MASTER THESIS

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Software Engineering

Object Detection Based on Convolutional Neural Networks Trained on Synthetically Generated Data

By: Francisco Depascuali

Student Number: 1620299001

Supervisor 1: Mandl, Thomas
Supervisor 2: Schreiber, David

Vienna, Austria, December 22, 2017

FH University of Applied Sciences
TECHNIKUM WIEN

Declaration

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (for example see §§21, 42f and 57 UrhG (Austrian copyright law) as amended as well as §14 of the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien).

In particular I declare that I have made use of third-party content correctly, regardless what form it may have, and I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see §14 para. 1 Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.“

Vienna, Austria, January 8, 2018 Signature

Kurzfassung

Objekterkennung ist inzwischen einer der aktivsten Forschungsbereich der Computergrafik. Nichtsdestotrotz ist es weiterhin ein offenes Forschungsthema. Die größte Herausforderung in diesem Forschungsfeld ist die Bildvariation. Jedes Objekt erzeugt für den Betrachter beliebig viele Bilder basierend auf der Stellung, der Position, der Lichtverhältnisse und der Umgebung. Es wurde jedoch gezeigt, dass es - genügend Traningsdaten vorausgesetzt - möglich ist mit sehr einfachen statistischen Modellen computergrafische Probleme mit guten Resultaten zu lsen. Die Beschaffung der benötigten Menge an Daten für reale Szenarien ist kein leichtes Unterfangen; typischerweise werden zwischen 50k und 100k Bilder benötigt. Diese Arbeit beschäftigt sich mit dem Problem der Erzeugung von hinreichend großen Datensets, die groß genug sind, durch die Erforschung von Werkzeugen, die synthetischen Daten erstellen. Diese Datensets werden dann genutzt um einen Objektdetektor, der den letzten Stand der Technik widerspiegelt, nämlich der Single Shot MultiBox Detector (SSD) zu trainieren, der in einem Datenset von 284 realen Bildern eine Genauigkeit von 83,9% (mAP) erreicht. Die in diesem Projekt erreichten Resultate zeigen Potential für die Verbesserung in anderen Bereichen der Computergrafik wie Videoüberwachung, medizinische Bildgebung, autonomes Fahren, Augmented Reality, Robotik sowie für die Computergrafik im Allgemeinen.

Schlagworte: Objekterkennung, Faltendes neurales Netzwerk, Synthetisch erzeugte Daten

Abstract

Object detection has become one of the most active research areas in computer vision. Even so, it continues to be an open research topic. The hardest problem in this field is image variation: each object in the world casts infinite images to the viewer, depending on its pose, position, and the surrounding light conditions and background. However, it has been shown that when there is enough training data available, even simple statistical models can address computer vision problems with outstanding results. The required data for real-world scenarios tend to be hard to obtain in large quantities; usually between 50k and 100k images are needed. This thesis addresses the problem of availability of big enough data sets by exploring synthetic data generation tools. Those datasets are then used to train a state of the art object detector, Single Shot MultiBox Detector (SSD), achieving 83.9% of accuracy (mAP) in a dataset of 284 real-world images. The results obtained in this project hold potential for improving other areas of computer vision, such as video surveillance, medical imaging, autonomous driving, augmented reality and robotics, as well as advancing the current state of computer vision in general.

Keywords: Object Detection, Convolutional Neural Networks, Synthetically Generated Data

Acknowledgements

This project wouldn't have been possible without the help of my academic supervisor Thomas Mandl, who was always available for answering questions and provided proper guidance throughout the whole investigation. His advice over maintaining focus on the research questions and reminders of the scope have been crucial for arriving to a great work within the given constraints.

I would also like to thank my technical supervisor, David Schreiber, for his constant advice throughout the whole investigation, combined with the freedom to try different approaches and providing all the tools needed for this project to work.

I definitely have to mention Andreas Zweng, an expert of the field, for his suggestions along the investigation, and Gustavo Fernandez, with his valuable feedback for preparing the presentations. Furthermore, the inspiration of this thesis relays on the work done by Emilio Tylson, that constitutes a solid base in which this thesis is built on.

In addition, I would like to mention and express my gratitude to Austrian Institute of Technology (AIT), specially to Andreas Kriechbaum-Zabini, Axel Weissenfeld and the whole computer vision team for their constant support.

Last of all, I would like to thank Ivan Klejnak and Lucas Kania, as we traveled through the same journey together. Thanks for all the feedback along the investigation, and for those experiences that shaped us, not only in a technical aspect, but also in our personalities.

Contents

1 Introduction

Teaching computers to be able to understand the world that surround them has been actively studied by scientists. One critical aspect of this understanding is computer vision: the ability of a computer to process visual input from the real world. The brain's natural capability to process images hides the extremely difficult task it is for computers. At the heart of this problem is the infinite number of images that an object can cast to a viewer's retina. Each image will have a different object position, size, orientation, pose and light conditions. Humans are usually able to correctly detect objects, even when the object isn't completely visible. This is not the same for computers: a small modification of an image, which could be imperceptible for human vision, can radically change what the computer is able to identify. This is explained by the difference in the input of human vision and computers: while we see colours and shapes, computers visualise a collection of numbers. These numbers, in computer vision terms, refer to a collection of pixels. A pixel can be represented in different ways, for example HEX and RGB.

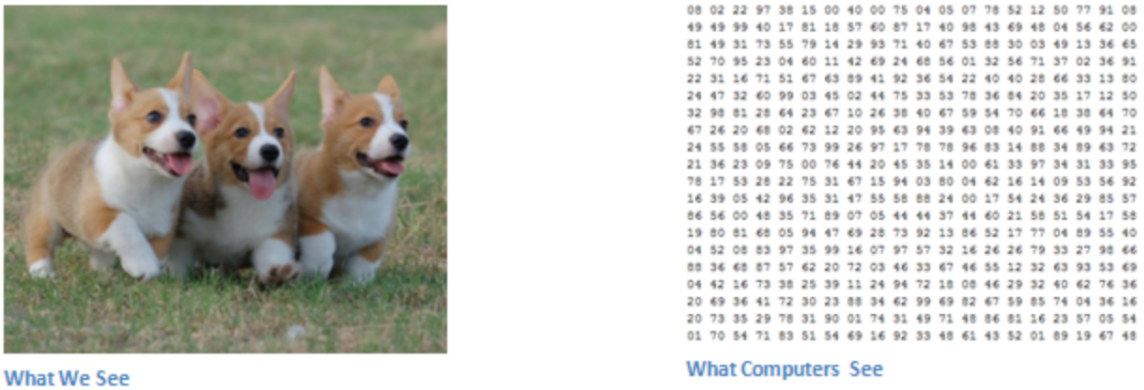


Figure 1: The left image shows input for human vision input compared to computers. Subtle variation of the left image (that may be imperceptible for human vision) radically changes the input for the computer. Image extracted from [1].

There are different approaches for performing classification tasks (deciding wether an image contains an object or not). Former approaches were based on engineered features, which means that they don't learn the features by themselves, but that the features are designed by who implements the algorithm. It was not until 2012 that a type of statistical model, Convolutional Neural Networks (CNNs), had its breakthrough by winning the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [2]. The model proposed, called AlexNet [3], laid the foundation of modern research in computer vision in general. From that point onwards,

CNNs achieved the best results in almost all important competitions, including the previously mention ILSVRC [2] and PASCAL Visual Object Classes [4]. The focus on CNNs continued, with some great advances such as Microsoft's model ResNet [5] which won ILSVRC 2015 with an incredible error rate of 3.6%. To picture this, just consider that human vision generally achieves between 5% and 10%. This outstanding resulte, and the complexity involved in performing even better results, made classification seem as a (partially) solved problem, and the focus turned to object detection. This is the task of computer vision in which this thesis will focus on, that deals with detecting instances of semantic objects of a certain class (such as people, buildings or cars) in images.
The main implementation of CNNs for object detection is Regions with CNN (R-CNN), which was presented in 2013 [6]. It outperformed by more than 30% the previous best result on VOC 2012 [4], and introduced the standard way of performing detection. Region based CNNs will be described in detail in chapter 2: Basics.
Several improvements were made to R-CNN [6], namely Fast R-CNN [7] (April 2015) and Faster R-CNN [8] (June 2015). The focus was on the performance metric (frames per seconds), but the concepts stayed similar to R-CNN. Throughout 2016, You Only Look Once(YOLO) [9] and Single Shot MultiBox Detector(SSD) appeared, also region based CNNS, became the standard in object detection, mixing both high accuracy and real time performance. SSD300 was the first real-time detection algorithm that could achieve more than 70% accuracy.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	$\sim 1000 \times 600$
Fast YOLO	52.7	155	1	98	448×448
YOLO (VGG16)	66.4	21	1	98	448×448
SSD300	74.3	46	1	8732	300×300
SSD512	76.8	19	1	24564	512×512
SSD300	74.3	59	8	8732	300×300
SSD512	76.8	22	8	24564	512×512

Figure 2: Summarizes the main object detection algorithms with their accuracy and performance in PASCAL VOC 2007 [4] test dataset. The suffix 300 or 500 refers to the resolution of the images used for training. Performance is measure in frames per second (FPS) and accuracy is the mean average precision(mAP), that will be explained in chapter 2: Basics. SSD300 [10] presents a formidable tradeoff between accuracy and performance. Results taken from SSD [10].

1.1 Applications

There are diverse fields in which object detection is applied, such as face detection, surveillance cameras, vehicle detection, manufacturing industries, robotics and security.

Object detection holds potential to modify human life in considerable ways. Different areas could be improved by this technology. For example, face detection (task of identifying human faces in digital images). It is a special case of object detection that can be used for:

- facial motion capture for converting the movement of a human face to be able to use it for computer graphics.
- facial recognition for security systems and human computer interfaces.
- Photography: auto focus and gesture detection.
- Marketing: calculate via the face which culture the person belongs to and select relevant advertisements.

Other example of object detection applications is people counting. Statistics algorithms analyze quantitative data on how people behave. Being able to detect people would automate the whole data gathering and applied in plenty of situations. For example, to count the number of people that access a store or marketing effectiveness.
One of the main driving force of object detection is vehicle detection. It is already used by Advanced driver-assitance systems (ADAS) for alerting the driver of potential problems and avoiding collisions. It is also used by autonomous cars, which heavily rely on object detection. According to [11], "One of the major open challenges in self-driving cars is the ability to detect cars and pedestrians to safely navigate in the world."
Other applications constitute manufacturing industry (robots which are able to detect objects and interact with them), and detecting suspects in videosurvilliance cameras. It is highly possible that object detection will spread to additional areas, such as Internet of Things (IoT) and mobile devices, but they require more computing power and efficient algorithms.

1.2 Why weren't CNNs introduced before?

The rise of CNNs usage is based in two important factors: increase of performance and availability of data. CNNs were not massively used until neural networks could be trained on GPUs (2005). Availability of annotated data has been increasing throughout the years, specially with known datasets such as the ones provided by VOC [4] and ILSVRC [2]. As performance improved throughout the years at a bigger pace than availability of annotated data and is deeply related to the hardware aspects, the scope of this work relays on availability of annotated data. It is important te note that when there is enough training data available, even simple models can address image classification problems with outstanding results. L. G. Valiant proved in a Theory of the Learnable (1984) [12] that if there is a finite number of functions N, then every training error will be close to every test error once you have more than log N training cases by a small constant factor. If the training error is close to the test error, it means that there is no overfitting, therefore the network will have high accuracy when applied to data different than the training one.

1.3 Approaches when annotated data is insufficient

The previous theorem helps to understand the importance of availability of data. This section will mention the current approaches used when annotated data is scarce or not available.

1.3.1 Reusing a previously trained CNN

CNNs are constrained to detect the classes provided by the training datasets, which makes it harder to reuse them for detecting new classes. A key technique frequently used to reuse that training is called transfer learning, that allows a model to be fine-tuned to detect other classes, without the need of training the network again from scratch. This technique is convenient and widely used but highly depends on the shared characteristics between the original classes and the new ones. Even though this approach usually leads to great results [13], annotated data is still needed for fine-tuning, which makes transfer learning not always applicable.

1.3.2 Training CNN from scratch

In addition to the difficulty of reusing a previously trained network, the main obstacle is that data needs to be annotated: for each image in the training dataset, additional information is required(for object detection, the bounding box of the objects present in the image, and the class it belongs to). There are also scenarios in which annotated data in large quantities ([13] suggests tens of thousands) is insufficient or that need high specialization for detecting certain classes. For example, a CNN trained for detecting airplanes would be suitable for a specific manufacturer, but will not be able to detect an airplane of other manufacturer as negatives. The solution (and current aproach) is to obtain thousands of images and manually label them. While is not imposible, it tends to be expensive, and hard for the images to represent all the possible conditions that could be present in addition to the object (for example, different light and weather conditions for the airplane model).
The approach taken in this thesis, as in viewpoint estimation [14], is to generate images synthetically and use them for training a CNN for object detection. Although there are similarities between [14] and this work, the problem to solve varies drastically: while in [14] the objective was to estimate the viewpoint, here the goal is to detect the bounding box of the class of interest. As images are synthetically generated, the number of images that can be generated is arbitrary. The key point is to modify the conditions of a 3D scene (light, camera position and shadows) to generate images as realistic as possible. A fundamental aspect of synthetic data generation is that images are annotated automatically (obtaining the annotations for each image via the scene), so manual labelling and obtaining large amount of images for training is not longer required. The same generated dataset can be used for different tasks, for example viewpoint estimation [14] and object detection. This thesis will propose different strategies for synthetic image generation to resemble real world images and contrast them quantitatively

with real data. This involves a deep evaluation of different ways of generating synthetic data, focusing in the characteristics and tradeoffs. It will allow to answer the main question pursued in this research work: could an object recognition algorithm, trained solely in synthetic data, obtain a high detection accuracy in real world images?

This work starts in Chapter 2: Basics, with a detailed overview of object detection, explaining what it is, the difficulties involved and how did the algorithms evolved to the settlement of CNNs as the state of the art approach. Chapter 3: System Overview covers the proposed system, not only how it works internally but also how it interacts with other systems. Chapter 4: Experiments will talk about the decisions behind the experiments designed, and what will be tested with them. Chapter 5: Results gathers the quantitative results of the experiments. Chapter 6: Discussion shows a detailed analysis of the experiments, considering the quantitative results and a qualitative analysis (via images and visualisation tools) contained in the experiments. Finally, Chapter 7: Conclusions states the conclusions derived from this work and proposes open questions for future work.

2 Basics

2.1 Classification

Classification can be formulated as: given an input array of pixels which represent an image, determine with certain probability to which object category it belongs ($f: Pixels \rightarrow Categories$). It involves a combination of two tasks:

1. Given an image represented by pixels, identify the features that it exhibits ($f: Pixels \rightarrow Features$)
2. Correlate the features to identify the presence of an object and to which category it belongs ($f: Features \rightarrow Category$)

2.2 Localization

Localization includes recognizing the category of an object (classification), and identifying its spatial extent with a tight bounding box that covers all its visible parts [15]. This task is more difficult than classification, the output now includes a bounding box surrounding the object. Implicitly, this involves an algorithm to propose multiple bounding boxes, and measure the distance (L2) of each prediction (that contains the object) with the ground truth.

