the confidence function $c_{ij}(\mathbf{x})$ is based on the function from Figure 3-16.

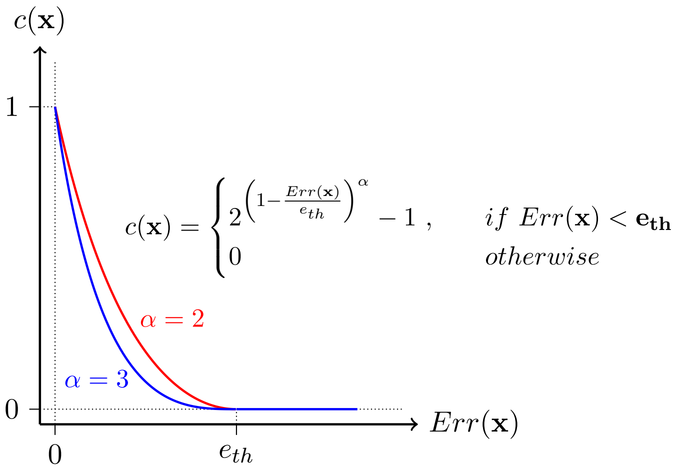


Figure 3-16: Confidence $c(\mathbf{x})$ as a function of the error $Err(\mathbf{x})$ between the prediction and the ground truth. The function has a cut-off value e_{th} and a steepness parameter α that determines the steepness of the slope

The error $Er(x)$ is defined as the difference between the ground truth and the model prediction. To achieve precise localization of the correct prediction in the grid, a sharp exponential function with a cut-off value e_{th} is chosen instead of a monotonically decreasing linear function. The sharpness of the exponential function is defined by the parameter α. The confidence function is applied to all control points and the mean of these values is then assigned as the confidence of the prediction.

Overview

The learning process consists of several components; the learning algorithm itself, the weight initialization and the loss function. Each of these components fulfills a task, Table 3-3 shows each component, their specific task and the method used in this research.

Component	Function/Task	Method
Weight initialization	Helps the training algorithm with a starting gradient to optimize from	Transfer learning
Loss function	Defines the problem, helps the learning algorithm to determine the effects of a weight change	three part loss function (pose, confidence and probability loss)
Training algorithm	Training of the network by learning the weights in each layer	Momentum SGD

Table 3-3: Components of the training procedure used in YOLO$_{POSE}$

In [58] it was found that both the initialization and the momentum are crucial in SGD. Due to a poorly initialized network not being able to be trained with momentum. Similarly a well-initialized network does not perform as well when the momentum is not included or poorly tuned. The momentum SGD shows fast convergence and robustness to different tasks and datasets, thus this is the algorithm chosen in this research.

3-6 Chapter Overview

The framework discussed in this chapter builds on the existing YOLOv2 architecture, re-designing it to perform pose estimation rather than object detection using bounding boxes. The relevant components are summarized by the following:

- **Output layer:** Restructured to include the pose parameters, discarding the bounding box parameters from YOLOv2 network.
- **Pre-processing:**
  - Re-sizing images: Using geometrical transform to re-size image to 416x416 pixels.
  - Data augmentation: Random crop (ensure that no objects are lost), horizontal flipping and random color distortion.
- **Post-processing:**
  - Normalize the confidence for each image
  - Prune everything with a class probability $P < 0.5$
  - Prune everything with a confidence $C < 0.5$
  - Prune everything that falls within 10% of the pose prediction of an adjacent cell, only keeping the prediction with the highest confidence.
- **Training procedure:** Using momentum SGD, with transfer learning to initialize the weights.

Chapter 4

Implementation and Results

In this chapter tuning parameters, training choices and results are discussed. First, the different hyper-parameters are discussed. Second, the evaluation technique used to measure the accuracy of the network is explained. Finally the results are presented, analyzed, and compared with current state-of-the-art pose estimation techniques.

All experiments in this chapter are performed on Microsoft Azure cloud service. The virtual machine runs on the NV-6 instance[1], this instance has 6 cores (E5-2690v3) and a single Tesla M60 GPU.

4-1 Tuning the Network

The network weights are initialized using the weights fine tuned for the YOLOv2 architecture on the object detection task described in [14]. These weights are generated by training first on the ImageNet dataset and then on PASCAL VOC 2007+2012 object detection dataset. The training was performed using the momentum **S**tochastic **G**radient **D**escent (SGD) technique with a momentum of 0.9 and a weight decay of 0.0005 [13].

Each batch during the training process is set to 25 images. This is the maximum amount of images that can be stored on the GPU at one time without exceeding the internal memory. The maximum amount of images per batch is used to help the network learn the correct weights. A higher number of images and thus higher diversity of scenes and objects helps the network to generalize the weights better for the different classes and poses defined in PASCAL3D+.

The learning rate is kept at 0.001, incrementally increasing or decreasing the learning rate did not show any improvements. Increasing the learning rate resulted in the loss not converging, and decreasing the learning rate did not increases the accuracy only prolonging the training time. The momentum is set to 0.9 as this is the most common value used in momentum SGD. Smaller values tend to cause fluctuations, averaging over a smaller number of examples

[1] https://azure.microsoft.com/en-us/blog/azure-n-series-general-availability-on-december-1/

means it can't find a clear path in the data, becoming just as noisy as the data itself. Larger values tend to smooth out the data curve to much causing it to not following the same path as the data.

Finally the number of epochs needed to properly train the network depends on the hyper parameters chosen, the type of data fed to the network and the type of task given to the network, for most experiments 80 epochs is enough to see a stagnation in the learning process. However, in some cases the number of epochs need to be higher due to the accuracy still increasing. Table 4-1 gives an overview of the hyper-parameters used during the experiments.

Learning rate	0.001
Momentum	0.9
Batch size	25
Epochs	80

Table 4-1: Hyper-parameters selected

4-2 Evaluation

Detection evaluation

When evaluating models for binary classification on a given dataset of positive and negative samples, usually four types of data are defined: **T**rue **P**ositive (TP), **T**rue **N**egative (TN), **F**alse **P**ositive (FP) and **F**alse **N**egative (FN), whereas true and false refer to whether the positives or negatives are correctly classified by the model. For multi-class classification, the negative samples of one category refers to all the other classes. Eq. (4-1) and Eq. (4-2) define the precision and recall metrics.

$$P = \frac{TP}{TP + FP} \tag{4-1}$$

$$R = \frac{TP}{TP + FN} \tag{4-2}$$

Basically, Precision P represents the fraction of correctly predicted instances among all retrieved instances. It is an measurement of the correctness of the model output. Recall R represents the fraction of correctly predicted instances among all positive instances which should be retrieved. It measures the instance retrieving ability of the model. The TP in this case is defined by the pose error being less than 4%, a correctly defined class and correctly estimated relative position of the object in the image. Multiple detections of the same object are considered FP. The **A**verage **P**recision (AP) is defined as precision averaged across all values of recall between 0 and 1:

$$AP = \int_0^1 p(r)dr \tag{4-3}$$

where r is recall and $p(r)$ is the precision at recall r. The **I**nterpolated **A**verage **P**recision (IAP) technique is used as the precision measure and is defined by Eq. (4-4).

$$AP = \sum_{k=1}^{N} \max_{\tilde{k} \geq k} P(\tilde{k}) \Delta r(k) \tag{4-4}$$

Instead of using the precision that was actually observed at cutoff k similar to the traditional 11-point approach [59], the interpolated average precision uses the maximum precision observed across all cutoffs with higher recall. This is a technique that is used in the PASCAL VOC challenge to determine the accuracy of bounding box's around objects.

Pose Evaluation

Average **V**iewpoint **P**recision (AVP) includes the pose estimation into the precision measure. A TP is then defined by the pose similarity of the prediction (φ, θ, r) to the ground truth $(\hat{\varphi}, \hat{\theta}, \hat{r})$. The pose similarity is calculated by determining the error of the azimuth φ, elevation θ and distance r, see Eq. (4-5).

$$\Delta_\varphi = |\hat{\varphi} - \varphi|,\ \Delta_\theta = |\hat{\theta} - \theta|,\ \Delta_r = |\hat{r} - r| \tag{4-5}$$

The error has to fall within $\varphi < 15°$, $\theta < 7°$ and $r < 4$ to be considered a TP.

4-3 Results

In this section the training choices and results will be discussed. For the training choices the effects on the accuracy of the network will be analyzed and discussed to determine which techniques work best. Similarly a new post-processing step will be implemented to determine the effect. Finally the results will be compared to current state-of-the-art techniques. These techniques will be compared using the metrics used in [16].

4-3-1 Training choices

During the training operation different experiments were conducted to determine which technique resulted in accuracy gains and which did not. The following experiments were conducted:

- **Baseline**: Generating a baseline for the spherical pose representation to tune the hyperparameters and to determine which modifications in subsequent experiments yield the best results.
- **Data clustering**: Clustering the data in sequential ranges for elevation, distance and azimuth. Also a different structure of the bins is considered, in order to determine which data clustering technique best helps the network determine the correct pose.
- **Free selection**: Allowing the network to learn the best ground truth reference point for each object. To determine if this free selection helps the network to pick regions that better represent the pose and class of an object.
- **Data augmentation**: Generating more data by augmenting existing data. To determine if this limits overfitting in the network.
- **Data selection**: Testing whether specifying how data is introduced effects the learning process, to determine if this helps the network to better understand its objective.