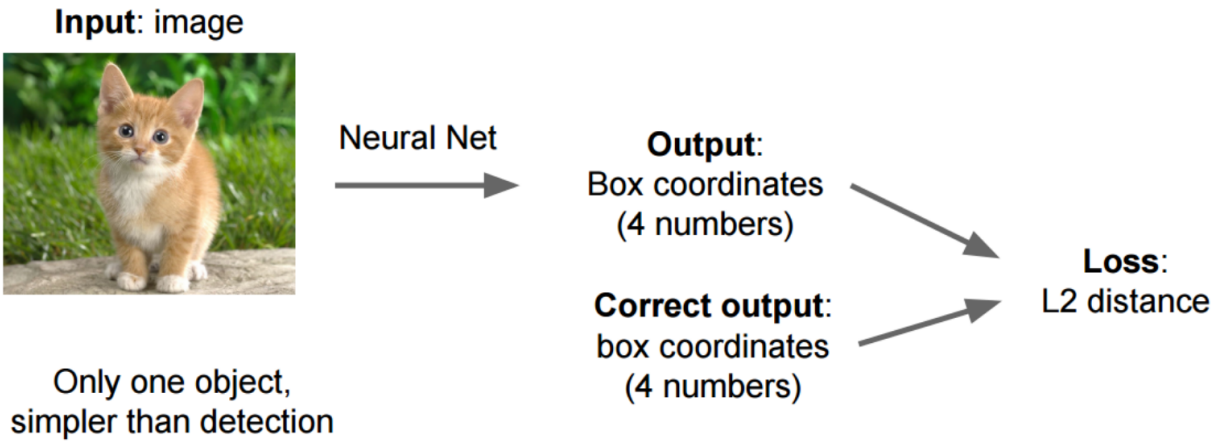


Figure 3: Visual example of the task of localization. Taken from [16].

2.3 Object Detection

The goal of object detection is to detect all instances of objects from a known class, such as people, cars or faces in an image. It involves classification (section 2.1) and localization (section 2.2). There is a very large number of possible locations and scales at which object can be present and that need to somehow be explored. It has a greater complexity than localization because the detector must be able to also differentiate between different instances, that in some cases may be similar.

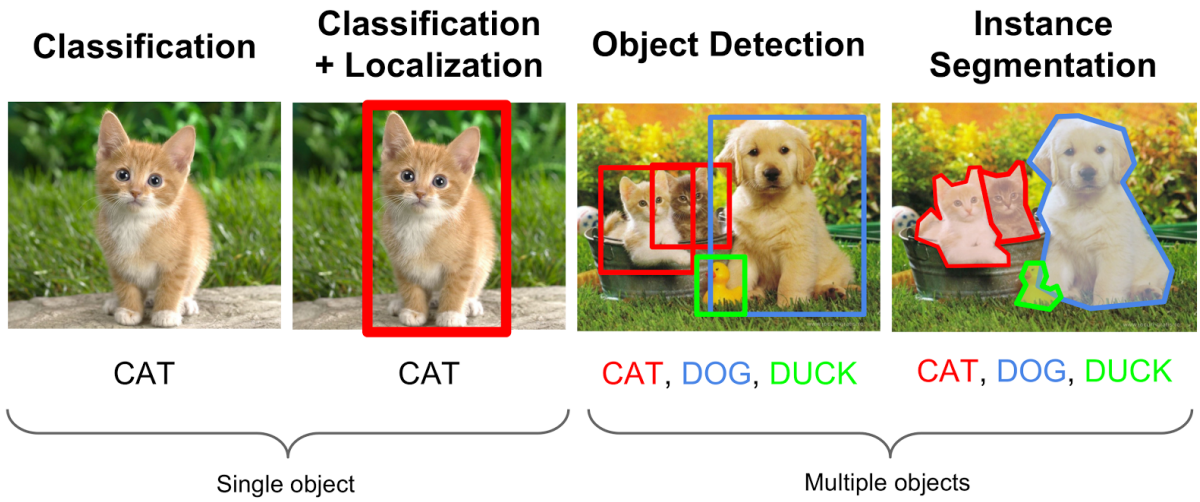


Figure 4: Main computer vision tasks pursued by researchers. From left to right, there is an increase in difficulty, and each task use techniques implemented by the previous ones. Image extracted from [17].

2.4 Feature detection

The concept of feature detection refers to the ability of identifying certain points in an image that describe the image contents (such as edges and corners) [18]. Local features (or interest points) can be defined as region of pixels that are near each other and associated with one or more properties.

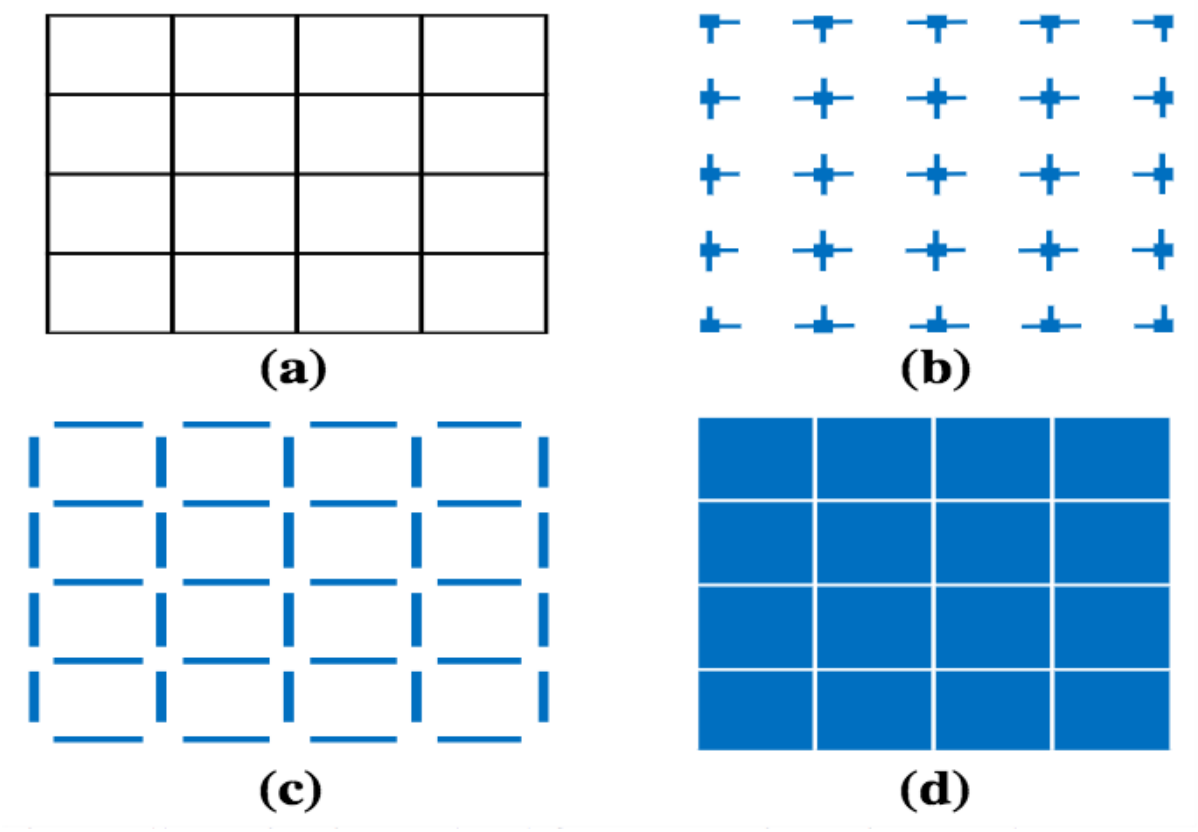


Figure 5: Illustrative image local features (a) input image, (b) corners, (c) edges and (d) regions. Extracted from [18].

A feature detectors refers to the algorithm or technique that extracts local features and outputs them to other processing stage that describes its components (feature descriptor). A local feature has a spatial extent because of the local pixels neighbourhood (they correspond to an object or part of an object) [18]. Ideal features should contain these qualities [18]:

- Distinctiveness: The pattern underlying the feature should be clearly distinguished from other parts of the images.
- Locality: Features should be local, so to reduce the changes of getting occluded.
- Quantity: The number of detected features should be sufficient to describe the image.
- Efficiency: Features should be identified in a short time (specially for real time applications).
- Repeatability: Given two frames that exhibit a common pattern identified as a feature, it should be identified in both of them (even with occlusions).
- Invariance: In scenarios where there is a change in size, scale or rotation but still represents the same feature, it should be detected as such.

– Robustness: When there is a small deformation (noise, blur or small color variations), the feature should still be detected as such.

The concept of identifying features is independent of the algorithm being used. The procedure on how features are defined, and how feature extraction works is what varies between algorithms. Feature detectors define features beforehand and identify them on the input data (therefore there is no training involved).

Features Detector	Invariance			Qualities			
	Rotation	Scale	Affine	Repeatability	Localization	Robustness	Efficiency
Harris	■	-	-	+ + +	+ + +	+ + +	+ +
Hessian	■	-	-	+ +	+ +	+ +	+
SUSAN	■	-	-	+ +	+ +	+ +	+ + +
Harris-Laplace	■	■	-	+ + +	+ + +	+ +	+
Hessian-Laplace	■	■	-	+ + +	+ + +	+ + +	+
DoG	■	■	-	+ +	+ +	+ +	+ +
Salient Regions	■	■	■	+	+	+ +	+
SURF	■	■	-	+ +	+ + +	+ +	+ + +
SIFT	■	■	-	+ +	+ + +	+ + +	+ +
MSER	■	■	■	+ + +	+ + +	+ +	+ + +

Figure 6: Comparison of feature detectors. It is important for them to be invariant to rotation, scale or affine transformations. At the right, how effective is the feature detector for each expected quality. Image extracted from [18].

Feature descriptors were the prevalent feature extraction algorithm choice until the advent of Convolutional Neural Networks (CNNs). CNNs identify and extract features by themselves, without the need of specifically implementing a feature detector algorithm. They learn through the training data, and generate features descriptors from it automatically. The feature detector not being explicitly implemented explains why CNNs are so difficult to understand, with advances due to empirical evidence rather than theoretical facts [5, 13]. This will be explained more in detail in section 2.5: Convolutional Neural Networks (CNNs) and section 2.9: Historical Progress.

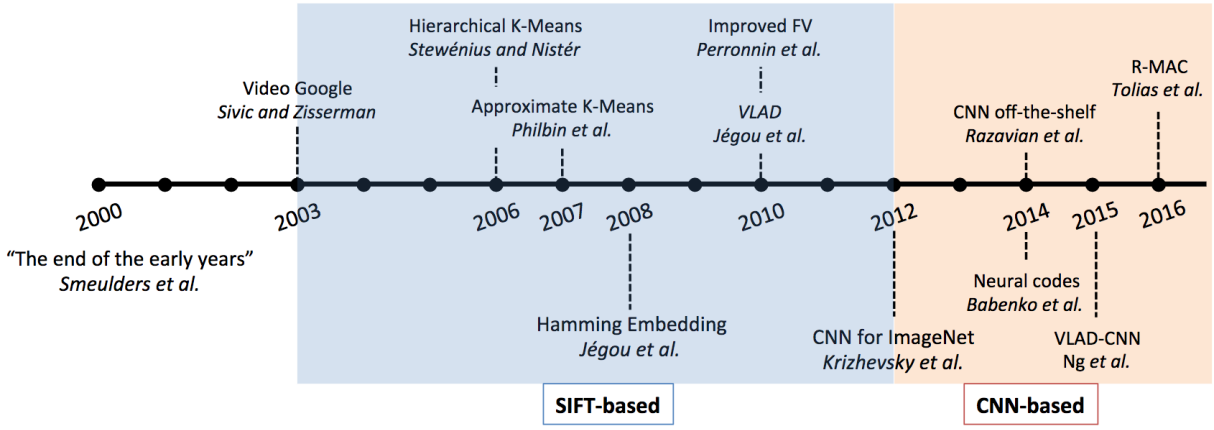


Figure 7: .

2.5 Convolutional Neural Networks (CNNs)

Vision is one of the most incredible skill that human posses. In only fraction of seconds, we can identify objects within our field of vision, without even thinking about it. Not only it is possible to detect objects, but also to identify certain characteristics (lines, curves and shape) that distinguish them from the background they are surrounded by. Convolutional Neural Network is a model built by studying the way our brain process images. They constitute a type of multilayer perceptron (MLP): layers of neurons connected in a way that information flows through the network, just like our brains. The most important aspect about CNNs, apart from the different possible network architectures, is the weights of the connection between neurons, which resembles the synapsis occurring in our brain. The weights are caled parameters and trained in each new iteration. What differentiate CNNs from MLP is that CNN fit exactly with image vision tasks because they exploit properties that images exhibit (pixels which are near each other usually constitute the same feature) and mimic hierarchichal learning from how the human vision works. The outcome is better efficiency (the size of input is reduced in each layer) and less number of parameters (or weights to train). CNNs work by extracting succesively low level features in the first layers and then building a more advanced understanding in deeper ones. For example, the early layers extract edges, and subsequent layers will combine those edges to detect features such as the shape of the object. Finally, the last layers will combine high level features and map them to a specific class.
In general, a CNN will contain three different types of layers:

– Convolutional layer: Layer that contains a set of filters with a small receptive field. A convolution (dot product) is made between the input and each filter in every forward pass.

– Pooling layer: It is generally located after a convolutional layer, reducing the spatial dimensions (width and height) of the input for the next layer.

– Fully connected layer: Usually placed at the end of the CNN, they map the non-linear combinations of features to the output which decides to what class the input image belongs.

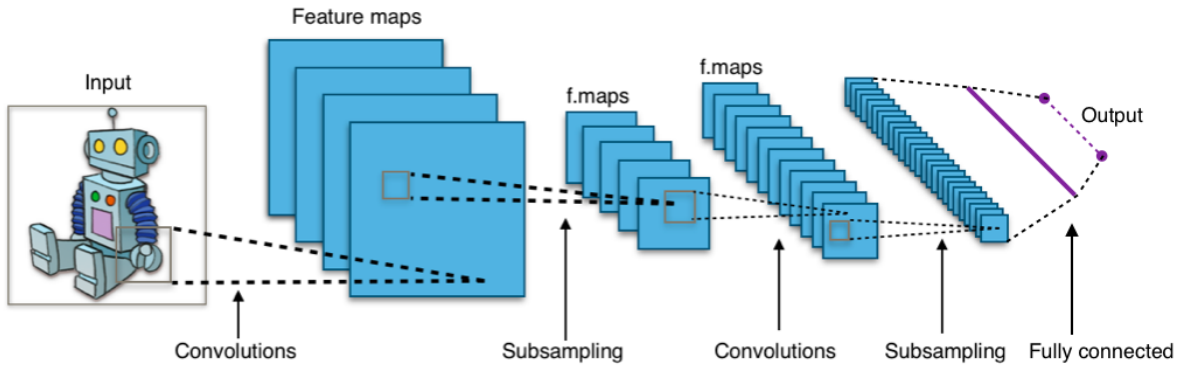


Figure 8: A high level perspective of how CNN process a region of the image. Notice the presence of convolutions, and pooling (subsampling). At the end, there is usually one or multiple fully connected layers. Image extracted from [19].