For the first experiments the results generated are evaluated using the AVP metric only concerning the azimuth angle, not all three parameters (azimuth, elevation and distance). This is to determine when a similar result is obtained compared to the current state-of-the-art, as these techniques only look at the azimuth angle. In the final experiment (data selection, **Section 4-3-6**) all three metrics (AP, AVP for azimuth and AVP for azimuth, elevation and distance) are discussed to fully analyze the end result. In cases where the resulting experiment leads to less accurate results the loss function results are used to give an explanation as to why this might be, in other cases the loss function nicely converges; therefor it is not included in the explanation.

4-3-2 Baseline

First a baseline is generated to determine what effect each modification to the learning process has on the overall accuracy. The small number of parameters that need to be trained due to the removal of the bounding box parameters, makes it much easier to define a loss function that can guide the network to an optimal solution. The baseline therefore already shows descent results compared to current state-of-the-art results, reaching an AVP of 25% as is shown in Figure 4-1. This accuracy is achieved using the YOLO$_{POSE}$ framework described in the previous chapter. No data clustering is added to the baseline method therefore the number of bins is set to 1 ($n = 1$). Likewise, there is no data augmentation, no data selection and no normalization of the confidence scores. The subsequent experiments will introduce these methods one-by-one.

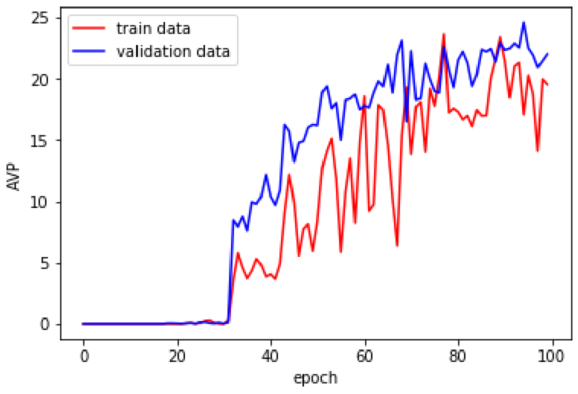
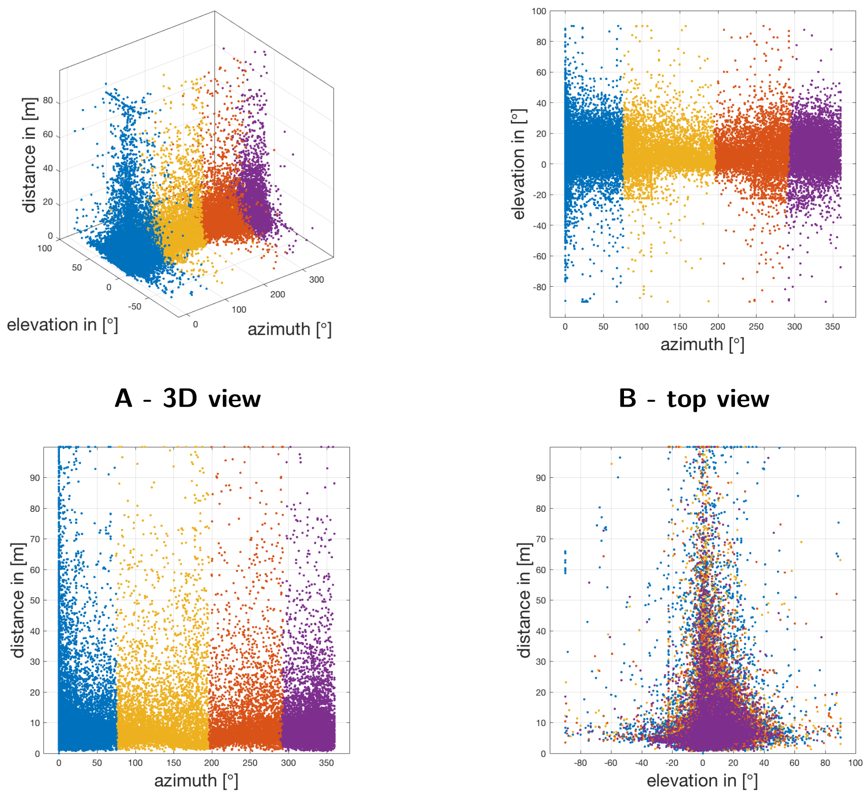


Figure 4-1: Illustration of the AVP for the spherical pose evaluation

When training for the baseline results, the hyper-parameters in the network were tuned to obtain optimal results. A decreasing learning rate was initially used, starting at 1.0 and decreasing every 10 epoch's by a factor of 10. The graph shows that for the first 30 epoch's nothing happens, only when the learning rates reaches 0.001 does the network start to learn. Taking smaller steps also did not effect the accuracy of the output. It was later determined that keeping the learning rate at 0.001 showed the bests results, maintaining the same accuracy yet converging much faster. The learning rate of 0.001 is applied to the remainder of the experiments conducted in this chapter. As was discussed in **Section 4-1** the hyper-parameters are summed up in Table 4-1.

4-3-3 Data clustering

To help the network, it is possible to guide the network in certain directions. One such technique is data clustering, where data is clustered in regions that represent a certain correlation between points. To test this a k-means clustering technique was used to determine whether there is a noticeable correlation between data points (azimuth, elevation and distance). K-means clustering is used because we want to know the nearest cluster center, such that the squared distances from the cluster are minimized. This makes it easier to regress the pose, as a smaller range is always easier to regress over. Figure 4-2 shows the result from the k-means clustering technique performed on the pose parameters. From this figure it is evident that there is no real correlation that is significant, other than the azimuth seems to have the most affect on determining the cluster regions. This is evident from the fact that the data is split over the azimuth axis rather than elevation or distance. This is logical because the data has a more spread out range of azimuth angles, where elevation and distance are concentrated in the region close to zero. Most images in the dataset are made at close range (distance < 10m) and at the same level as the object (elevation close to zero $\{-20°, 20°\}$). However to test the effect of data clustered, the data is split based on elevation, distance and azimuth respectively. This is to test the individual effects that each parameter has on the learning process.



A - 3D view **B - top view**

C - front view **D - side view**

Figure 4-2: Graph of the k-means clustering operation. The different colors represent the clusters generated in the k-means operation, in this case $k = 4$.

Elevation

To start, the elevation is clustered in 4 bins ($n = 4$) to see the effect it has on the learning process. The number of clusters determines the complexity of the system, a higher number of clusters means there are more parameters that need to trained and a lower number means that the effects of clustering become less visible. Initially setting the number of bins to 4 ensure fast convergence while still allowing to test the effects of clustering. The bins are initially sequenced in ranges; $[-90°, -45°]$, $[-45°, 0°]$, $[0°, 45°]$, $[45°, 90°]$. The resulting effect on the learning process can be seen in Figure 4-3. It is evident that clustering the elevation data into bins has a negative effect on the learning process. Figure 4-5 shows this more clearly as the elevation loss does not converge, instead it continually fluctuates. Using different ranges in the elevation bins (i.e. clustering by number of data points resulting in ranges $[-90°, -13.3°]$, $[13.3°, 6.9°]$, $[6.9°, 16.8°]$, $[16.8°, 90°]$, see Figure 4-4) results in the same fluctuating elevation loss. As the k-means clustering showed, the data is just too concentrated making it difficult to learn to choose the correct bin. The bins with larger angles have little data to train from (sequenced clustering) or have the data widely spread out (clustering based on number of data points) making it difficult to learn to accurately regress the pose.

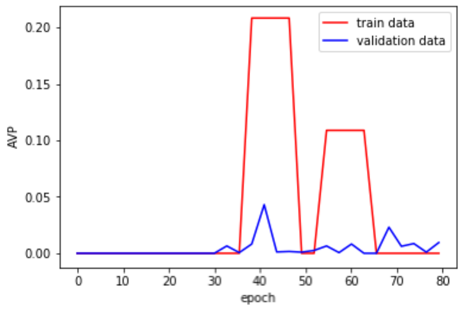


Figure 4-3: Illustration of the AVP for the elevation structured in bins

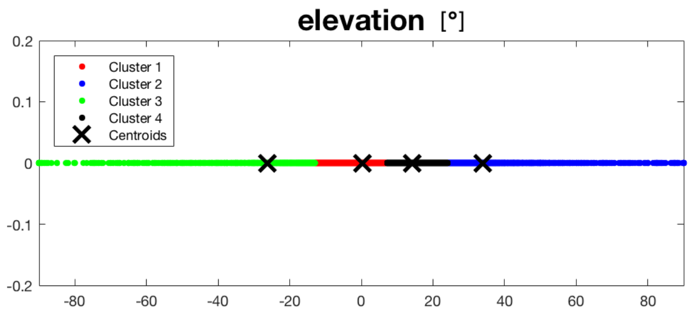
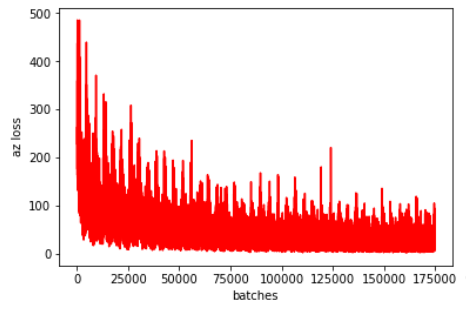
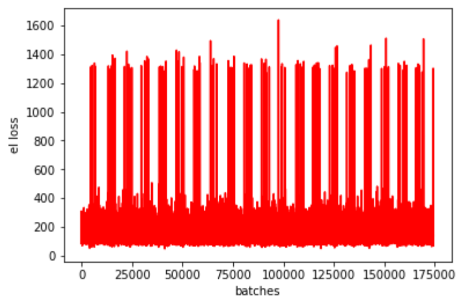


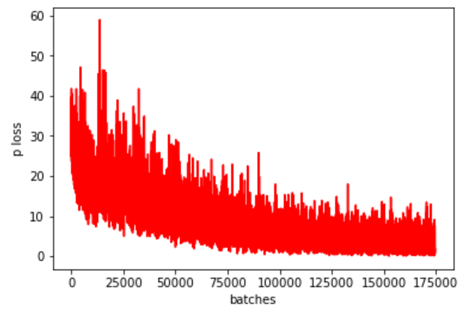
Figure 4-4: Illustration of the clustering of the elevation data in bins of equal number of data points.