2.5.1 Convolutional layer

What distinguishes CNNs from other neural networks is the convolution operation, which is done by the convolutional layers. Multiple filters are contained in the convolutional layers, and each filter detects different low level features of the input. Each filter is used in a convolution with the input image, which is in other words a matrix multiplication (as shown in fig. 9).

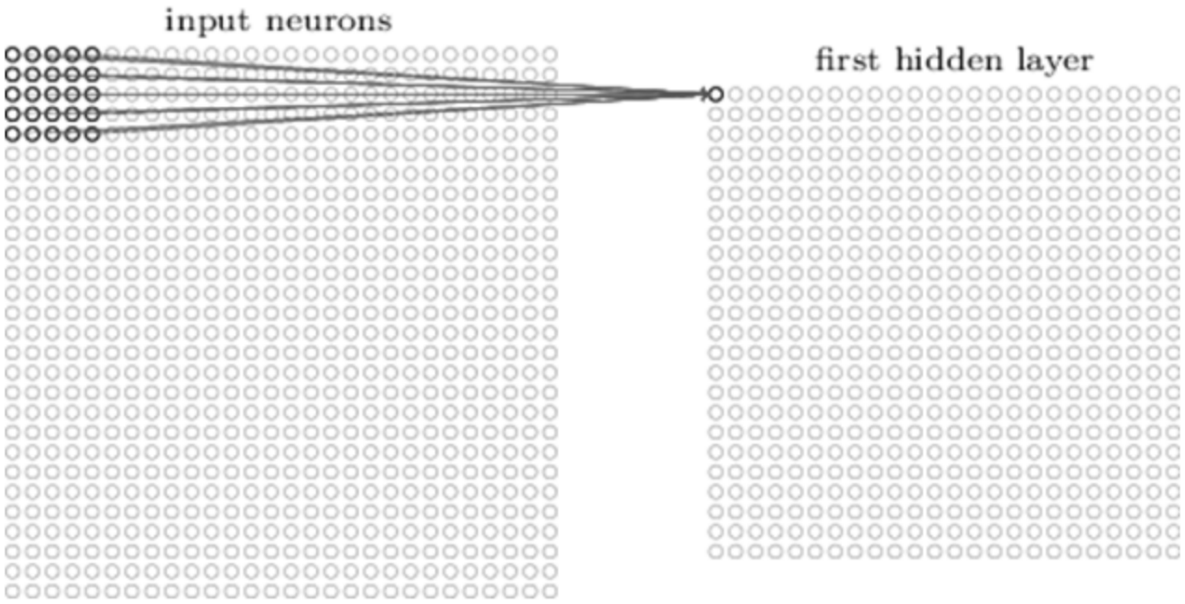


Figure 9: Visual representation of how convolutional layers work. Notice the size of the filter (5 x 5). The first hidden layer size need to match the output of the convolutional layer. Image extracted from [1].

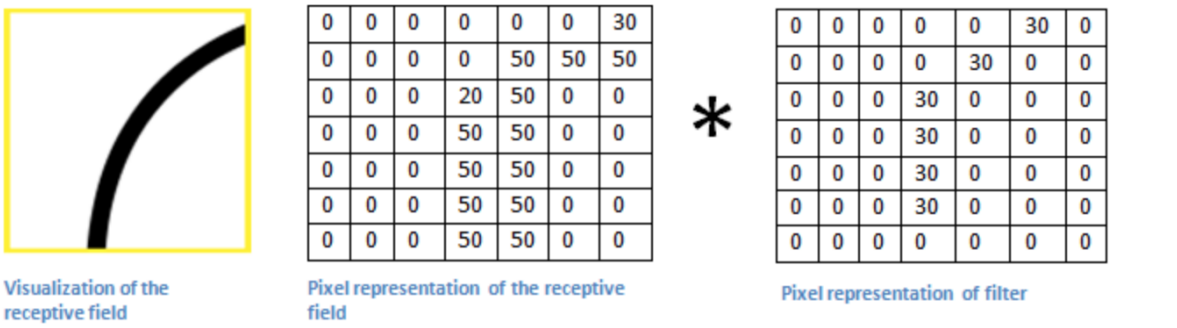


Figure 10: A specific region of the image is feeded through the network. At the right, a convolution is made between the pixel representation of that region and the pixel representation of the filter. The result of this operation results in a high number, which means that this region of the image produces a high activation map on that filter. Notice that this is just one filter of the layer: there are multiple filters and the output of them is then used by successive layers. Also, as this filter detects edges, it is contained on the first layers of the network. Image extracted from [1].

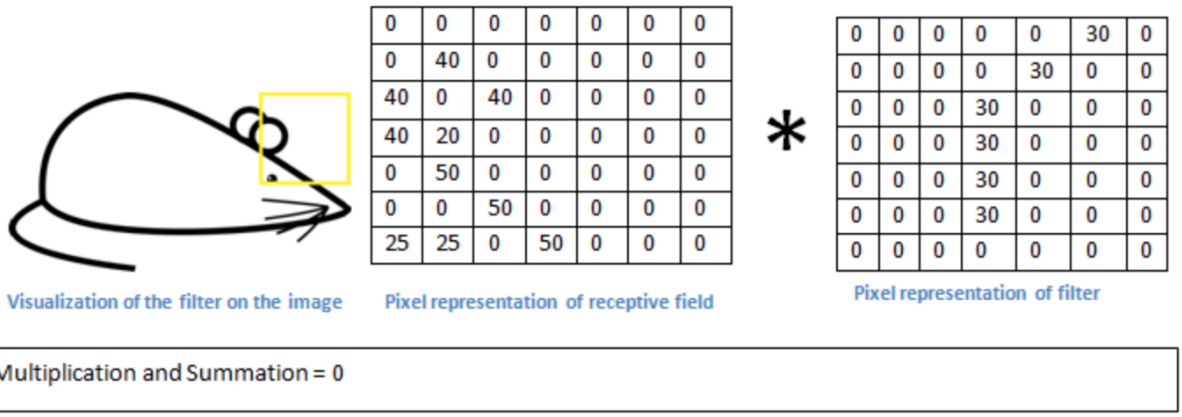


Figure 11: In this case, the convolution between the pixel representation of the region and the filter gives a small value. This means that the filter, which detects certain type of edges (see Figure fig. 10) couldn't detect the edge in the input region provided. Image extracted from [1].

The main benefits of applying convolutions is that:

- By applying a smaller filter, the output is reduced, which leads to less weights to train and less mathematical operations (better performance).
- The different filters can share weights, which will also result in better performance.
- If the input of the layer changes, the output will change proportionally.

2.5.2 Pooling layer

The pooling layer reduces the spatial dimensions of the image (it is also called downsampling). It is true that information is lost, but this is a positive aspect because it implies less mathematical operations (better performance). The reasoning of why pooling works relates with the assumption of feature locality: pixels which are near each other will probably belong to the same feature. Therefore, a sample contained in a group of pixels will already represent the feature, avoiding to analyse all the near pixels. In other words, the important aspect is the relative location of the feature to other features rather than the specific location of the feature in the input. Pooling also reduces overfitting, as a direct consequence of reducing the number of parameters (as there is less information, it is harder for the network to overfit on the dataset it was trained on). A commonly used pooling filter is the max pooling, because it is a non-linear function (therefore, it can compute more functions than for example convolutions, which is basically a linear matrix product).

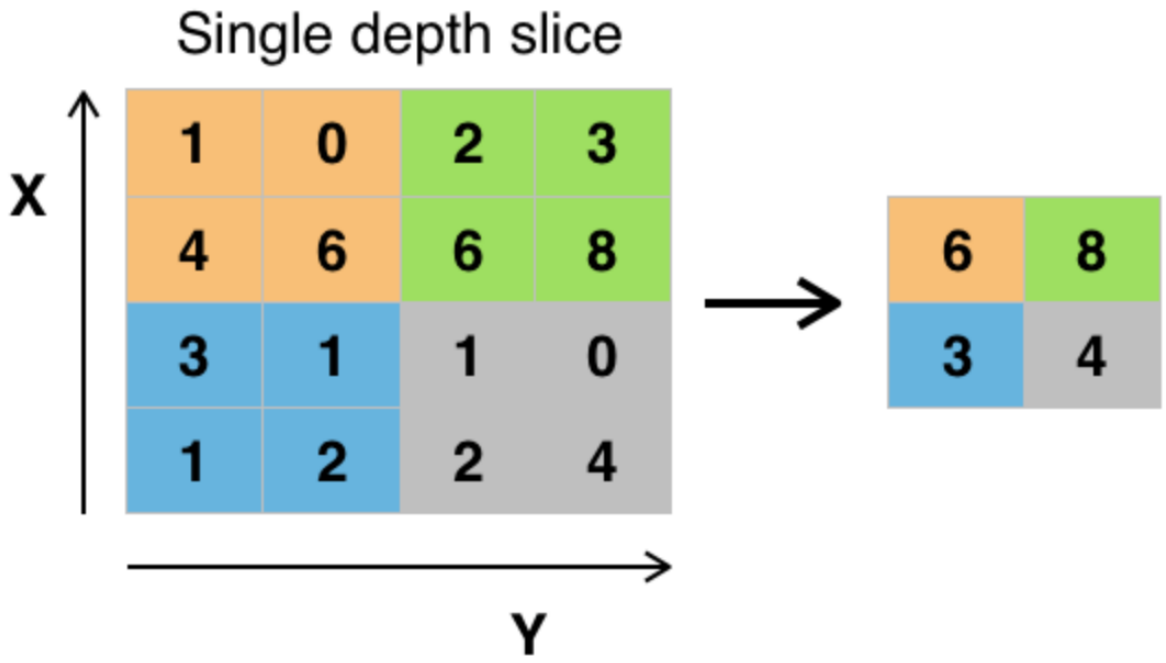


Figure 12: In this case, the type of pooling used is max pooling with filter size 2x2. Max-pooling partitions the input image into a set of non-overlapping rectangles and outputs the maximum value for each such sub-region . Notice how the spatial size of the input is reduced. The size of the pooling filters is usually small, to avoid loosing high amount of information. Image extracted from [20].

2.5.3 Fully connected layer

The fully connected layer is a layer in which every input neuron is connected to every output neuron (a common layer of a multilayer perceptron). The output from the convolutional and pooling layer will represent high levels features ($f: Pixels \rightarrow Features$) and the fully connected layer will receive those features and decide to which class it belongs ($f: Features \rightarrow Category$). The layer learns by the training data supplied, and adjust the weights accordingly (same as convolution and pooling layers). Though convolutional layers could be connected directly to the output, fully connected layers allows to learn non-linear combinations of the input (in other words, the features). This is because the activation function is usually the sigmoid, and not a linear function (like g(z) = z). When the activation function used in non-linear the network is able to represent more complex functions.

2.6 Known datasets

2.6.1 PASCAL VOC

The PASCAL Visual Object Classes (VOC) challenge is a benchmark in object detection, providing the vision and machine learning communities with a standard dataset of images and annotations, and standard evaluation procedures [4]. It started in 2005 with 4 classes: bicycles, cars, motorbikes and people, containing 1578 images with 2209 annotated objects. It was in 2007 that PASCAL VOC consolidated (along with ILSVRC) as the standard competition in image vision tasks. The 2007 dataset consisted of 9936 images containing 24,640 annotated objects, with 20 classes:

- Person: person
- Animal: bird, cat, cow, dog, horse, sheep
- Vehicle: aeroplane, bicycle, boat, bus, car, motorbike, train
- Indoor: bottle, chair, dining table, potted plant, sofa, tv/monitor

PASCAL VOC also introduced mean average precision (mAP), the standard measure of accuracy in object detection which is explained in detailed in section 2.7: Accuracy.

2.6.2 ILSVRC

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a "benchmark in object category classification and detection on hundreds of object categories and millions of images" [2]. It started at 2010 for being able to compare different algorithms in object recognition tasks, and became the standard benchmark for large-scale object recognition. It consists of two components: a public dataset and an annual competition. It developed novel crowdsourcing approaches for collecting large-scale annotations, described in [2]. ILSVRC contains 1000 object classes, with more than 1 million images. For object detection category, images are collected from Flickr using scene-level queries. The training dataset contains images with annotations for objects that are present and negative images that don't contain any instance of some object categories. Similar to VOC, ILSVRC uses precision and recall. The threshold chosen to determine if a detection is good is also 0.5. However, when detecting difficult objects (which PASCAL VOC labeled as "difficult" and ignored them during evaluation), ILSVRC adjusts the threshold (considering that 50% overlap is not precise for small objects that occupy few pixels):

$$\text{Threshold} = \min(0.5, \frac{width * height}{(width + 10)(height + 10)}) \tag{1}$$

2.7 Accuracy

PASCAL VOC [4] mentions that it is incorrect to just consider the percentage of correctly classified examples, because the prior distribution over classes is significantly nonuniform. This is very important for object detection, because the detector will encounter thousands of negative (non-class) examples for every possitive example. The accuracy of the object detector should consider the tradeoff between different type of information retrieval errors. For detection tasks, PASCAL VOC requires the participants to submit a bounding box and a confidence level for each detection. This allows to measure the accuracy, without asigning an arbitrary cost function.

2.7.1 Recall and precision

In information retrieval algorithms, recall and precision are the main metrics used for comparison. Precision is defined as the fraction of relevant instances among the retrieved instances and recall is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. PASCAL VOC defines recall as the proportion of all positive examples ranked above a given rank and Precision as the proportion of all examples above that rank which are from the positive class [4].
In a mathematical perspective, precision and recall can be defined as:

$$\boxed{\begin{aligned}\text{Precision} &= \frac{\text{Number of correctly detected instances}}{\text{Total number of instances}} \\ \text{Recall} &= \frac{\text{Number of correctly detected instances}}{\text{Total number of detected instances}}\end{aligned}} \tag{2}$$

Precision and recall are calculated by obtaining the following values:

- True positives (TP): Detection of an object when it is there.
- False positives (FP): Detection of an object when it isn't there.
- False negatives (FN): Not detecting an object when it is there

$$\boxed{\begin{aligned}\text{Precision} &= \frac{TP}{TP+FP} \\ \text{Recall} &= \frac{TP}{TP+FN}\end{aligned}} \tag{3}$$

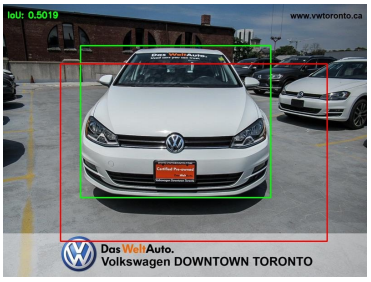
Both depend on relevance, and defining the concept of relevance is a difficult topic in object detection. It implies determining if an instance retrieved is correct or incorrect. For classification, it is straightforward to define relevance: the object being part (or not) of a particular class. However, in object detection the output is a bounding box, therefore relevance includes not only classification but also comparing the proposed bounding box to the object's real bounding box.

2.7.2 Measuring relevance

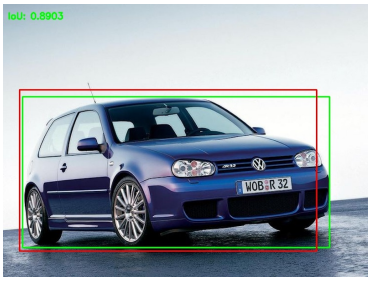
The standard concept of relevance (proposed by PASCAL VOC [4]) is called Intersection over Union (IoU). It measures the overlapping of the predicted bounding box with the ground truth bounding box and outputs a value in the interval [0, 1]. A relevant prediction in object detection asserts that:

- The object belongs to the class predicted.
- The IOU between the predicted bounding box and ground truth is greater than a threshold provided.