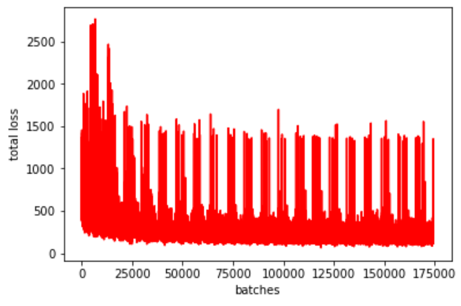
A - Azimuth error



B - Elevation error



C - Class probability error



D - Total error

Figure 4-5: Illustration of the loss function during training for the elevation structured in bins

Distance

Similar to elevation clustering, distance clustering is done in a sequence of 4 ranges ($n = 4$); $[0, 25]$, $[25, 50]$, $[50, 75]$, $[75, 100]$. The resulting graphs are shown in Figure 4-6 and Figure 4-7. From Figure 4-7 a similar effect is noticed as was the case in the elevation clustering instance. Instead of the elevation loss fluctuating, now the distance loss fluctuates constantly and is not able to converge. Using other cluster ranges results in the same fluctuating distance loss. Like in the clustering of the elevation, the distance clustering is not converging because data is clustered in a small region, with sparse number of data points at larger distances.

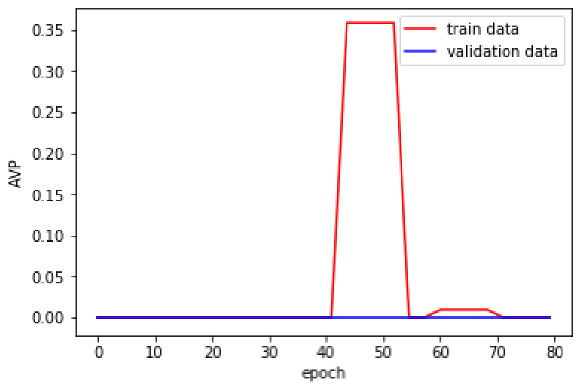
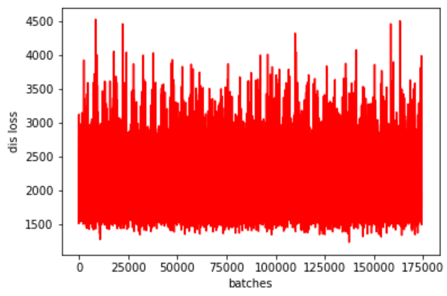
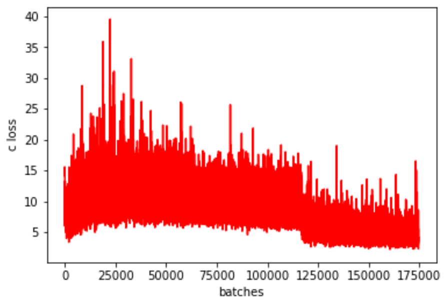


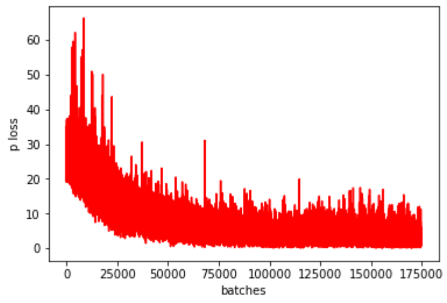
Figure 4-6: Illustration of the AVP for the distance structured in bins



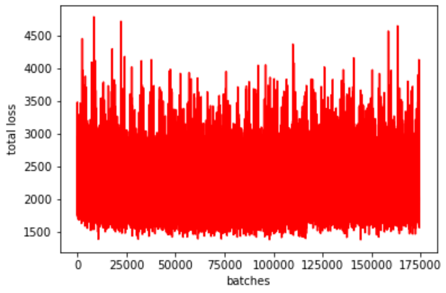
A - Distance error



B - Confidence error



C - Class probability error



D - Total error

Figure 4-7: Illustration of the loss function during training for the azimuth structured in bins

Azimuth

Finally the azimuth is clustered in a sequence of 4 ranges; $[0°, 90°]$, $[90°, 180°]$, $[180°, 270°]$, $[270°, 360°]$. The resulting graphs are shown in Figure 4-6 and Figure 4-7. Unlike the previous two cases the clustering of the azimuth data resulted in a positive effect on the learning process of the network. The accuracy is starting to go towards current state-of-the-art results as accuracy increases from an AVP of 24% (see Figure 4-1) to an accuracy of 29% (see Figure 4-8). As was mentioned in the beginning of this section (see Figure 4-2) the clustering of the data mostly follows the azimuth axis. Therefore, it makes sense that clustering data is most effective when applying it to the azimuth. Because the data is more spread out, each bin has a similar range and a similar number of data points. This helps the network to have higher confidence it its prediction for which bin to select.

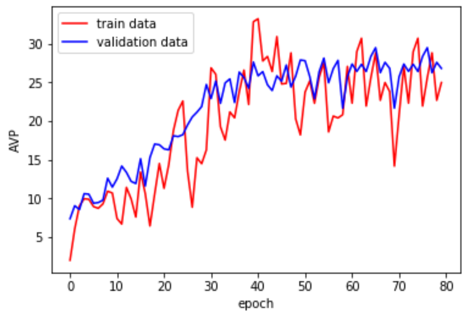


Figure 4-8: Illustration of the AVP for the azimuth structured in bins

Selecting a bin arrangement

Other bin structures are explored to test whether more optimal results can be obtained. Instead of using a sequence of ranges as was used in the case above. Several different approaches were tested; equal number of data points per bin, selecting different starting point of the sequential ranges ($[315°, 45°]$, $[45°, 135°]$, $[135°, 225°]$, $[225°, 315°]$), using the opposing pose in each bin. The first two approaches showed the same results as Figure 4-8, because the data is evenly spread out in the azimuth, it closely represents the sequential range approach. Likewise, shifting the sequential range did not effect the learning process. Using opposing poses per bin did however, show an improvement in the pose estimation. Figure 4-9 shows how the bins are structure compared to the sequential approach. The idea behind it is that using opposing poses helps the network better distinguish between front and back view, left and right view, etc. This is similar to adding a binary option to the bin selection, helping the network to better distinguish between orientations. Simultaneously the idea is that this also helps to better understand small nuances in the pose, as the ranges per side are smaller in each bin. In the sequential method the range on one side equals 90° while that of the opposites method is 45° per side, taking advantage of the binary selection approach of choosing an opposing poses. As long as the correct side is predicted, the pose has a smaller range to regress over.

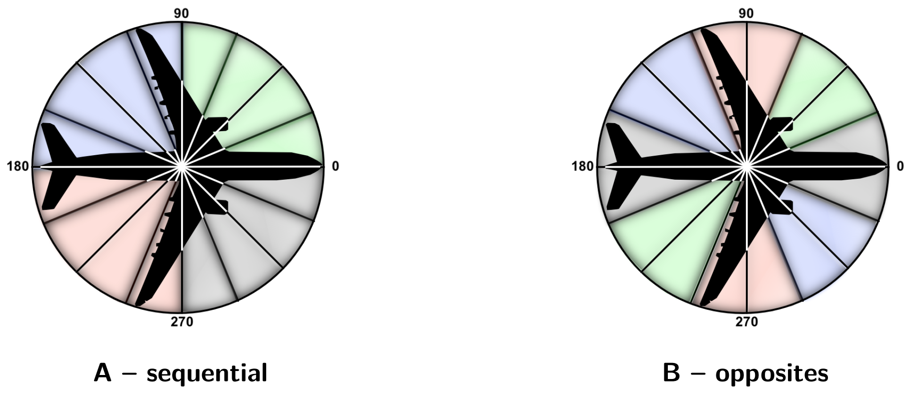


Figure 4-9: Illustration of the different bin types. The red, green, blue and **black** highlights represent the different bins.