(a) Not accurate detection: IoU = 0.15



(b) Good detection: IoU = 0.5



(c) Almost perfect detection: IoU = 0.89

Figure 13: Comparison between different IoU. Notice the intersection area of the estimation (red) with the ground truth (green). The standard is to consider 0.5 as a good detection (according to PASCAL VOC [4]). Examples taken from the test dataset.

2.7.3 Computing precision-recall curve

The algorithm for calculating precision and recall was proposed by PASCAL VOC [4]. The IoU introduced before makes it possible to obtain the precision-recall curve. A collection of images are given as input to the detector and it outputs a list of predictions (each one containing class, confidence and bounding box). Each prediction output by the detector is compared with the ground truth. Multiple detections of the same object in an image are considered false detections e.g. 5 detections of a single object count as 1 correct detection and 4 false detections. The pseudocode algorithm (section 2.7.3: Computing precision-recall curve) shows how precision and recall are calculated.

Algorithm 1 Calculates precision and recall based on a given threshold

procedure PRECISION_RECALL($threshold$)
    $TP \leftarrow 0$
    $FP \leftarrow 0$
    $FN \leftarrow 0$
    for each $prediction$ in $predictions$ **do**
        $IoU = calculate\_IoU(prediction,\ ground\_truth)$
        if $IoU > threshold$ **and** object not detected yet **then**
            $TP \leftarrow TP + 1$
        else
            $FP \leftarrow FP + 1$
        end if
    end for
    for each $image$ in $images$ **do**
        if no prediction made for image **then**
            $FN \leftarrow FN + 1$
        end if
    end for
    $precision \leftarrow \frac{TP}{TP+FP}$
    $recall \leftarrow \frac{TP}{TP+FN}$
    Return ($precision$, $recall$)
end procedure

Precision and recall depend on the threshold value given. A detection is considered good when the threshold is 0.5, metric proposed by PASCAL VOC [4] and also used in ILSVRC [2]. Nonetheles, it is recommended to try different thresholds depending on the problem (for example, medical systems will require a higher threshold value). The values of precision and recall with different thresholds is called precision-recall curve.

2.7.4 Computing mean average precision

The mean average precision (mAP) used in PASCAL VOC is called interpolated average precision (Salton and Mcgill 1986) [4]. The average precision summarises the shape of the precision/recall curve for a particular class, and is defined as the mean precision at a set of eleven equally spaced recall levels [4]:

$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} p_{interp}(r)$

The mAP is then defined as calculating the mean of the average precision(AP) of all classes:

$mAP = \frac{1}{\sharp classes} \sum_{c \in \{classes\}} AP(c)$

2.8 Synthetic Image Generation

Acording to McGraw-Hill Dictionary of Scientific and Technical Terms [21], synthetic data is "Any production data applicable to a given situation that are not obtained by direct measurement". In this work, the images per see will be qualified as "synthetic" or "real", though those words just describe the way of obtaining them.
The lack of annotated datasets lead to explore new approaches of obtaining data. There are basically two different ways of generating synthetic data:

- Augmenting real images by modifying certain aspects (such as horizontally flipping, random crops and color jittering). Augmentation was specially proposed by AlexNet [3] in where they affirm that "fancy PCA could approximately capture an important property of natural images, namely, that object identity is invariant to changes in the intensity and color of the illumination". This reduced the top-1 error rate by over 1% in the competition of ImageNet 2012 [2].

- Create completely synthetic datasets from scratch. There are different ways to accomplish this, that will be discussed next.

2.8.1 Generating images via a game engine

One valid and promising approach would be to obtain screenshots from game engines and include those in the training dataset. For example, for cars it would be valuable to create a scene with a city (or play a driving game) and drive the car around while modifying the camera position. The AI could just move randomly through the game and take an arbitrary number of images while driving. Unfortunately, this isn't applicable to every type of problem, and generating a 3D scene like this would take a long time. [22] implents this type of synthetic generation tool with UnrealCV engine for indoor scenes.

Figure 14: A game engine that allows to create realistic images. Modifications can be introduced, for example to the camera position (left and center image) and the sofa color (right image). Image extracted from [22].

2.8.2 Blending objects into the scene

Instead of creating such a complex scene, a simple scene can be designed. This approach is proposed by [23] for training an object detector with synthetic data in indoor scenes. It "superimpose 2D images of textured object models into images of real environments at variety of locations and scales", and this superposition can be done in different ways. The easiest case is to just overlay the object image on the background image: the problem is that the position of the object would be incorrect, as if it were floating over the image. They go a step further and perform a semantic analysis to correctly blend the synthetic object into the scene. The analysis required for placing the object degrades the performance of synthetic data generation, so it is a tradeoff between how realistic images should be and the time that required to generate them. Furthermore, 2D images of the object are used instead of a 3D model, so there are less possibilities to modify the characteristics of the object.

Figure 15: The red bounding boxes surround the objects that are blended to the indoor scene. This tool produces highly realistic images, but with performance costs. Image extracted from [24].

2.8.3 Proyecting the object on the image

The last approach inspected, similar to [23] but without blending the object into the exact position, is proposed by viewpoint estimation using synthetic images [14]. It is the same concept, with the advantage that there are no penalities in performance but at the same time the images are not as realistic. That approach proved to be successful for [14]. In that work, the author creates an abstraction over Blender, an open-source computer graphics software,

by using Blender's python API [25]. The tool requires a scene, a 3D model and a configuration file. Via the configuration file it is possible to modify different render parameters of the scene at runtime: for example, the light conditions, color of the 3D model, camara position and lights characteristics.

Figure 16: Composition of 160 images generated by viewpoint estimation synthetic dataset generation tool. Notice the presence of a ground and sky, the car color and the light conditions. Image extracted from [14].

2.9 Historical Progress

2.9.1 Overview

Former approaches to image vision tasks were based on feature detection, which consists in specific patterns or specific features which are unique, can be easily tracked and can be easily compared. The definition of a feature is not generic: it depends on the type of images analysed. Features usually describe aspects such as color, shape, texture or motion. There are different examples of feature descriptors algorithms, such as edge orientation histogram or shape context (2000) (describing shapes that allows for measuring shape similarity and the recovering of point correspondences).
Convolutional neural networks (CNN) is a statistical method in computer vision that models how the brain process images through the visual cortex. They are commonly used for computer vision tasks. They group neurons in each layer, which are in charge of identifying different features of the input images and they model features in a hierarchical way. These means that in the first layers, CNNs are able to identify low level features such as edges and curves, while in deep layers they build up more complex features. This way of learning can be possible because pixels in a specific region of an image tend to be similar to each other. In other words,

CNNs exploit spatial dimensionality that images exhibit. This section will describe the relevant contributions which lead to the settlement of CNNs as the standard choice for object detection tasks.

2.9.2 Hubel and Wiesel 1962

There are special regions of cells in the visual cortex that are in charge of processing visual input. This idea was shown by Hubel and Wiesel in 1962 [26], where they demonstrate that certain cells responded (or fired) only in the presence of edges of a certain orientation. They showed that different groups of neurons fired when exposed to vertical, horizontal or diagonal edges. They also found out that all of these neurons were organized in a columnar architecture and that together they were able to produce visual perception. This idea of specialized components inside of a system having specific tasks (the neuronal cells in the visual cortex looking for specific characteristics) is one that machines use as well, and is the basis behind CNNs. While CNNs continued to be studied since the work of Hubel and Wiesel, the lack of computing power and annotated data restricted their usage.

2.9.3 Yann LeCun (1989)

One of the first empirical applications of neural networks was proposed in handwritten recognition with a back propagation network. Yann LeCun et al showed that a large back-propagation (error is propagated backwards to adjust weight values) network can be applied to real image-recognition problems [27]. In contrast with other algorithms in which the input were the feature vectors, the neural network was fed directly by images of handwritten digits. This work already showed that the architecture of the network "strongly influences the network's generalization ability" [27]. Handwritten recognition became the standard classification task, because it is easily described by an array of either white or black pixels and maps from image space to category space.

Figure 17: Examples of normalized digits from the testing set. Image extracted from [27].

2.9.4 SIFT (1999)

One of the most used feature extraction algorithm is Scale-invariant feature transform (SIFT, 1999) [28], which implements a feature detector with scale and orientation invariance of the image. It detects sets of point of interest in an image and for each one of them computes a histogram-based descriptor with 128 values. SIFT involves different algorithms for each step: from extracting features (computing the gradient for each pixel), indexing and matching the features to the ones on the database to outliers detection and probability analysis. The downsides of SIFT are that it is computationally expensive and exhibit difficulties of pattern extraction with changes of light conditions.

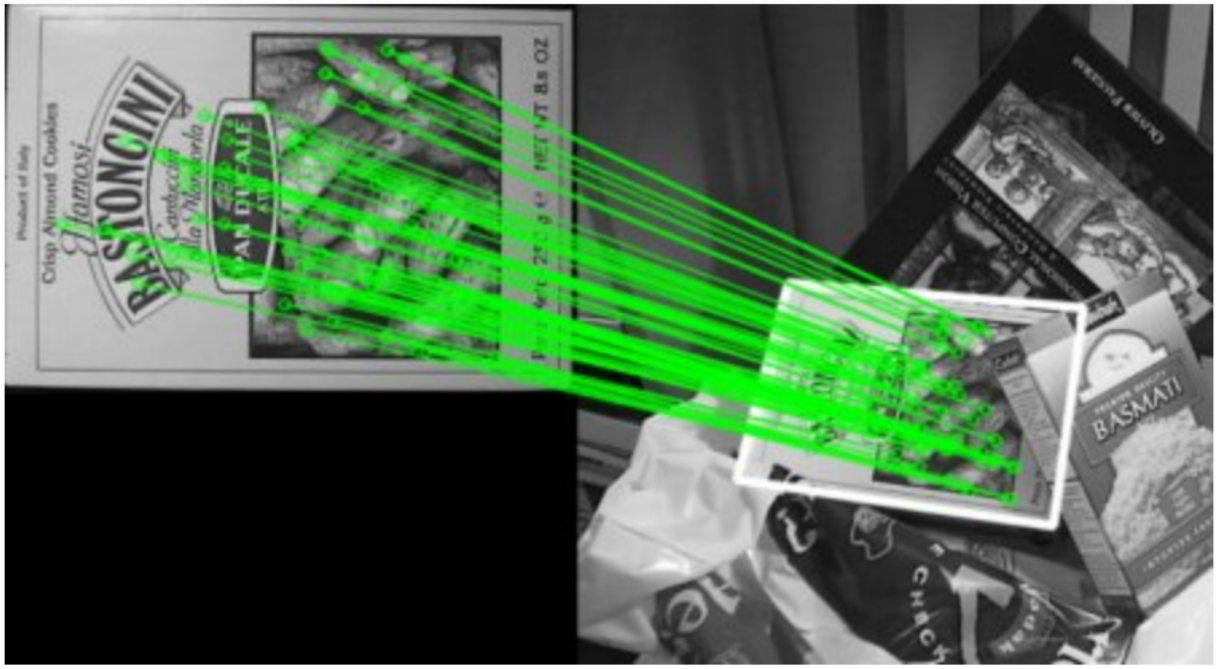


Figure 18: An example of applying SIFT detector. It looks for the points of the image at the left in the image at the right.

2.9.5 MSER (2002)

Other well known feature extraction algorithm is maximally stable extremal regions (MSER, 2002) [29]. It gathers from an image a number of co-variant regions called MSER, a stable connected component of some level sets of the image. Elliptical frames can be attached to MSERs by fitting the ellipses to the region.

(a) Input image. (b) Extracted MSERs. (c) Fitted ellipses.

Figure 19: An example of using MSER feature extraction algorithm.

2.9.6 ImageNet Classification with Deep Convolutional Networks (2012)

2012 was the year that established CNNs as the predominant algorithm for classification tasks. This was motivated by ImageNet Classification with Deep Convolutional Neural Networks [3]. In that paper, Alex Krizhevsky et al proposed "a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes". The CNN proposed achieved a top 5 test error rate of 15.3%, while the second-best entry achieved 26.2% [3]. This was the first time that a CNN obtained such outstanding results in ImageNet, a historically difficult dataset.

The architecture of the network proposed was called AlexNet [3] and consisted of convolutional, max-pooling, dropout and fully connected layers. Because training CNNs was still computationally expensive, AlexNet [3] architecture was simple (compared to current architectures). It was trained using two GTX 580 GPUs for five to six days. Krizhevsky et al introduced other interesting concepts such as data augmentation (synthetically modifying images, like image translations, horizontal reflections and patch extractions) and dropout, which reduces overfitting to the training data.

2.9.7 ZF Net (2013)

With the outstanding results obtained in 2012 by AlexNet [3], new variations of CNNs were proposed for ILSVRC 2013 [2]. The winner of the competition, ZF Net [13], achieved 11.2% error rate. It was similar to AlexNet [3], but different filter sizes were used. ZF Net [13] was trained with 1.3 million images, for twelve days on an NVIDIA GTX 580 GPU.

The main contribution was made through their paper "Visualising and Understanding Convolutional Neural Networks" [13]. The objective of this paper was to address the unclear understanding of why CNNs performed so well, and how they could be improved. In that paper, Zeiler and Fergus propose a visualisation technique, called Deconvolutional Network, that reveals how the input stimuli excite the feature map of each layer. In other words, it allows to choose an input image, feed it to the network, and understand what features is each filter of the layers learning. This unlocks even more insight on how CNNs work: images could be modified, such as rotated or occluded, and deconvolutions would help to understand visually how the activation map of each layer changed. Therefore, there is visual feedback that allows to identify if the network has correctly classify an object or is fixed at certain characteristics of it.

2.9.8 VGG Net (2014)

VGG Net was an architecture proposed in [30] which achieved 7.3% error rate in ILSVRC 2014. It was not the winner of the competition, but improved the error rate from ZF Net [13] (11.2%)

and introduced the idea that CNNs should have a deep network of layers for hierarchichal learning to work. VGG Net contains 19 layers with small filters [30], rather than the few layers of AlexNet and ZFNet (8 layers) [3, 13]. It is important to notice that more layers involves more parameters, which requires better computing power. VGG was trained on four Nvidia Titan Black GPUs for two to three weeks.

2.9.9 GoogLeNet (2014)

GoogLeNet [31] was the winner of ILSVRC 2014 [3], obtaining a top 5 error rate of 6.7% (VGG followed up with 7.3%). For this architecture, Szegedy et al proposed a deep CNN (same concept of depth as VGG [30]) with 22 layers. The key insight from this architecture is proposing a new type of layer, named Inception, in which they show that layers do not need to be stacked sequentially. Inception Layer groups different layers, such as convolutions and max pooling. Roughly speaking, this CNN contained 22 layers (or 100 layers if counting the ones contained in the Inception Layer). Training was done in "a few high-end GPUs within a week". It also dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M [3]), but the authors advice of memory efficiency and computational power required.

2.9.10 Microsoft ResNet (2015)

Residual neural networks (ResNet) was a type of network introduced by Deep Residual Learning for Image Recognition [5]. It is a CNN developed by Microsoft that won the ILSVRC 2015 with an incredible error rate of 3.6%. To realise what this means, human vision usually scores between 5-10%. Resnet followed the idea of creating a deep network, but it exceeded by far the previous number of layers used by preceding networks. It proposes a 152 layer network architecture, which set the new records for classification and detection (paired with Faster R-CNN). The network was trained on 8 GPUS for "two to three weeks" [5].

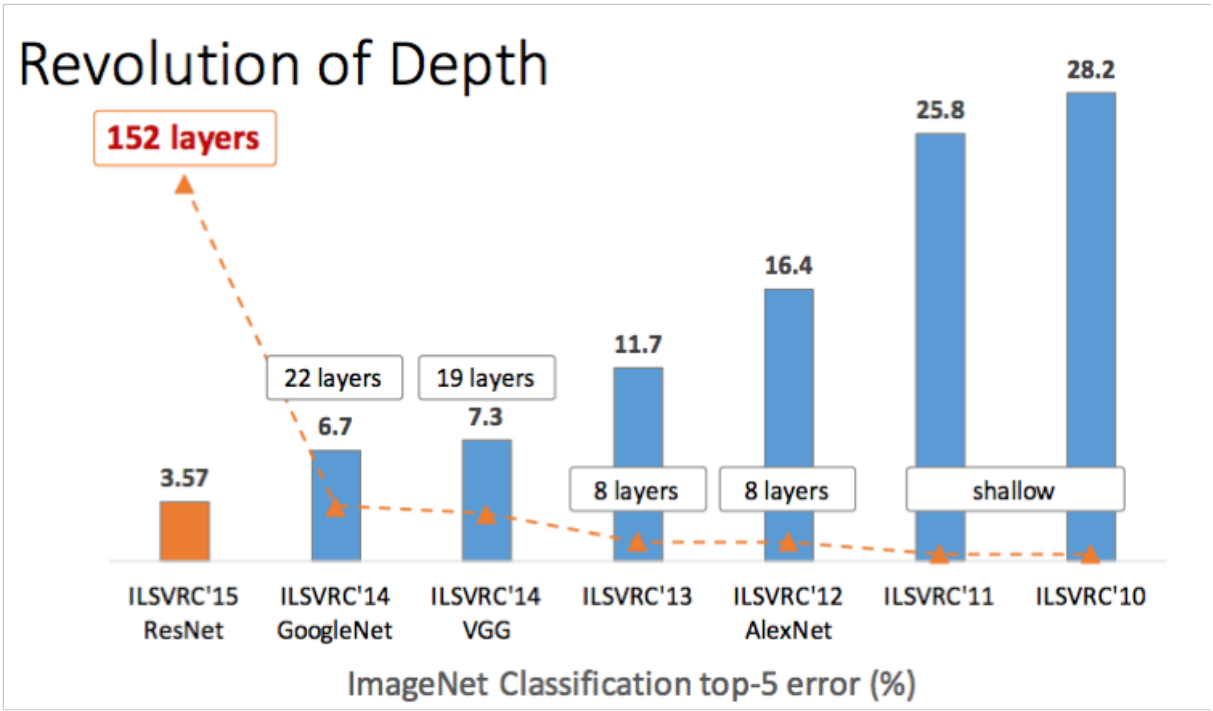


Figure 20: This figure shows the correlation between CNNs depth and the accuracy in classification tasks. It should be read starting from the right. Note the leap between ILSVRC'11 and ILSVRC'12, which was explained in (section 2.9.6: ImageNet Classification with Deep Convolutional Networks (2012)). This figure summarises several network architectures previously explained. Image extracted from [5].

2.9.11 R-CNN (2013)

The introduction of Regions with CNN (R-CNN) [6] in 2013 is one of the most important contributions to object detection. It is a simple and scalable algorithm that outperformed by more than 30% the previous best result on VOC 2012 (achieving 53.3% mAP) [6]. The objective of R-CNN is to bridge the gap between image classification and object detection. In other words, given a certain image, draw bounding boxes over all of the objects. This process can be separated in two big steps: region proposal and classification. Any implementation of region proposal can be supplied. Selective Search is a common choice, which generates 2000 different regions that have a high probability to contain an object. The proposals are then resized and fed into a trained CNN (AlexNet [3] or ResNet [5] for example). A feature vector is extracted for each region and is then used as input of different support vector machines(SVM) that are trained for each class and output a classification. There is a last pass through a bounding box regressor which returns the most accurate bounding box.

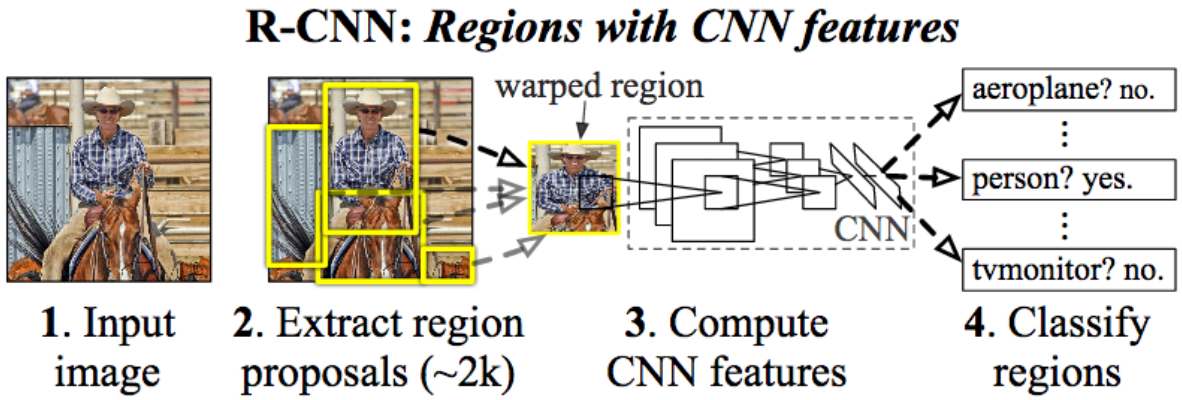


Figure 21: The system extracts region proposals from the image, compute features for each proposal using a trained CNN, and classifies each region using class-specific linear SVMs, achieving 53.7% on PASCAL VOC 2010. Image extracted from [32].

2.9.12 Faster R-CNN (2015)

Several variations of R-CNN appeared on 2015, trying to increase its performance. One of the most known variations of R-CNN is Faster R-CNN [8] which works similar to R-CNN but produce the region proposals on the last convolutional feature map (R-CNN did it on the input image). As the input image already went through several convolutions layers, the size of the feature map is lower, so requiring less performance for the region proposal.

2.9.13 YOLO

You Only Look Once (YOLO, 2015) [9] is part a type of region based CNNs, drastically more efficient than R-CNN. Recall that R-CNN first generates potential bounding boxes in an image and then classify them, with post processing operations to refine the bounding box and eliminate duplicate detections. On the other hand, YOLO proposes a network that computes directly from pixels to bounding box coordinates and class probability. A single CNN predicts simultaneously bounding boxes and class probabilities for those boxes. This leads to a faster algorithm (45 FPS with no batch processing on a Titan X GPU), and also a fast (and less accurate) version that runs at "more than 150 FPS" [9]. This makes it possible to stream a video in real time with less than 25 ms of latency.

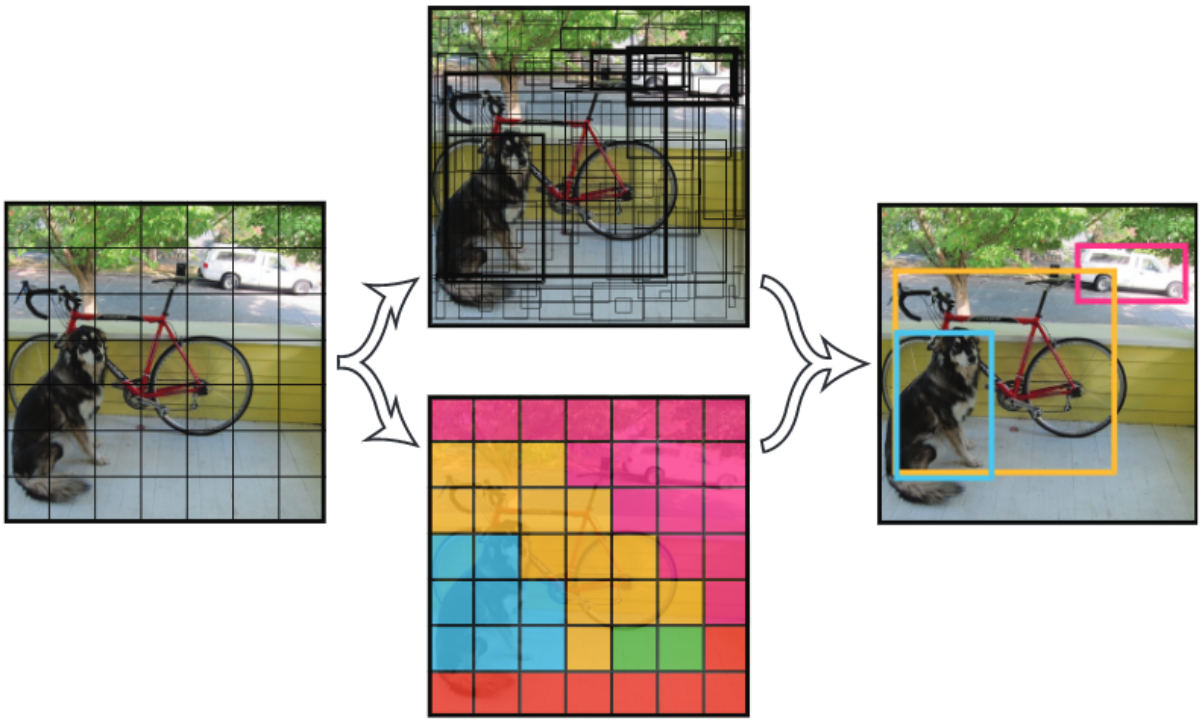


Figure 22: YOLO divides the image into an even grid and simultaneously predicts bounding boxes, confidence in those boxes, and class probabilities. Image extracted from [9].

2.9.14 SSD

Single Shot MultiBox Detector(SSD, 2016) [10] is another object detector that belongs to the class of region based CNNs. It is similar to YOLO in the sense that it avoids hypothesizing bounding boxes, resampling pixels or features for each box, and applying a high-quality classifier [10]. The difference with YOLO is that SSD considers different aspect ratios of the input image. The filters used are different for each aspect ratio and combined at the last stages. The final prediction is the union of all these predictions.

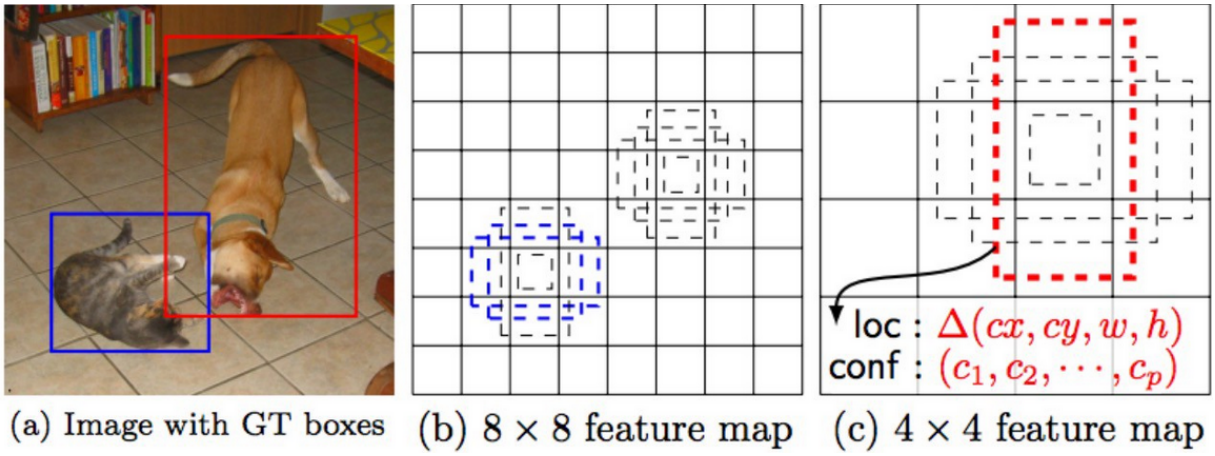


Figure 23: The left image is used for training, with its correspondant ground truth. Two different feature maps represents the aspect ratio (size of the boxes). Note how the middle one (8x8) with smaller boxes detects the cat, while the last one (4x4) detects the dog. Image extracted from [10].

SSD works by having a base classification network (as the ones mentioned before, like RESNET [5] or VGG [30]). It then adds convolutional layers to the network to the end of the base network. These layers decrease in size progressively and allow predictions of detections at multiple scales [10].

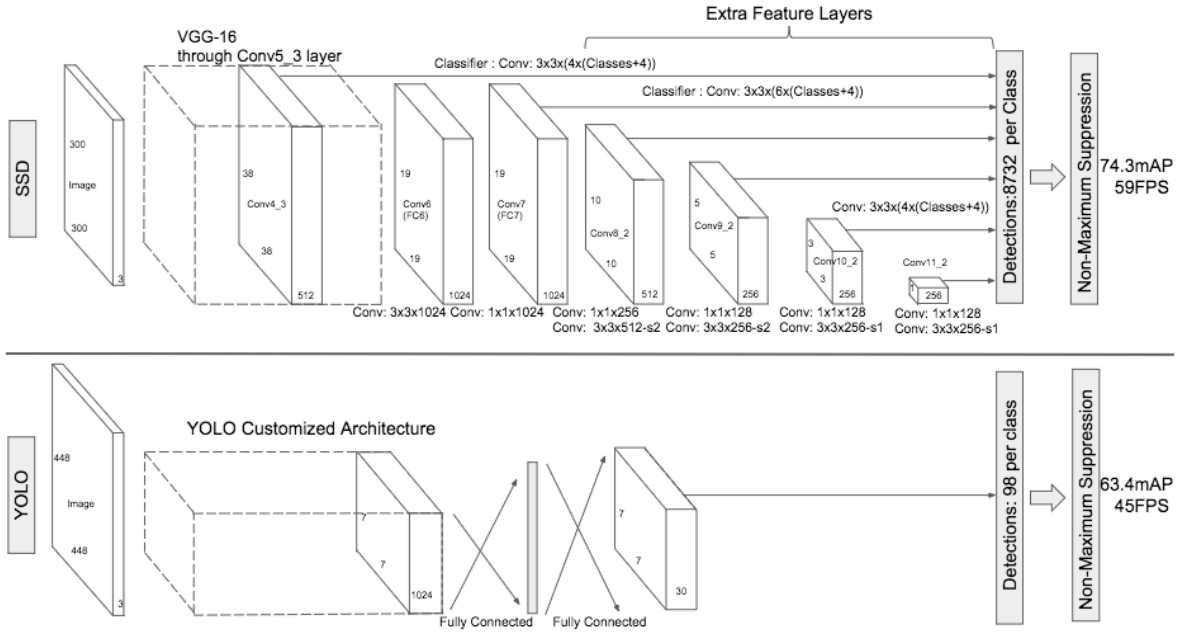


Figure 24: Comparison of the architectures of YOLO [9] and SSD [10]. Note that SSD considers different aspect ratios (different convolutional filters size such as conv 1x1x1024 and conv 1x1x256) and then combines the predictions after the extra feature layers pass. Image extracted from [10].