The result of this technique is shown in Figure 4-10. The graph shows that the accuracy increases from the 29% in the sequential case to 36% in the opposites case. This is a significant performance increase of approximately 7%. The result is now on par with current state-of-the-art approaches, see Table 2-2.

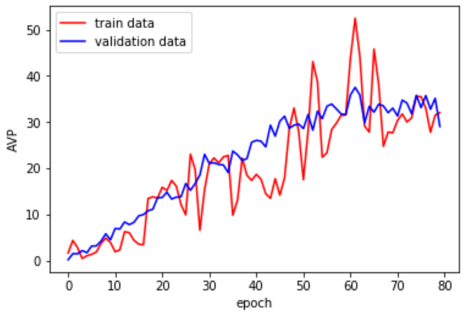


Figure 4-10: Illustration of the AVP for the azimuth structured in bins that are separated by 180°

4-3-4 Free selection of ground truth point

In the previous experiments, during the training process the ground truth point of each object is set as the grid cell where the center point of the object resides. The network learns to estimate the pose for this cell, increasing its confidence value over that of other cells. Now instead of using the center of the object, the ground truth point is freely selected from all grid cells that encompass the object. During training the grid cell that has the best pose prediction for an image after a forward pass of the network, is used as the ground truth grid cell. The idea behind this technique is that the network is able to freely determine which regions of the image best represent the pose. Figure 4-12 shows an illustration of this free selection. The results of this technique are shown in Figure 4-11 and Figure 4-13. It is evident that the network learns in the first 35 epochs but then hits a local minima, it keeps fluctuating not gaining any more traction to be able to increase the accuracy. Figure 4-13 shows that the confidence loss continues to fluctuate throughout the training process. Like the other cases that failed to work, the fluctuating loss shows where the problem lies. Because the ground truth grid cell is freely selected, the network has a hard time determining which grid cell to use. This effects the confidence value, as the network is not able to confidently determine which grid cell best represents the objects pose. Thus the network is not able to converge to an optimal solution.

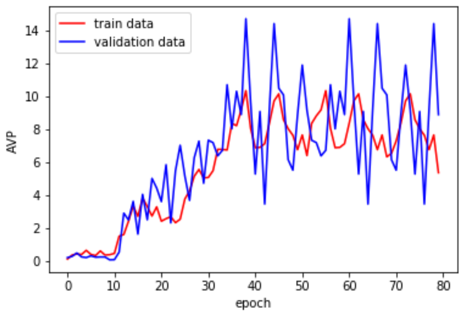


Figure 4-11: Illustration of the AVP for the free selection of the ground truth point

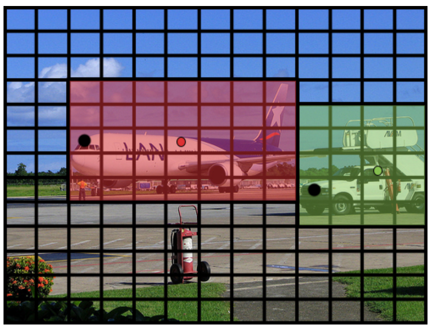


Figure 4-12: Illustration of free ground truth cell selection. The red and green points, represent the cells where the center points of each object resides. The **black** points represent which region the network might select when given a free choice.

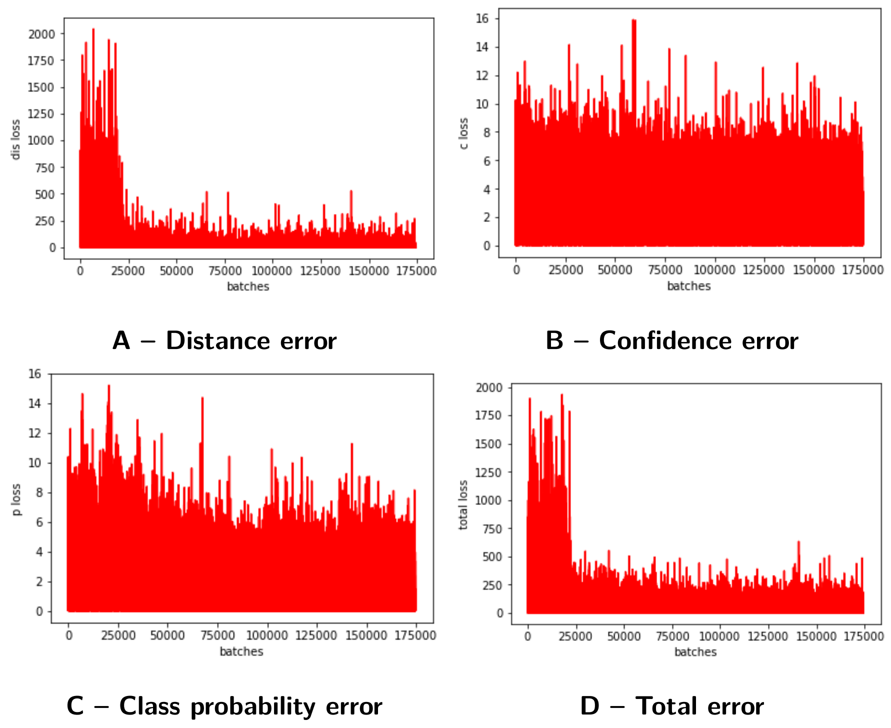


Figure 4-13: Illustration of the loss function during training for the free selection of the ground truth point

4-3-5 Data augmentation

The data augmentation step is discussed in detail in *Section 3-3*. In this experiment data augmentation is applied to the dataset to determine the effect on the performance. The reasoning for data augmentation is that it is expected to reduce overfitting of the network. By introducing new unique data points into the learning process, it is expected that this helps the network to better learn weights that can better distinguish between objects and their poses. The result of data augmentation is shown in Figure 4-14. Accuracy now reaches 40%, approximately 4% points up from the previous best result. The increase in unique images helps the network to learn weights that are less biased to specific colors or specific orientations and positions. This shows that data augmentation helps in reducing the offer-fitting of the network.

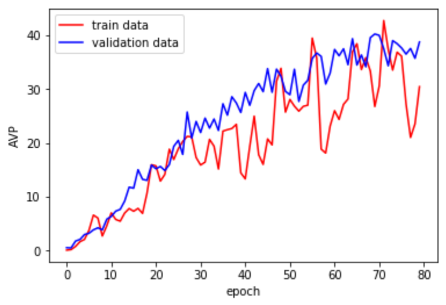


Figure 4-14: Illustration of the AVP for the data augmentation case

4-3-6 Data selection

The final experiment performed for the pose estimation task, as described throughout this paper, is selecting how the data is fed into the network. The idea behind this experiment is that by selecting data and feeding the network in a specific way can pushed the network in a certain learning direction. In this case the data is fed into the network by splitting the batches, where 10 images are selected that contain *more* than 1 object and 15 where there *only* exist only 1 object. Because there are significantly more images with single objects, the images with multiple objects are run through the network multiple times. Each time all the images containing multiple objects have gone through the network, a new random sequence is generated to make each batch unique. Because most of the data is single images, the network tends to focuses more on single objects that are centered around the middle of the image. Resulting in the confidence scores of grid cells in the center of the image to be higher than those of surrounding cells, making it harder to detect multiple objects. Using this data selection technique it is expected that the network will increase the confidence in other cells where objects might reside.

For this last experiment the AP, AVP for azimuth and the AVP for azimuth,elevation and distance are plotted to better analyze the results. Figure 4-15B shows the the same graph as all previous experiments. Unlike previous experiments more than 80 epochs are used as the network continues to learn only leveling out at around 140 epochs, due to the increased amount of batches per epoch and increased complexity. The end result is an accuracy of 50.1% that is up 10% points from the previous results. This is a significant increase and shows the importance of correctly selecting the data that is fed into the network. If only PASCAL VOC data was used this would be less of an issue, because the ImageNet data is also included during training this selection has much more effect. Most of the ImageNet data consists of images containing a single object often centered around the middle.

Figure 4-15C shows the result of the pose estimation of all three parameters, not just azimuth. The accuracy is significantly lower than that of the azimuth, but this is expected as one parameter is easier to get correct than three. With an accuracy of 30.2% the accuracy is still far from optimal.

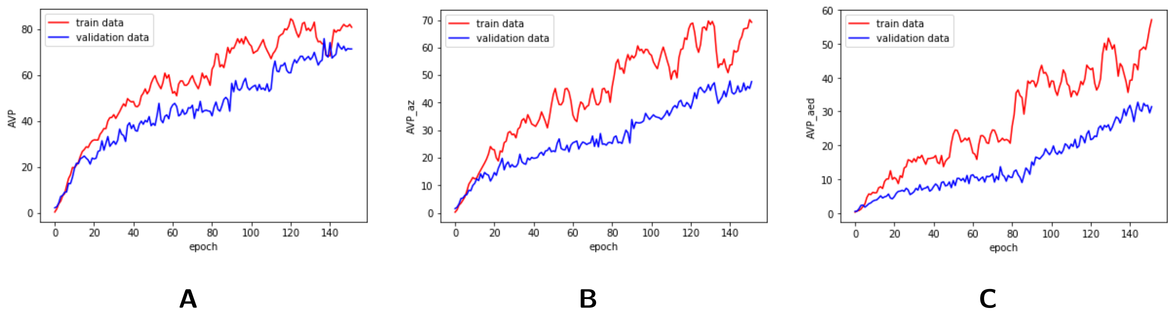


Figure 4-15: Illustration of the AP, AVP azimuth and AVP azimuth, elevation and distance for the data selection case. **A** – AP **B** – AVP azimuth **C** – AVP azimuth, elevation and distance

4-3-7 Normalizing the confidence

In the post-processing step the predictions made by the network are filtered to determine which of these predictions are considered to be correct. In the previous experiments the following steps were taken in the post-processing step.