2.9.15 CNNs consolidation

Computational resources and availability of annotated data has increased throughout the years. In 2005, several investigations established the value of using GPU in machine learning. By 2011, [3] proposed high performance CNNs for image classification, based on GPU computing, which achieved the best results in MNIST handwritten digit recognition conquest. New investigations spawned from that moment and CNNs got popular in the scientific world. Performance power increased an order of magnitude (from 10a3 to 10a4 GFLOP/second) since 2008 to 2016 (with the introduction NVIDIA Titan GPU), and parallel computation provided even better results. Availability of annotated data has also increased (not at the same rate as computational power), driven by the appearence of datasets such as PASCAL VOC [4] and ILSVRC [2]. In object detection, images need to contain annotations with the class that a particular object belongs to and the region of the image in which it is located (4 points that describe the bounding box).
In the past, object detection was implemented using engineered features, for example SIFT [28] and SURF [33]. While they are still in use, feature-extraction algorithms don't scale with the gain on availability of annotated data. Furthermore, they lack the ability to learn from data itself (features need to be previously defined). The outcome is that statistical models outperform feature-extraction in all modern competitions since 2012 (as shown in section 2.9: Historical Progress).
There are plenty of statistical approaches to object detection, but there are some set of features that make CNNs uniquely fit for that task. The concept of CNNs is not a new idea, as they have been around since the work of Hubel and Wesel. The explanation of why they weren't used before as they are now rely in two facts:

- Performance: CNNs require to compute a high number of (simple) mathematical operations. It was not until the introduction of GPUs that it was feasible to train CNNs in a reasonable amount of time, specially because of the fact (shown by [27] and [13]) that CNNs require several layers to be able to extract features hierarchically from data.
- Annotated Data: CNNs need considerable amount of information and this information need to be annotated. This means that not only training images are needed, but also annotations about the image.

which, as explained before, both increased throughout the years. The main features that distinguish CNNs from other statistical models is that:

- They exploit feature locality: This means that pixels that are near each other tend to belong to the same feature (for example, edges in low level features).
- Work at different granularities: First layers are able to detect low level features, while more in depth layers detect more complex ones. This allows to hierarchically extract features, which makes possible to apply transfer learning (use the already trained layers for other object detection task).

- Best results in main image recognition competitions: Mentioned in the section 2.9: Historical Progress, they obtained the best results in all known datasets, such as PASCAL VOC [4] and ILSVRC [2].
- Slow to train, but fast when deployed: This allows to run detections faster compared to other algorithms, providing better performance.

The last and most important point is that CNN get better with more annotated data, which is exactly the objective of this thesis. A mechanism of reliable generating arbitrarily sized datasets that resemble real world images allows to rely only on performance (needed for generating the datasets).

3 System Overview

The system developed provides the ability to generate synthetic images, train SSD object detector with them and run detections on images or videos with the trained network. For ease of simplicity, the system was designed having in mind independent pipelines. These pipelines represents an abstraction over the functionalities required for an object detector. Before starting the implementation, a deep analysis was made of the requirements of the system. As for every step of detection there is an input, a processing of the input and an output, independent pipelines appeared to be an effective architecture. For communicating the pipelines, two models were designed: dataset and experiment. Due to the number of parameters involved, different pipelines also receive these parameters via configuration files. This chapter will describe in detail how the system works, detailing the task that each pipeline abstracts.

3.1 Algorithm chosen

The statistical method chosen was CNNs, based on the advantages they provid (described in chapter 2: Basics, section 2.9.15: CNNs consolidation). The main alternatives of CNNs analyzed were YOLO [9] and SSD [10], because of their results in PASCAL VOC [4] datasets. Though the accuracy in VOC 2007 is similar for both methods (table 1), a qualitative analysis for car detection was done with both of them. SSD showed better results regarding the stability of the detected bounding box. Another important point is that SSD is implemented on top of a well known framework for deep learning, Caffe [10]. It already implements tools regarding performance, debugging and network visualizations. The downside of this choice, however, is that YOLO v2 shows greater performance [9]. As accuracy and stability of bounding box is critical for the application of this paper, SSD was chosen. It is important to remark that the main focus of this paper is on synthetic dataset generation, which means that the techniques described can be applied to both YOLO and SSD, and other types of object detection algorithms.

Algorithm	mAP	FPS (Titan X)
YOLO	63.4	45
SSD-300	77.2	46
YOLO v2 544 x 544	78.6	40
SSD-512	79.8	19

Table 1: Comparison of YOLO [9] vs SSD [10]. In every modern object detection algorithm, not only the accuracy is compared, but also the performance (FPS). In this case, due to hardware limitations (GeForce GTX 960) and to have a performant method, SSD-300 (trained with 300 x 300 input image size) is chosen as the base architecture. However, SSD-512 (trained with 512 x 512 input image size) is also explored.

3.2 Models

Conceptually, the project expose the following models:

3.2.1 Dataset

A dataset is an abstraction that contains all the necessary information to interact with the data. The pipelines that work with datasets exposes an interface that receives a dataset and perform their task. The dataset model provide plenty of customisation options, such as:

- Number of images
- Type of dataset (train or test)
- Path to dataset
- Image width and height
- Path to the scene that will be load in Blender.
- Path to the scene configuration
- Path to the annotations
- Class id (represents the class we want to detect, for example "apple"). The requirements only demand to detect a single class.

An additional model CombinedDataset was created, which exposes similar properties of Dataset but with the difference that it contains multiple datasets. Because Dataset and CombinedDataset share a similar interface, pipelines don't need to distinguish between them.

3.2.2 Experiment

Instead of running experiments and manually recording results, an Experiment model was created. This model contains all the information related with preparing a new experiment, running an old experiment (with the same parameters) and storing the visualisations. The experiments model provide plenty of customisation options, such as:

- Number of test images to use.
- A train dataset, that can either be Dataset or CombinedDataset.
- Experiment name and path.
- Path to snapshots directory, where the trained weights will be stored.
- Path to the different networks (training, testing and deploy).
- Path to the configurations of the networks (for training, testing and deploying).
- Path to qualitative detections of the default images and videos.
- Visualizations, with the filters and plots.
- Detections of the test dataset with the trained network.

Both Experiment and Dataset are persisted and loaded, to easily use them for other tasks. For example, an experiment can be used to test the network in a new dataset.

3.3 Pipelines

Several pipelines were designed, along with services and utilities that are shared by the different pipelines. The pipelines exposed are:

- Dataset Generation: Generate an arbitrarily sized synthetic dataset, using a 3D engine. (In this case, Blender [25] is used).
- Train: Trains the detector with the generated datasets.
- Detection: Run detections of chosen images/videos.
- Visualization: Generate plots and visualize what the filters are learning.
- Viewpoint estimation: Generate a synthetic image that resembles the detected object, by using viewpoint estimation [14] tool.

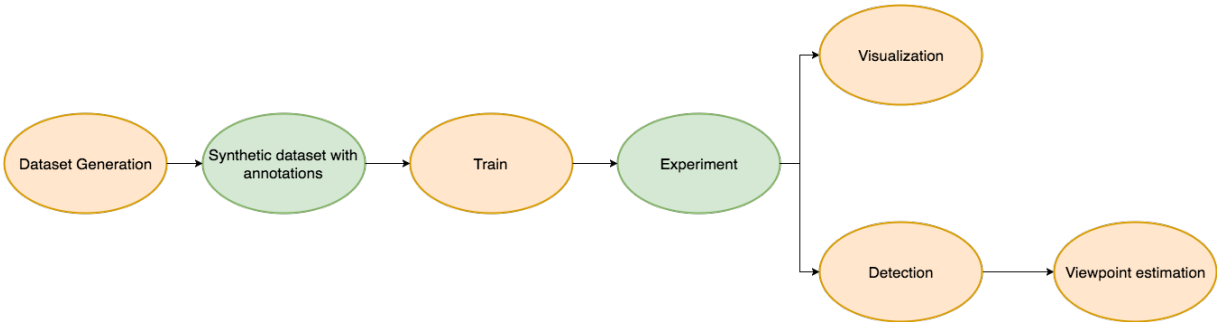


Figure 25: High level overview of the main pipelines. The train pipeline is the most important one, which receives a synthetic dataset and train the CNN. After this training happens, the resulting network can be used by other pipelines. Visualization will generate useful charts and graphs about the learning to understand what is happening with the network. Detection allows to use the trained network to run detections of images and videos. Via the Detection module, the network can be deployed and be part of bigger systems. Last of all, the detection can be "plugged" into viewpoint estimation [14].

Although those are the main pipelines, several other components were developed for abstracting shared logic. They include:

- Annotation manager: Used to write and read annotations with specific formats.
- Persistance manager: Contains the logic to serialise and deserialize the different models (Dataset, CombinedDataset and Experiment).
- Image manager: Utility functions for images, such as cropping, drawing rectangles on images or loading images into memory and saving a into file.
- Video manager: Functions used for handling video, such as obtaining frame rate or converting video to frames and vice-versa.
- Train monitor: Displays in different terminals CPU usage, GPU usage, and accuracy over iterations.
- Logging manager: Provides a logging facility used by all components.
- File manager: An abstraction over python file operations, such as ensuring file exists (or not) before opening or listing all files in directory with certain extensions.
- LMDB manager: Allows to generate a LMDB from a Dataset and to read images from a LMDB (useful to ensure that the LMDB dataset is created correctly).

Figure 26: Shows the shared components, used by different pipelines.

3.4 Dataset generation pipeline

This pipeline gathers the background images (or textures), a 3D model and the scene parameters, and generates a synthetic dataset. The synthetic generation module used was first proposed in [14], but containing it in that project made it complex to reuse it. Therefore, the synthetic generation tool was extracted to a component, independent of viewpoint estimation [14] and object detection. In order to continue the research done by [14], both [14] and the object detection module access the synthetic dataset generation tool by the same interface. The synthetic generation tool requires a 3D scene in which the 3D model is placed on, and the supplied scene parameters (that will modify the characteristics of the scene for obtaining different images). With these input parameters, the dataset generation tool is able to create a synthetic dataset (with annotations) of arbitrarily chosen number of images.

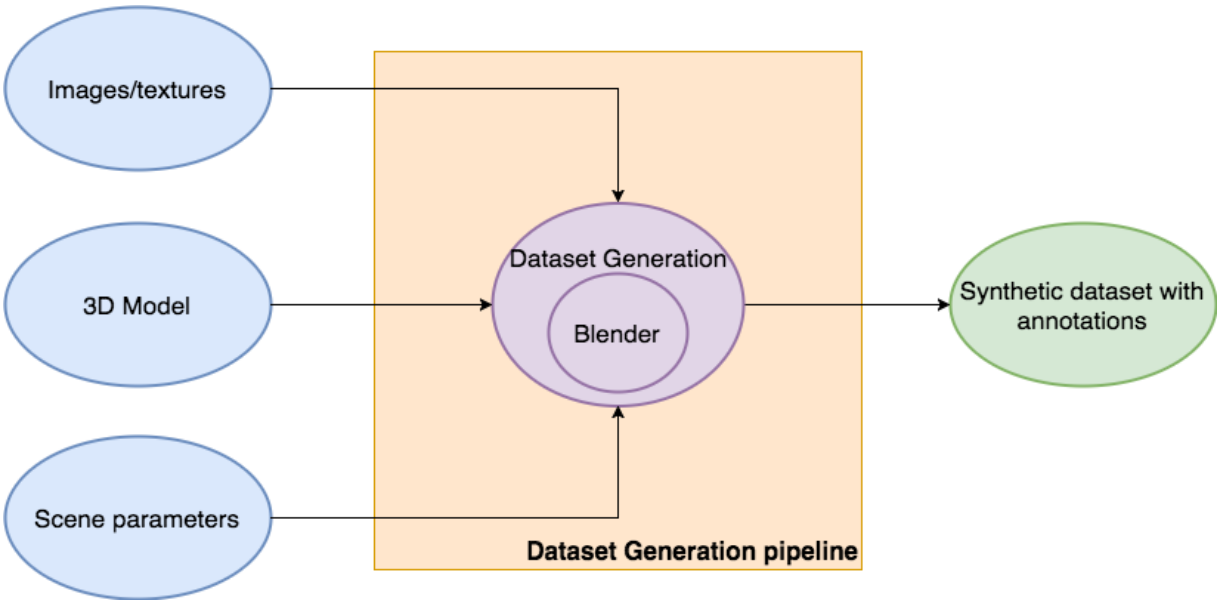
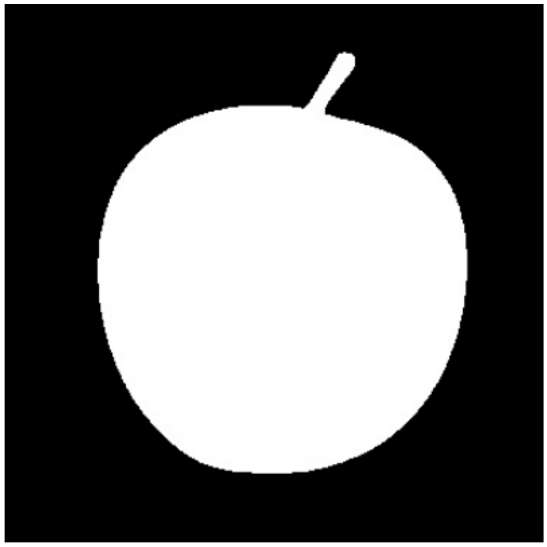


Figure 27: Overview of the dataset generation pipeline. In blue, the input of the pipeline, green refers to internal to this system and violet are external tools. The output is colored green. Notice the arrows, which indicate the information flow.

3.4.1 Annotations

The dataset generation tool supports to annotate the class that the 3D model belongs to, the viewpoint (three angles describing the object's pose) and the bounding box(four coordinates). In this work, the focus is on the latter, as it is required for object detection. To be able to annotate the bounding boxes, a new scene with just the 3D model with white color and a black background is designed. A simple function to find the contours provided by opencv (collection of computer vision algorithms) is used to obtain the four points that describe the bounding box of the object. This automates the annotation of the bounding box, as no human interaction is needed, but as a tradeoff it doubles the amount of images generated (one for the synthetic image generated, one for the black image with white 3D model).

(a) Synthetic generated apple.



(b) Black and white image of the apple used to extract the bounding box via opencv find contours.

Figure 28

3.4.2 Scene parameters

The first step for designing the scene is to define the global coordinate system (xyz orthogonal axes), and the origin point. All the objects contained in the scene will be loaded according to this reference. Note that the objects (3D models and lights, but more could be added) are loaded before the dataset generation starts, and have to be added manually. A camera is also needed: it will represent the observer of the scene. The camera is one of the most important component for generating the synthetic images: it will be moved randomly throughout the scene (always with the 3D objcet in sight) and generate the synthetic images. For a more in depth explanation about the scene description, refer to [14]. Once the scene is already designed, containing the 3D model(s), lights and camera, a blender API bpy is used to access the textures

of those objects at runtime (while generating the images). A yaml scene configuration file has to be supplied, with the chosen scene characteristics. These characteristics include minimums and maximums for the different aspects that constitute each object. The scene configuration parameters used in this thesis are contained in the appendix. For example, for the camera it is possible to set:

- Distance: range of distances between the camera and the 3D model.
- Azimuth: range of camera azimuth angle.
- Elevation: range of camera elevation.
- InPlane: range of in-plane rotation.