- Prune everything with a class probability $P < 0.5$.
- Prune everything with a confidence (objectness) $C < 0.5$.
- Determine if predictions in adjacent cells fall within 10% of each other.
- Prune everything that falls within 10%, only keep the prediction with the highest confidence.

Because the confidence score differs significantly per image and per object in the image, pruning everything with $C < 0.5$ can cause correct prediction to be overlooked. To resolve this issue the confidence per image is normalized. Ensuring that all images contain a prediction and so that smaller objects with lower confidence values are not ignored. After the normalization the predictions are pruned using the same cut-off value ($C < 0.5$), see **Section 3-4** for the updated post-processing steps. This has shown a significant improvement on the overall accuracy of the YOLO$_{POSE}$ framework. With an AVP for azimuth of 63.0% and an AVP for azimuth, elevation and distance of 40.4%

4-3-8 Overview of the experiments

Experiment	**Remarks**	**AVP** [%]
Baseline	-	25
Data clustering	Helping the network to find correlations	
- elevation	Data is too concentrated in the 0° region $\{-20^\circ, 20^\circ\}$	0
- distance	Data is too concentrated in the small distance region ($r < 10$m)	0
- azimuth	Data is nicely spread forming equal sized clusters	29
- bin structure	Using opposites poses further guides the network in learning the pose	36
Free selection	Network could not confidently point out a region in the image	10
Data augmentation	Helps with reducing overfitting of the network	40
Data selection	Gives an incentive to the network to give higher confidence to objects outside the center region	50.1
Normalizing the confidence	Improves the prediction filtering in the post-processing step	63

Table 4-2: Overview of the experiments conducted

4-3-9 Comparison with the state-of-the-art

Table 4-3 provides the details of the AVP 24 bins performance improvements over all classes as well as a comparison with six baselines: VDPM [16] was the first technique used on PASCAL3D+, which uses a modified version of **D**iscriminative **P**art based **M**odel (DPM) [5], DPM-VOC+VP [33] also uses a modified version of DPM to predict poses, Render for CNN [37] uses real images from Pascal VOC as well as CAD renders for training a CNN based on AlexNet, and [41] uses a VGG16 architecture and ImageNet data to classify orientations for each object category. More recent baselines include MT-CNN [44] which is an independent type architecture, where two completely decoupled and different deep **N**eural **N**etwork (NN) are used, and finally SSD$_{POSE}$ [43] which uses **S**ingle **S**hot **D**etection (SSD) to regress the pose of objects with a single pass of the network.

The differences between **Y**ou **O**nly **L**ook **O**nce (YOLO)$_{POSE}$ and these other techniques is that YOLO$_{POSE}$ does not look at predicting the bounding box of an object. The only concern of the YOLO$_{POSE}$ architecture is to determine how many objects are in the image, what their individual classes are and what 3D poses they have. Not having to predict the bounding box helps limit the number of parameters that need to be trained in the network.

Methods	Joint Object Detection and Pose Estimation (24 View AVP)												
	aero-plane	bi-cycle	boat	bus	car	chair	table	motor-bike	sofa	train	monitor	Avg. AVP	Avg. AP
VDPM[16]	8.0	14.3	0.3	39.2	13.7	4.4	3.6	10.1	8.2	20.0	11.2	12.1	29.5
DPM-VOC+VP[33]	9.7	16.7	2.2	42.1	24.6	4.2	2.1	10.5	4.1	20.7	12.9	13.6	27.1
RCNN+Alex[37]	21.5	22.0	4.1	38.6	25.5	7.4	11.0	24.4	15.0	28.0	19.8	19.8	56.9
VpKps[41]	37.0	33.4	10.0	54.1	40.0	17.5	19.9	34.3	28.9	43.9	22.7	31.1	56.9
MT-CNN[44]	43.2	39.4	16.8	61.0	44.2	13.5	29.4	37.5	33.5	46.6	32.5	36.1	59.9
SSD$_{POSE}$[43]	33.4	29.4	9.2	54.7	35.7	5.5	22.9	30.3	27.5	44.1	24.3	28.8	59.3
YOLO$_{POSE}$	71.8	41.1	33.6	63.7	**56.3**	25.8	50.1	47.5	59.6	**56.3**	58.3	50.1	75.3
YOLO$_{POSE}$ norm.	**87.9**	**52.7**	**38.9**	**69.0**	55.5	**32.8**	**50.7**	**49.0**	**60.6**	54.4	**60.7**	**63.0**	**84.5**

Table 4-3: Joint object detection and pose estimation of azimuth (φ) (24 View AVP)

From Table 4-3 it is evident that the YOLO$_{POSE}$ architecture significantly outperforms the current state-of-the-art approaches. In all classes the AP and AVP are higher than other techniques. This is most likely due to using less parameters by ignoring the bounding box parameters, helping the network to more directly predict the pose. Because the YOLO architecture is so efficient the network also outperforms all other technique in the number of frames that can be processed each second. The only network that shows similar speeds is the SSD$_{POSE}$ architecture.

Contrary to these other techniques the YOLO$_{POSE}$ architecture also allows to determine the elevation and distance of objects. The results that are generated for this metric cannot be compared to other techniques as this is the first case where azimuth, elevation and distance are predicted simultaneously for PASCAL3D+. The end result of 30.2% and 40.4% is respectable compared to what other techniques are able to achieve for just the azimuth angle.

Methods	Joint Object Detection and Pose Estimation (24 View AVP)											
	aero-plane	bi-cycle	boat	bus	car	chair	table	motor-bike	sofa	train	monitor	Avg.
YOLO$_{POSE}$	42.4	14.0	27.3	56.2	48.9	3.4	31.7	19.2	27.8	**49.0**	7.5	30.2
YOLO$_{POSE}$ norm.	**67.8**	**25.9**	**32.5**	**63.2**	**49.6**	**5.2**	**32.0**	**24.6**	**34.0**	46.5	**13.7**	**40.4**

Table 4-4: Joint object detection and pose estimation of azimuth (φ), elevation (θ) and distance (r) (24 View AVP)

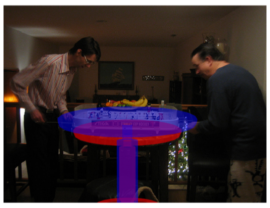
Figure 4-16 shows some of the outputs of the network on the validation data. Table 4-5 states whether prediction are correct and gives an observation of the scene and result.

Image	Correct/ Incorrect	Notes
A	Correct	Working under poor lighting condition.
B	Correct	Nicely matches the ground truth, no further remarks.
C	Incorrect	1 plane has not been detected and the other has a high elevation error.
D	Correct	Both objects are correctly predicted, even though one occludes the other.
E	Incorrect	Elevation is above threshold ($\theta > 7°$), making it only partially correct.
F	Correct	Shows that the ground truth model doesn't always overlap nicely, however the prediction nicely follows the ground truth.
G	Correct	Nicely matches the ground truth, no further remarks.
H	Correct	Distance error, although small, shows a big difference to the ground truth.
I	Correct	Doors and hood of car are open, showing that for difficult object representations a correct prediction can be given.
J	Correct	Both predictions are correct, yet the overlay does not match the ground truth nicely.
K	Correct	Nicely matches the ground truth, no further remarks.
L	Correct	Nicely matches the ground truth, no further remarks.
M	Incorrect	One object is not detected and another has an incorrect pose.
N	Correct	Nicely matches the ground truth, no further remarks.
O	Correct	Nicely matches the ground truth, no further remarks.
P	Correct	Working under strong occlusion.
Q	Correct	The relatively small and partly occluded objects are nicely predicted.
R	Correct	Working under strong occlusion.

Table 4-5: Caption

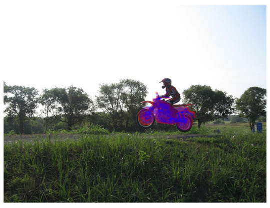
Figure 4-17 shows some additional outputs where the class prediction was incorrect (**A - C**), these problems often occur between object classes that are similar to one another (bus and car, bicycle and motorbike, chair and sofa). Finally **D** shows an example where a prediction was not properly pruned, generating two positive results for a single object.

The overall network seems to nicely predict the pose of each object when overlaying the CAD models. However, classification between similar objects, multiple predictions for one object and pruning of correct prediction when multiple object reside in the image are some of the issues that can be improved on.



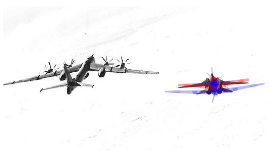
$Er_{azimuth}$: 1.56° , $Er_{elevation}$: 0.81°, $Er_{distance}$: 0.87

A



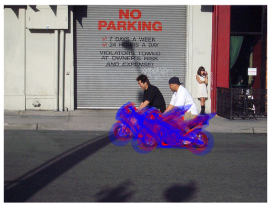
$Er_{azimuth}$: 0.14° , $Er_{elevation}$: 5.43°, $Er_{distance}$: 0.08

B



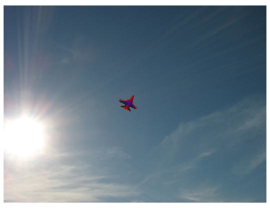
$Er_{azimuth}$: 1.151° , $Er_{elevation}$: 21.286°, $Er_{distance}$: 4.876

C



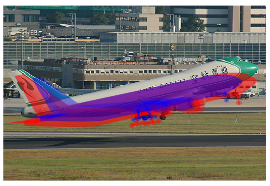
$Er_{azimuth}$: 11.31° , $Er_{elevation}$: 6.088°, $Er_{distance}$: 2.313
$Er_{azimuth}$: 3.374° , $Er_{elevation}$: 4.519°, $Er_{distance}$: 1.877

D



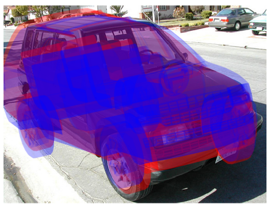
$Er_{azimuth}$: 2.014° , $Er_{elevation}$: 31.005°, $Er_{distance}$: 1.116

E



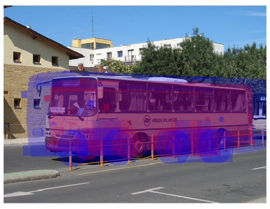
$Er_{azimuth}$: 1.353° , $Er_{elevation}$: 0.397°, $Er_{distance}$: 0.737

F



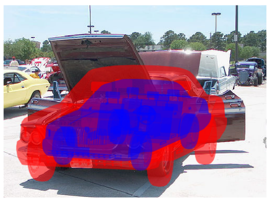
$Er_{azimuth}$: 0.981° , $Er_{elevation}$: 4.632°, $Er_{distance}$: 0.151

G



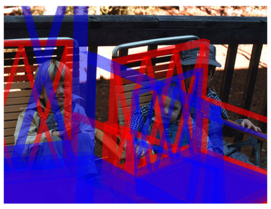
$Er_{azimuth}$: 1.45° , $Er_{elevation}$: 0.428°, $Er_{distance}$: 0.801

H



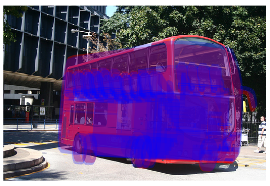
$Er_{azimuth}$: 2.872° , $Er_{elevation}$: 3.268°, $Er_{distance}$: 1.597

I



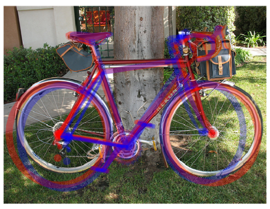
$Er_{azimuth}$: 1.802° , $Er_{elevation}$: 2.936°, $Er_{distance}$: 1.464
$Er_{azimuth}$: 3.647° , $Er_{elevation}$: 6.157°, $Er_{distance}$: 1.503

J



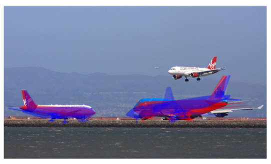
$Er_{azimuth}$: 0.11° , $Er_{elevation}$: 1.135°, $Er_{distance}$: 0.07

K



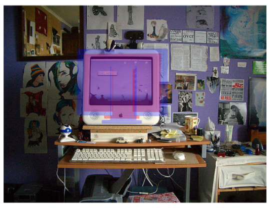
$Er_{azimuth}$: 7.274° , $Er_{elevation}$: 4.811°, $Er_{distance}$: 0.028

L



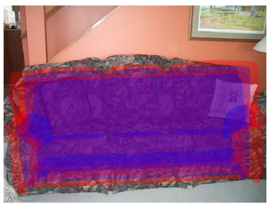
$Er_{azimuth}$: 13.19° , $Er_{elevation}$: 2.334°, $Er_{distance}$: 3.972
$Er_{azimuth}$: 39.895° , $Er_{elevation}$: 18.17°, $Er_{distance}$: 1.684

M



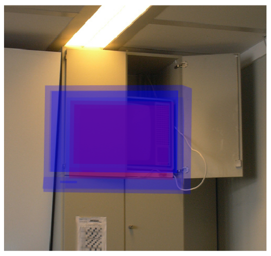
$Er_{azimuth}$: 2.05° , $Er_{elevation}$: 0.233°, $Er_{distance}$: 1.997

N



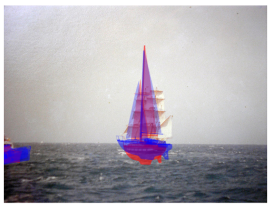
$Er_{azimuth}$: 2.06° , $Er_{elevation}$: 1.279°, $Er_{distance}$: 0.515

O



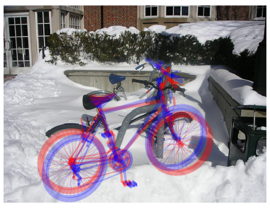
$Er_{azimuth}$: 4.802° , $Er_{elevation}$: 1.029°, $Er_{distance}$: 2.937

P



$Er_{azimuth}$: 8.313° , $Er_{elevation}$: 4.996°, $Er_{distance}$: 4.626
$Er_{azimuth}$: 7.583° , $Er_{elevation}$: 3.321°, $Er_{distance}$: 1.026

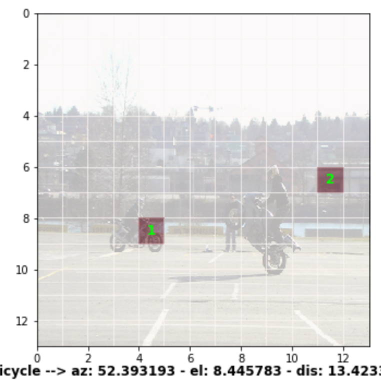
Q



$Er_{azimuth}$: 7.226° , $Er_{elevation}$: 2.48°, $Er_{distance}$: 0.226

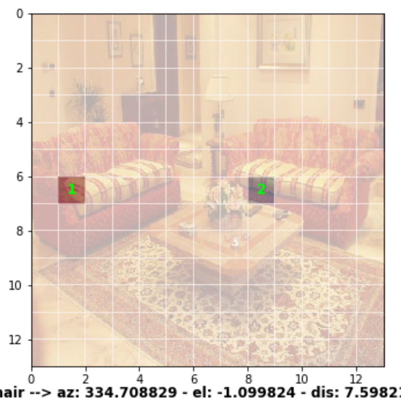
R

Figure 4-16: Illustration of the output results of the network, red represent the ground truth CAD model overlay and blue represent the CAD model overlay for the network prediction.



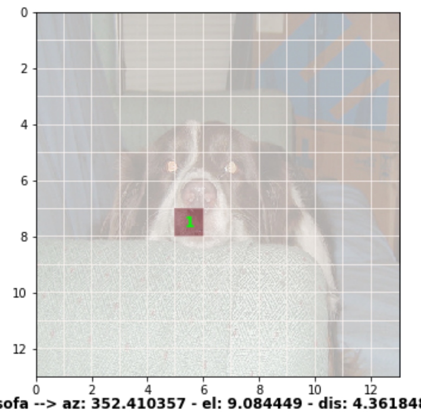
```
1-bicycle --> az: 52.393193 - el: 8.445783 - dis: 13.423301
az_err: 42.126596 - el_err: 5.151783 - dis_err: 5.583301
2-bicycle --> az: 38.315114 - el: -6.495135 - dis: 13.150722
az_err: 35.087557 - el_err: 9.789135 - dis_err: 5.310722
```

A



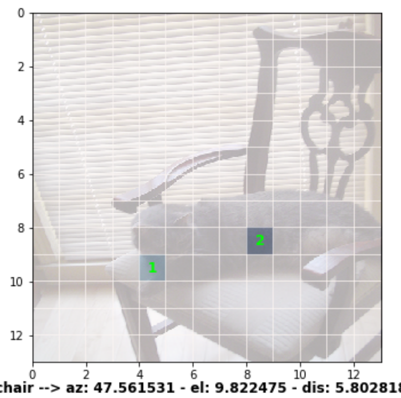
```
1-chair --> az: 334.708829 - el: -1.099824 - dis: 7.598216
az_err: 21.735586 - el_err: 7.021824 - dis_err: 4.161784
2-sofa --> az: 22.628438 - el: 11.137497 - dis: 6.654114
az_err: 2.224219 - el_err: 5.215497 - dis_err: 1.235677
```

B



```
1-sofa --> az: 352.410357 - el: 9.084449 - dis: 4.361848
az_err: 3.794822 - el_err: 9.084449 - dis_err: 4.361848
```

C



```
1-chair --> az: 47.561531 - el: 9.822475 - dis: 5.802818
az_err: 3.345234 - el_err: 31.271525 - dis_err: 1.702818
2-chair --> az: 30.351105 - el: 33.174151 - dis: 4.887103
az_err: 11.950448 - el_err: 7.919849 - dis_err: 0.787103
```

D

Figure 4-17: Illustration of the output results of the network with 13x13 grid. The number in the cell represent the cell with the highest confidence.