The scene will then take random values for each one of the properties.

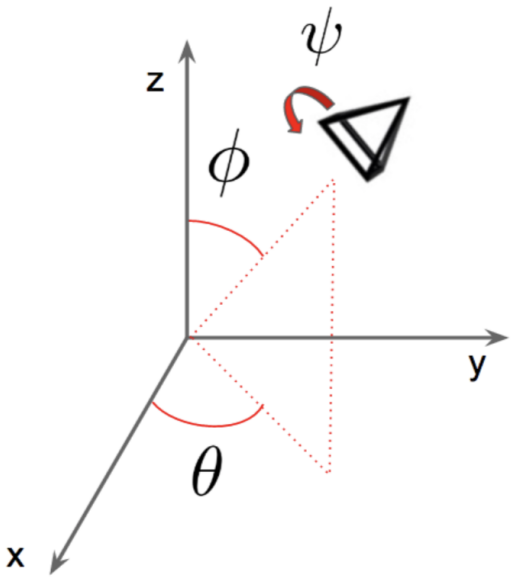


Figure 29: Visualization of the angles of the camera. Image extracted from [14].

3.5 Train pipeline

Training constitutes one the most important pipelines of the whole system. Given the train dataset, test datasets and train parameters, it trains the object detector and store the weights in the experiment. This module is an abstraction over SSD's fork of Caffe (framework for computer vision tasks) [10], and automates all the complexity required for traning using caffe. After this phase, the experiment contains the trained weights which allows it to be used by other pipelines that need the trained network (such as Detection or Visualization).

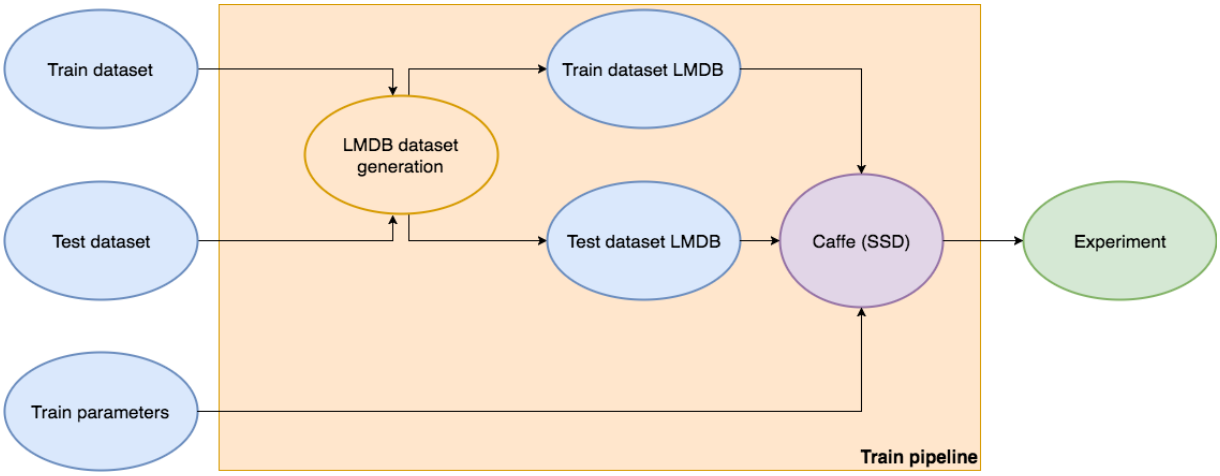


Figure 30: Overview of the train pipeline. Blue distinguish input data, orange means that it was developed in this work and green is used for the output (trained experiment) of the pipeline. Arrows indicate the information flow (from left to right), and external tools (SSD fork of Caffe [10]) with a violet background

3.5.1 LMDB datasets

The input format for Caffe tasks is called Lightning Memory-Mapped Database Manager (LMDB) [34]. It is a software library that provides a high-performance embedded transactional database in the form of a key-value store [34]. This database is exposed entirely in memory, therefore highly efficient for retrieving the data. It is extremely high performant and memory-efficient [34] and fully transactional with full ACID semantics. Caffe uses LMDB for every type of input dataset.

3.6 Visualization pipeline

Once an experiment is already trained (in other words, contains the trained weights), visualizations tools provided by caffe are used. It is straightforward to add a new visualization tool to this pipeline, as the visualizations are stored in a folder inside the experiment.

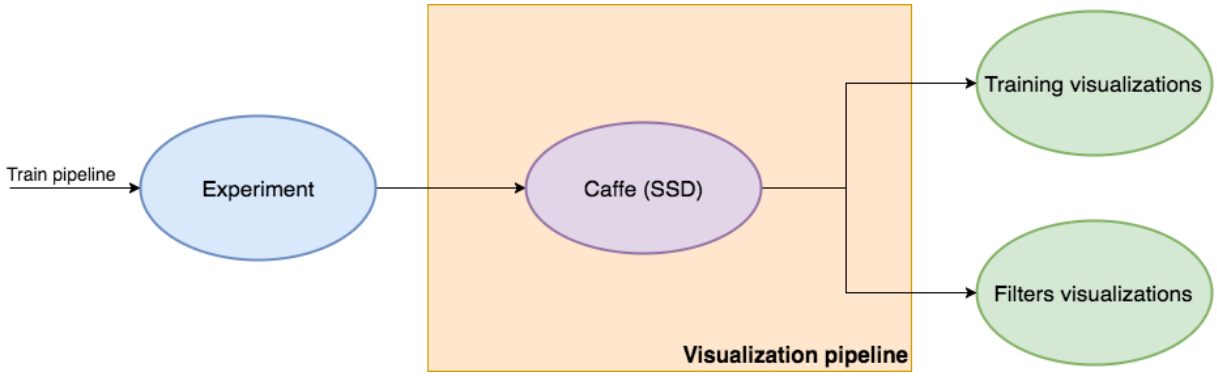


Figure 31: Overview of the visualization pipeline. Blue distinguish input data, orange means that it was developed in this work and green is used for the output of the pipeline. Arrows indicate the information flow (from left to right), and external tools (SSD fork of Caffe [10]) with a violet background.

3.6.1 Learning curve

Visualizing the learning curve is provided by Caffe. Different plots are available, but the most important is accuracy over iterations.

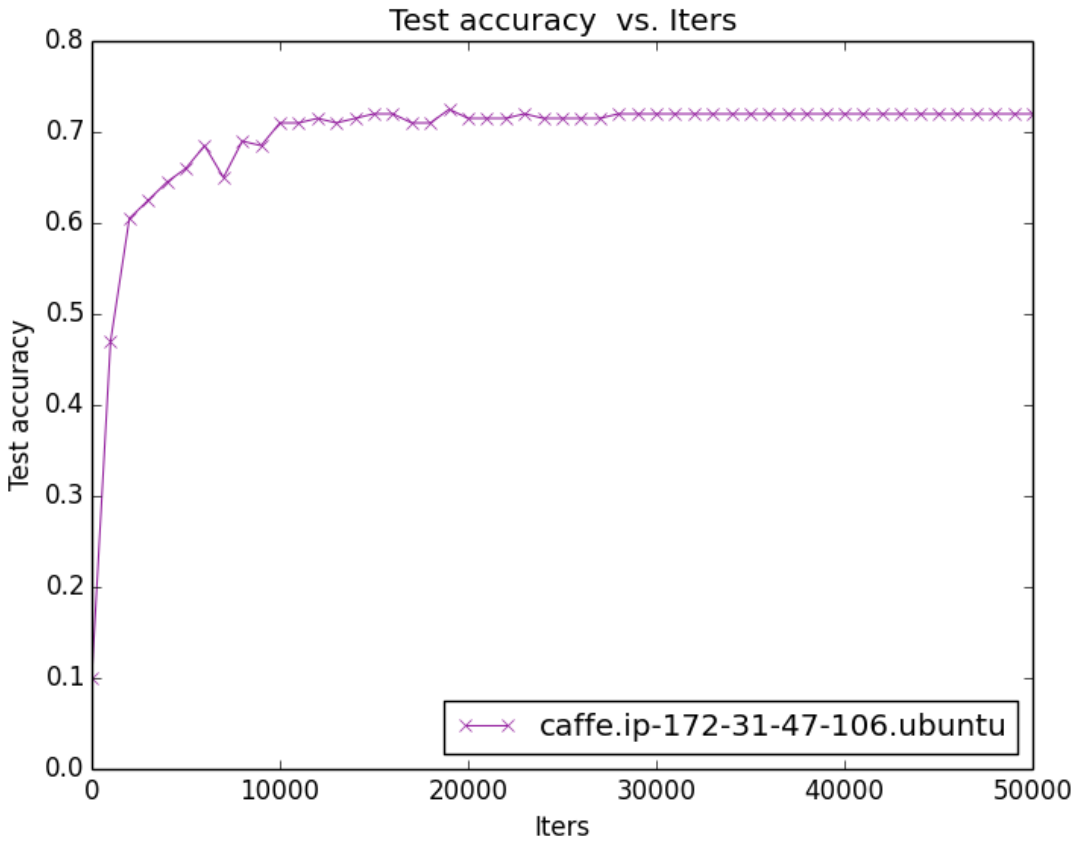


Figure 32: Sample of a plot of accuracy vs iterations generated by caffe.

3.6.2 Layers visualization

Albeit the improving results that CNNs have faced throughout the years, it is still a field that is guided by empirical methods rather than theory. This have lead to intuitive results that help to explain how CNNs learn, but there is still a considerable gap between theory and practice. One way of attacking this gap is to be able to visualize what a CNN is learning throughout the activation map of its layers (technique described in [13, 35]).

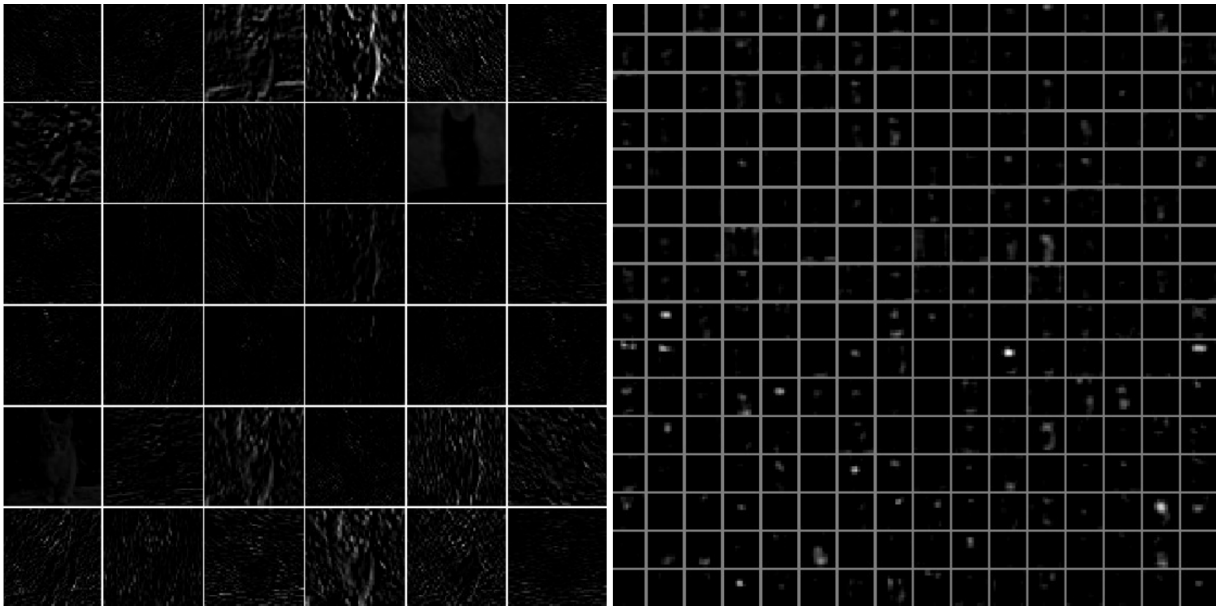


Figure 33: Typical-looking activations on the first CONV layer (left), and the 5th CONV layer (right) of a trained AlexNet looking at a picture of a cat. Taken from [35]

3.7 Detection pipeline

Once the model is already trained (and the weights contained in the experiment), it can be used to run detections in images or videos. That is the task of the Detection module. It coordinates the steps from gathering the images, loading the network in memory and running the detections. This module accept both images or videos, and has two possible outputs. The first option is to lay green rectangles on the input showing what did the model learned. This is the most used feature (both for qualitative analyzing what the network is learning and to visually show the network in action). The other option is the ability to crop the images for each detection it finds. This is useful for example for interacting with other components, such as viewpoint estimation [14] that need as input the region that contains the detection (and not the whole image).

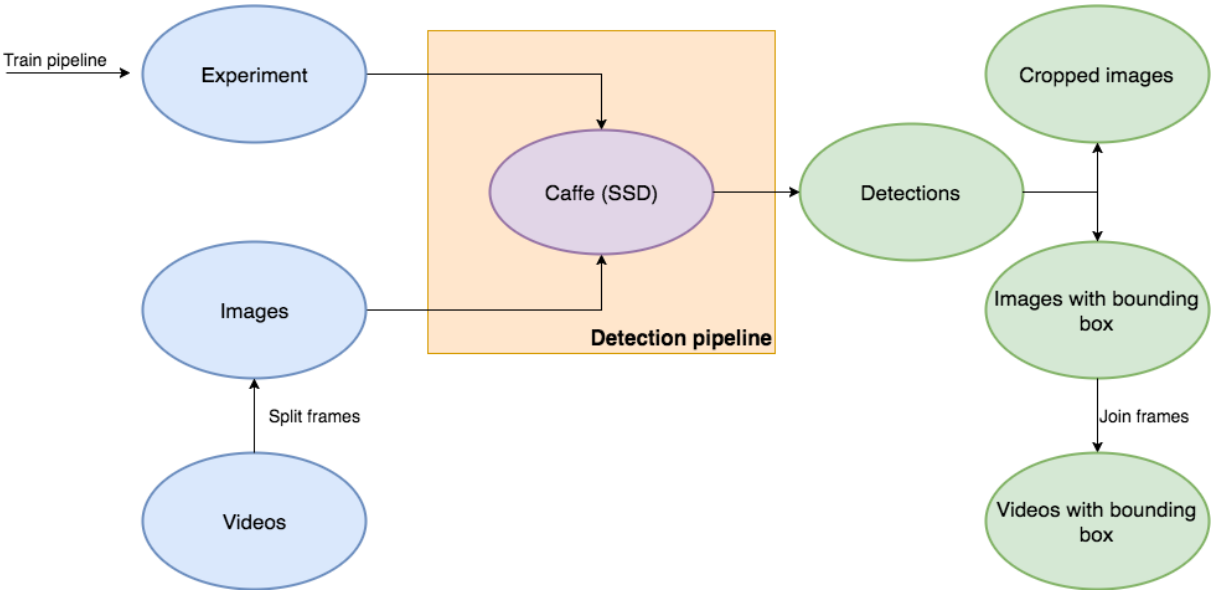


Figure 34: Overview of the detection pipeline. Blue distinguish input data, orange means that it was developed in this work and green is used for the output of the pipeline. Arrows indicate the information flow (from left to right), and external tools (SSD fork of Caffe [10]) with a violet background.