4-4 Chapter overview

Unsuccessful experiments:

- **Data clustering of the elevation and distance:** The network had a hard time selecting the correct bins for the elevation and distance based clustering, resulting in the network not converging to a local optima.
- **Free selection:** Giving the network a free selection to decide which part of an object best represents the pose did not yield any improved results. The problem was that the network could not confidently determine which regions to focus on, constantly fluctuating between different regions.

Successful experiments:

- **Data clustering of the azimuth:** Clustering the data over the azimuth angle has allowed the network to successfully distinguish between the different bins. Because the data in each bin is equally spread out, the network is able to confidently determine which bin to select, unlike the elevation and distance clustering. This helps the network by introducing a constraint on the range, over which the pose needs to be regressed. Further improvements were made when the bin structure was modified to classify between opposites. This has shown to significantly improve the accuracy of the pose prediction by reducing the range over which the network has to regress even further.
- **Data augmentation:** Data augmentation has shown that it can be used to limit the effect of overfitting. Improving the accuracy of the network by introducing more unique data that the network can learn from.
- **Data selection:** Data selection has shown that it is important how data is fed to the network in the learning process. By producing a batch that is split into data containing single objects and data containing multiple objects, the confidence score increased significantly for objects residing outside the center region of the image. The result is a significant improvement on the overall accuracy.
- **Confidence normalization:** It became evident that correct predictions were being prune in the post-processing step because the confidence scores fell bellow the threshold. Normalizing the confidence in the post-processing step helped prevent images and objects with lower confidence scores to be pruned from the output, without having to lower the threshold resulting in an increased number of FPs.

The results have shown to be a significant improvement on the current state-of-the-art, showing a 26.9% point increase over that of the next best result[44]. The output also show that the pose prediction that are correct nicely overlap with the ground truth. The problems mostly occur when their are multiple objects in the image and when classification goes wrong between similar classes (bus and car, etc.).

Chapter 5

Concluding remarks

In the previous chapters the problem of object classification and pose estimation has been addressed. Starting with a small introduction into the problem, stating the goals of this research and looking into related work. After which a detailed look into the YOLO$_{POSE}$ architecture was given, discussing individual components and the roles they play in the pose estimation problem. Finally the experiments and results were discussed, making a comparison with state-of-the-art methods obtained from the research done on related works.

This chapter will give a conclusion based on the findings made and results obtained in this research. The chapter will close off with some recommendations for future works.

5-1 Conclusions

The research conducted has demonstrated that the problem of *"Pose regression of 3D objects in monocular framework using a Convolutional Neural Network"* was achieved. The results have shown that there is a significant improvement in the accuracy of the pose estimation compared to the current state-of-the-art, showing an increase of $26,9\%$ points.

From the experiments conducted several conclusion can be drawn:

- Discarding bounding box parameters resulted in a simplified objective that allowed the network to focus more on the classification and pose prediction of objects.
- Data clustering only works when clustering is done properly, it is important to ensure that each bin has an equal range and number of data points to train on. Similarly the structure of the bins is important, constraining the network to choose between opposite poses is beneficial, reducing the range over which the network needs to regress.
- Data augmentation has been proven to work in other networks like [37]. The results in this research again prove that data augmentation can improve the overall accuracy, limiting the effects of overfitting of the network.

- Data selection has proven to be helpful for the network to understand where it should focus, in this particular case it is guiding the network to generate higher confidence scores for objects that lie outside the center region of the image.
- Post-processing in the YOLO$_{POSE}$ architecture can have a significant impact on the overall accuracy. It pays to properly filter the many predictions generated by the CNN, using a simple normalization step on the confidence scores has proven to result in a significant accuracy increase.

The research conducted has shown that accuracy gains are not only achieved through better network architecture but are also highly dependent on the training and processing techniques used. This is evident from the accuracy increase of 38% obtained in this research, from an AVP of 25% to an AVP of 63%.

5-2 Recommendations and future work

It is expected that the results obtained in this research can be improved on further. The following research is suggested to be conducted to test whether higher accuracy's are possible:

- **Dataset:**
  - Making the PASCAL3D+ data more accurate because the CAD models used for the annotations don't take size into considerations. A bike is similar in size to a aeroplane according to the PASCAL3D+ annotation. By implementing a scale for each objects class, it gives a better indication of the distance of objects.
  - Testing on different datasets, to determine the effectiveness of the network. This should also show further ways to improve the network, showing weaknesses not shown on the PASCAL3D+ dataset.
- **Network:** Using a different network like SSD$_{POSE}$ to determine if using the same technique as described in this research will show similar accuracy's.
- **Data augmentation:** Augmenting the data in such a way that the center regions aren't overly represented and augment the data so that more unique poses are added for the network to train on. Data augmentation can be expanded in a similar manner as in [37], using renders of 3D CAD models.
- **Data selection:** The main problem that was partially addressed by the data selection experiment, is that confidence of the center region still remains higher than the surrounding regions, resulting in objects being filtered out in the post-processing step. Suggested is that data selection is further expanded, the selection process might be improved further using techniques similar to [60].
- **Post-processing:** In the post processing step normalizing the confidence prediction per image resulted in an significant increase in accuracy. However, there are still predictions that are wrongfully pruned from the output. Different post-processing techniques like [61] can be used, where a **R**ecurrent **N**eural **N**etwork (RNN) is added to select the correct predictions.

- **Computational efficiency:** The YOLO$_{POSE}$ network has shown to be able to run in real-time on a powerful GPU (Tesla M60). However to run in real-time on a more portable solution like an Intel UP board[1] or NVIDIA Jetson[2] a few changes need to be made to the architecture. The tiny YOLO architecture, with a reduced number of layers is an example. The tiny YOLO architecture has shown to be able to run in real-time on a android phone[3] with good results.

[1] https://www.up-board.org

[2] https://developer.nvidia.com/embedded/develop/hardware

[3] https://github.com/natanielruiz/android-yolo

Appendix A

Failed loss functions

Two different pose representations were tested in this research. The reason for this is that pose representation can be done any number of ways, each with their own advantages and disadvantages. Quaternion and spherical coordinates are used as they pertain significance in robotics and PASCAL3D+ respectively. Quaternion pose representation, compared to Euler angles, are simpler to compose and avoid gimbal lock. Similarly they are more compact, more numerically stable, and more efficient when compared to rotation matrices. For these reasons the quaternion representation is often used in fields like computer vision and robotics. Spherical coordinates on the other hand are less common in robotics and are more commonly used in geography and astronomy to determine points on earth and of heavenly bodies. However to define the pose of multiple objects within an image the spherical coordinates are useful as they are a compact and efficient representation of the pose, only requiring 3 parameters (azimuth, elevation and distance). This appendix chapter will look at the failed attempt of using the quaternion representation.

Quaternion coordinate system

A quaternion can represent 3D reflections, rotations and scaling but cannot represent the translation, therefore the translation part needs to be handled separately. This results in a total of 7 parameters that need to be used to represent the 3D pose of an object. This includes the 4 quaternion parameters (p,q,r,w) and the 3 translation parameters (x,y,z). The network becomes harder to train as the number of parameters increase. The increased number of parameters results in a more complex loss function that has more possible local minima. Because of this difficulty the model wasn't able to properly train the network to perform accurate pose estimation of objects. The graphs in Figure A-1 and Figure A-2 show how the network struggles to accurately determine the pose. From Figure A-1 it is evident that the network hits a local minima that it cannot get out of. The values from the loss function show that the network learns to decrease the class probability and the confidence loss but it is not able to learn the pose loss. It constantly fluctuates between a range of 200 and 1200, not being able to converge. This was the case for a range of different tuning parameters and different loss functions.

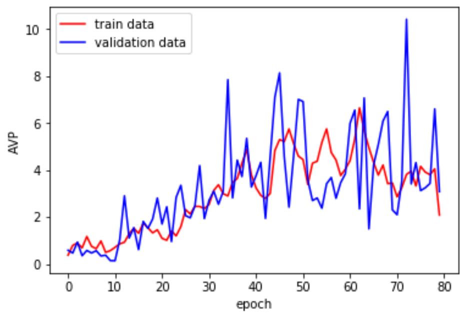
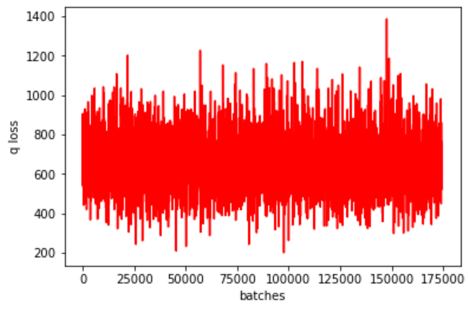
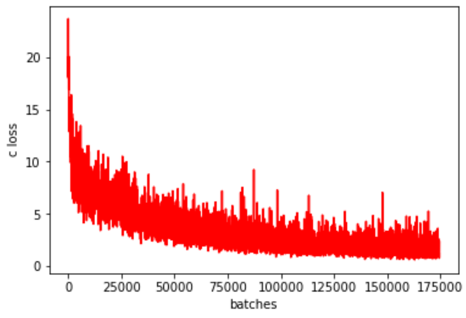


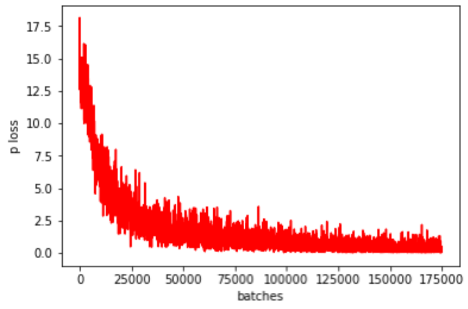
Figure A-1: Illustration of the AVP for the quaternion pose evaluation



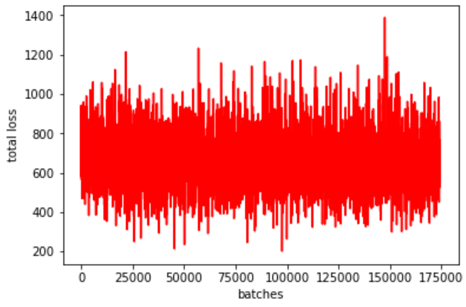
A - pose error



B - Confidence error



C - Class probability error



D - Total error

Figure A-2: Illustration of the loss function during training for the quaternion pose estimation

Bibliography

[1] L. G. Roberts, *Machine perception of three-dimensional solids.* PhD thesis, Massachusetts Institute of Technology, 1963.

[2] M. A. Fischler and R. A. Elschlager, "The representation and matching of pictorial structures," *IEEE Trans. Comput.*, vol. 22, pp. 67–92, Jan. 1973.

[3] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, pp. I–I, IEEE, 2001.

[4] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893, IEEE, 2005.

[5] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

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

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[8] M. Liang and X. Hu, "Recurrent convolutional neural network for object recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

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

[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, *SSD: Single Shot MultiBox Detector*, pp. 21–37. Springer International Publishing, 2016.

[11] D. G. Lowe, "The viewpoint consistency constraint," *International Journal of Computer Vision*, vol. 1, no. 1, pp. 57–72, 1987.

[12] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 4, pp. 509–522, 2002.

[13] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016.

[14] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, June 2010.

[15] 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*, pp. 740–755. Cham: Springer International Publishing, 2014.

[16] Y. Xiang, R. Mottaghi, and S. Savarese, "Beyond pascal: A benchmark for 3d object detection in the wild," in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2014.

[17] C. P. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection," in *Computer vision, 1998. sixth international conference on*, pp. 555–562, IEEE, 1998.

[18] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2, pp. 1150–1157, Ieee, 1999.

[19] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893, IEEE, 2005.

[20] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *International conference on machine learning*, pp. 647–655, 2014.

[21] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, no. 2, pp. 137–154, 2004.

[22] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *Image Processing. 2002. Proceedings. 2002 International Conference on*, vol. 1, pp. I–I, IEEE, 2002.

[23] 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*, pp. 580–587, 2014.

[24] M. B. Blaschko and C. H. Lampert, "Learning to localize objects with structured output regression," in *European conference on computer vision*, pp. 2–15, Springer, 2008.

[25] 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.

[26] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 91–99, Curran Associates, Inc., 2015.

[27] R. Girshick, "Fast r-cnn," *arXiv preprint arXiv:1504.08083*, 2015.

[28] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2147–2154, 2014.

[29] C. Szegedy, S. Reed, D. Erhan, D. Anguelov, and S. Ioffe, "Scalable, high-quality object detection," *arXiv preprint arXiv:1412.1441*, 2014.

[30] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, pp. 381–395, June 1981.

[31] D. F. Dementhon and L. S. Davis, "Model-based object pose in 25 lines of code," *International Journal of Computer Vision*, vol. 15, pp. 123–141, Jun 1995.

[32] F. Rothganger, S. Lazebnik, C. Schmid, and J. Ponce, "3d object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints," *International Journal of Computer Vision*, vol. 66, pp. 231–259, Mar 2006.

[33] B. Pepik, M. Stark, P. Gehler, and B. Schiele, "Teaching 3d geometry to deformable part models," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3362–3369, IEEE, 2012.

[34] M. Hejrati and D. Ramanan, "Analyzing 3d objects in cluttered images," in *Advances in Neural Information Processing Systems*, pp. 593–601, 2012.

[35] Y. Xiang and S. Savarese, "Estimating the aspect layout of object categories," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3410–3417, IEEE, 2012.

[36] M. Z. Zia, M. Stark, B. Schiele, and K. Schindler, "Detailed 3d representations for object recognition and modeling," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 11, pp. 2608–2623, 2013.

[37] H. Su, C. R. Qi, Y. Li, and L. J. Guibas, "Render for CNN: viewpoint estimation in images using cnns trained with rendered 3d model views," *CoRR*, vol. abs/1505.05641, 2015.

[38] M. Aubry, D. Maturana, A. A. Efros, B. C. Russell, and J. Sivic, "Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3762–3769, 2014.

[39] M. Osadchy, Y. L. Cun, and M. L. Miller, “Synergistic face detection and pose estimation with energy-based models,” *Journal of Machine Learning Research*, vol. 8, no. May, pp. 1197–1215, 2007.

[40] H. Penedones, R. Collobert, F. Fleuret, and D. Grangier, “Improving object classification using pose information,” tech. rep., Idiap, 2012.

[41] S. Tulsiani and J. Malik, “Viewpoints and keypoints,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

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

[43] P. Poirson, P. Ammirato, C. Fu, W. Liu, J. Kosecka, and A. C. Berg, “Fast single shot detection and pose estimation,” *CoRR*, vol. abs/1609.05590, 2016.

[44] F. Massa, R. Marlet, and M. Aubry, “Crafting a multi-task cnn for viewpoint estimation,” *arXiv preprint arXiv:1609.03894*, 2016.

[45] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3354–3361, IEEE, 2012.

[46] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, “Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes,” in *Asian conference on computer vision*, pp. 548–562, Springer, 2012.

[47] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, “Learning 6d object pose estimation using 3d object coordinates,” in *European conference on computer vision*, pp. 536–551, Springer, 2014.

[48] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *CoRR*, vol. abs/1409.4842, 2014.

[49] R. J. Douglas and K. A. Martin, “Recurrent neuronal circuits in the neocortex,” *Current biology*, vol. 17, no. 13, pp. R496–R500, 2007.

[50] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

[51] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[52] 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*, pp. 770–778, 2016.

[53] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *CoRR*, vol. abs/1502.03167, 2015.

[54] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International conference on artificial neural networks*, pp. 92–101, Springer, 2010.

[55] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

[56] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[57] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[58] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, pp. 1139–1147, 2013.

[59] C. D. Manning, P. Raghavan, H. Schütze, *et al.*, *Introduction to information retrieval.* Cambridge university press Cambridge, 2008.

[60] M. van der Wees, A. Bisazza, and C. Monz, "Dynamic data selection for neural machine translation," *arXiv preprint arXiv:1708.00712*, 2017.

[61] L. Ma, X. Kan, Q. Xiao, W. Liu, and P. Sun, "Yes-net: An effective detector based on global information," *arXiv preprint arXiv:1706.09180*, 2017.

Glossary

List of Acronyms

AP	**A**verage **P**recision
AVP	**A**verage **V**iewpoint **P**recision
BFGS	**B**royden, **F**letcher, **G**oldfarb, and **S**hanno
CNN	**C**onvolutional **N**eural **N**etwork
CV	**C**omputer **V**ision
DPM	**D**iscriminative **P**art based **M**odel
FN	**F**alse **N**egative
FP	**F**alse **P**ositive
fps	**f**rames **p**er **s**econd
GPU	**G**raphics **P**rocessing **U**nit
HSV	**H**ue-**S**aturation-**V**alue
IAP	**I**nterpolated **A**verage **P**recision
ILSVRC	**I**mageNet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge
IoU	**I**ntersection **O**ver **U**nion
NMS	**N**on-**M**aximum **S**uppression
NN	**N**eural **N**etwork
PnP	**P**erspective **n**-point **P**roblem
R-CNN	**R**ecurrent **C**onvolutional **N**eural **N**etwork
ReLU	**Re**ctified **L**inear **U**nit

RGB	**R**ed-**G**reen-**B**lue
RoI	**R**egion **o**f **I**nterest
RoIs	**R**egions **o**f **I**nterest
RNN	**R**ecurrent **N**eural **N**etwork
RPN	**R**egion **P**roposal **N**etwork
SGD	**S**tochastic **G**radient **D**escent
SS	**S**elective **S**earch
SSD	**S**ingle **S**hot **D**etection
SVM	**S**upport **V**ector **M**achine
TN	**T**rue **N**egative
TP	**T**rue **P**ositive
VGG	**V**isual **G**eometry **G**roup
VOC	**V**isual **O**bject **C**lasses
VpKps	**V**iew**p**oints & **K**ey**p**oint**s**
YOLO	**Y**ou **O**nly **L**ook **O**nce

List of terms

Epoch	A full pass through of the the entire training set
Regression	A set of statistical processes for estimating the relationships among variables
over-fitting	Refers to a model that models the training data too well, negatively impacting the perform
RGB	Color model based on additive color primaries
HSV	Alternative representation of the RGB color model