3.8 Viewpoint estimation pipeline

With the cropped image output from the detection pipeline, and supplying a chosen background with the scene parameters, the external viewpoint estimation [14] tool is used to extract the viewpoint from the cropped image. The synthetic generation tool was modified so that, given the estimation of the pose of the real object, recreates the 3D model accordingly. The output is an image (similar to the training dataset) that mimic the object's pose. An example will be shown in chapter 4: Experiments, section 4.7: Viewpoint estimation pipeline.

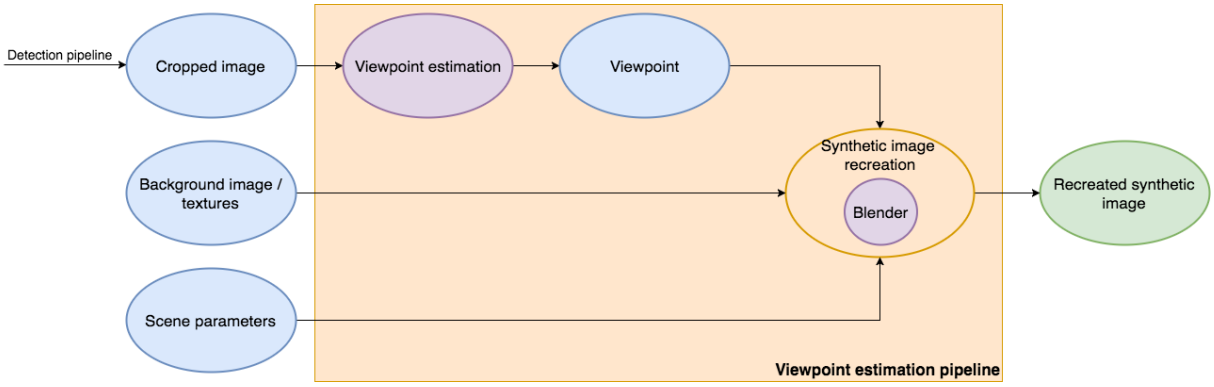


Figure 35: Overview of the viewpoint estimation pipeline. Blue distinguish input data, orange means that it was developed in this work and green is used for the output of the pipeline. Arrows indicate the information flow (from left to right), and every external tool (Viewpoint estimation [14] and Blender [25]) with a violet background.

4 Experiments

This chapter describes the experiments designed to test the accuracy of the object detection module trained on synthetically generated data. It shows a concrete application of the system (chapter 3: System Overview) to detect a 3D Volkswagen Golf car. It outlines the characteristics of the input for each of the pipelines, the type of visualizations obtained, the interaction with the dataset generation tool (abstraction over Blender [25]) and viewpoint estimation [14].

4.1 Type of experiments

The experiments focus in the following variables:

- Dataset size
- Type of background
- Images input size
- Adding an external 3D model (Peugeot)

while all other characteristics (such as network architecture or test dataset) remain constant. Therefore, the accuracy comparison of the detectors trained using different synthetic datasets is a reliable and reproducible.

4.2 Dataset generation

4.2.1 3D model

The application of this work was related to car detection, specifically Volkswagen Golf. Therefore, a 3D model of Golf was used for generating the images of the training dataset.

4.2.2 Ground and Sky

The image generation tool was already able to generate images of a scene with a car, a plane and a sky. Multiple textures are supplied for the ground and the sky, therefore generating different images. [14] mentions that there should be a considerable amount of textures, and different between training and testing. Combined with varying light conditions, this would help the model to avoid overfitting on already known textures.

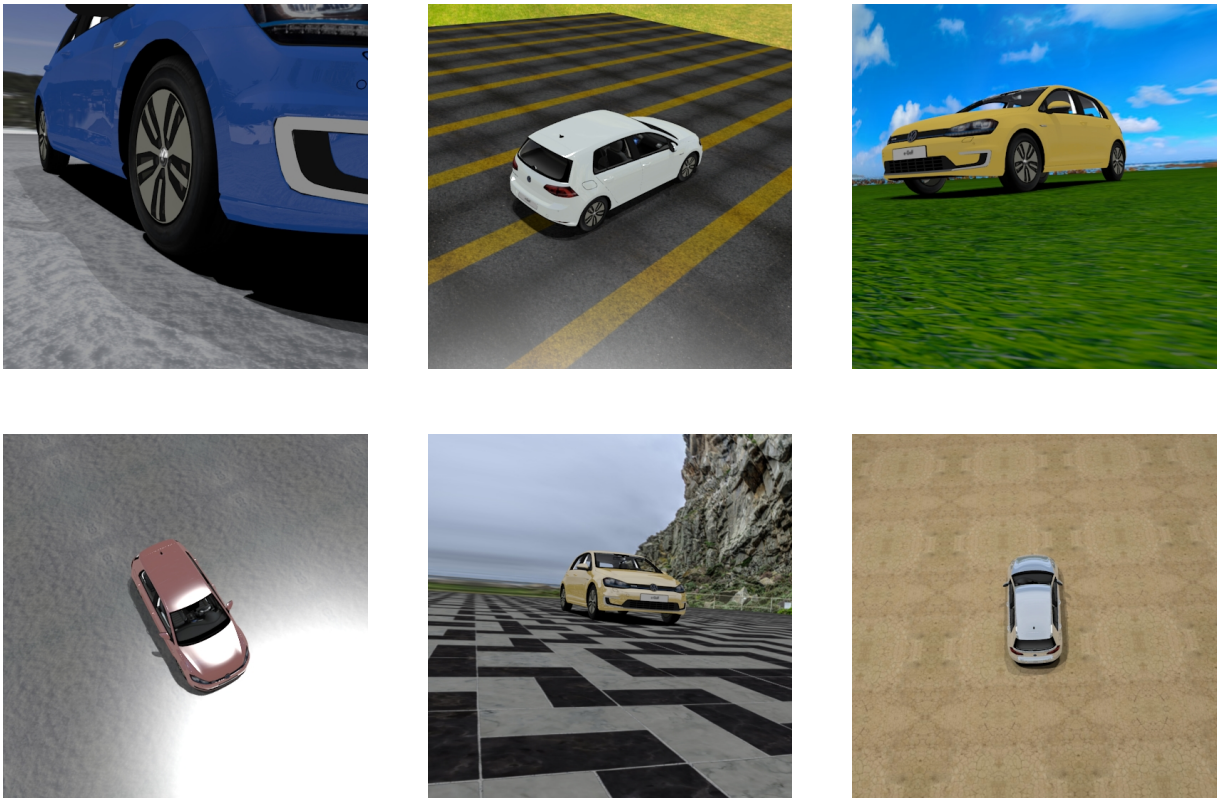


Figure 37: Sample of datasets generated using uniform textures in the ground and the sky, and a Golf 3D model.

4.2.3 Whole background

Another type of synthetic images generation was added to the generation module. Based on [22], which combines indoor backgrounds with objects by superposing the object in the image. By advanced means of support detection, [22] was able to predict where the object could be in the image according to the support areas of it. Therefore, it generated synthetic images which resembled real world images (refer to chapter 2: Basics, section 2.8.1: Generating images via a game engine). The same concept was implemented for the synthetic generation module, by adding whole background images instead of ground a sky. They consist of a scene with a single background image and a 3D model. The reasoning behind introducing this kind of images is that region-based CNNs sample regions from the image, taking negative examples from the background (which aren't labeled as part of the 3D model). Therefore, there is more real information by using real images as background, taking into account that the relative position of the car in the image isn't relevant for the algorithm.

4.2.4 Obtaining background images

A script was developed in Python which searches google images for certain categories and download batchs of 100 images for each category. In this work, 10 different categories were

used (1000 different backgrounds). Because the chosen 3D model is a Volkswagen Golf, the categories are designed to be related to the environment in which that model could be contained (such as "city" and "buildings"). Refer to the Apendix for the complete list of background categories.

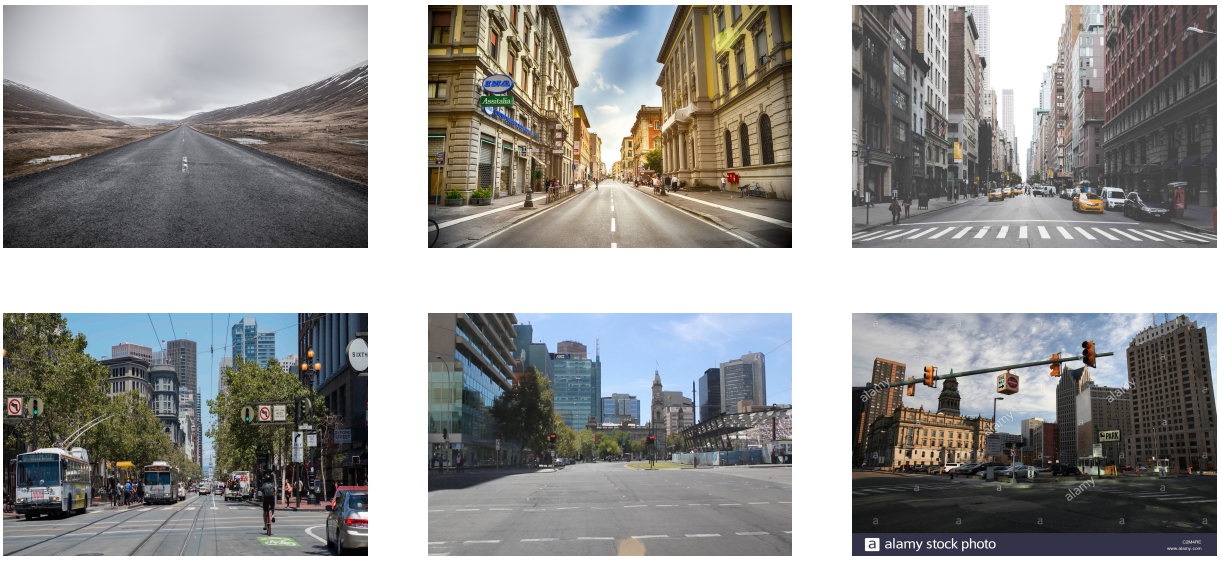


Figure 39: Samples of background images downloaded from google

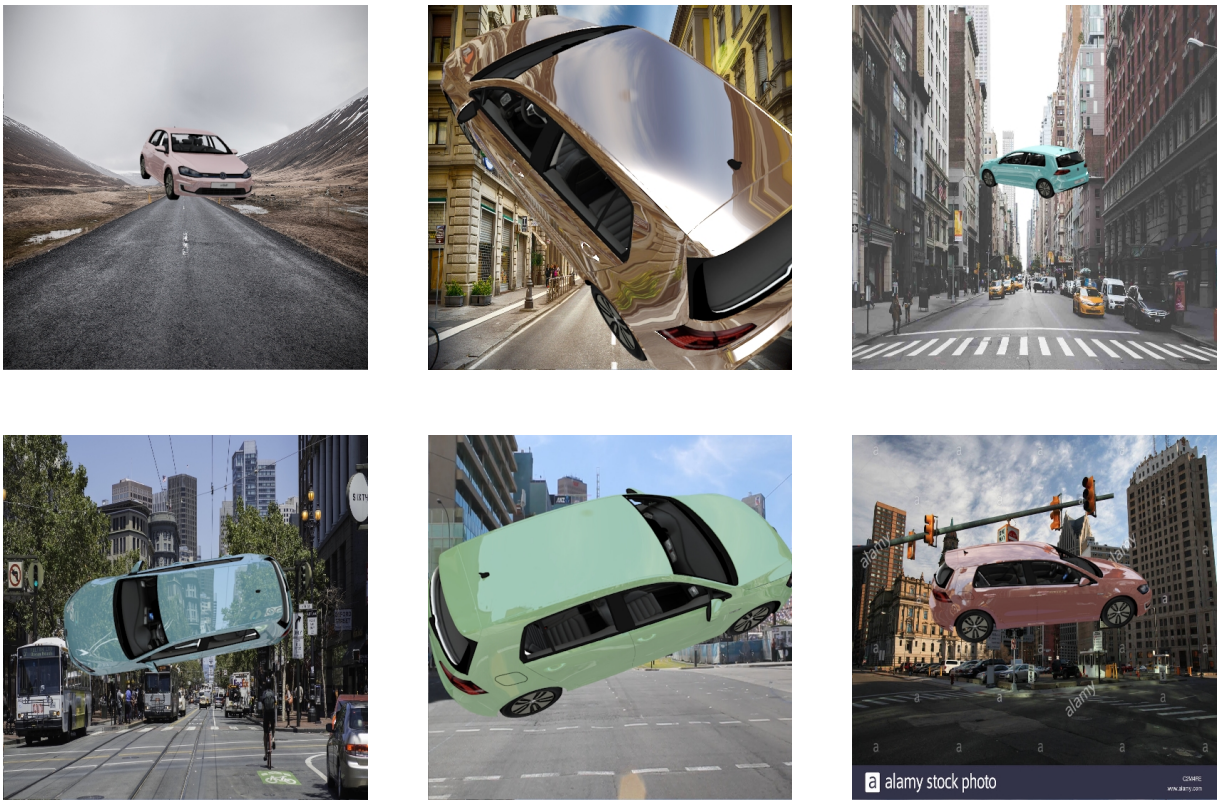


Figure 41: Sample of whole background datasets: background images combined with Golf 3D model.

Note that in most of the cases, whole background images do not resemble real images. However, the way region based CNNs (chapter 2: Basics, section 2.5: Convolutional Neural Networks (CNNs)) works makes those images suitable for training tasks. Recall that region based CNNs sample regions where an object could be (for example, 2000 region-proposal used in R-CNN [6]), so the image as a whole is not as important as the relative parts contained in it.

4.2.5 Combined

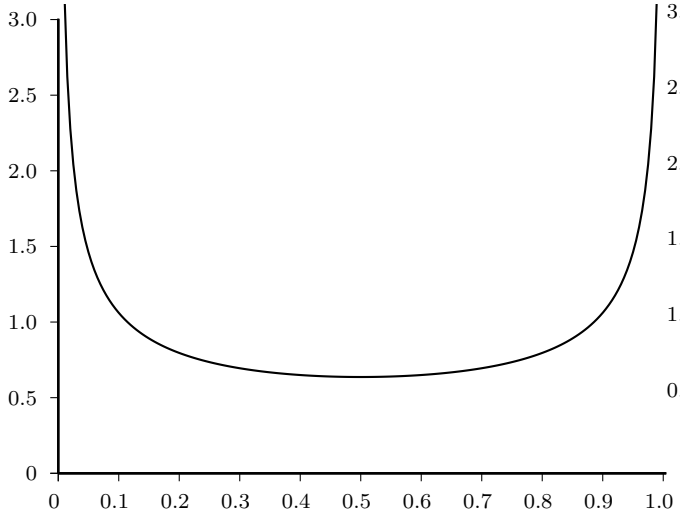
Ground and sky and whole background datasets are combined into a single dataset, to evaluate how the CNN reacts to images generated in different ways.

4.2.6 Scene parameters

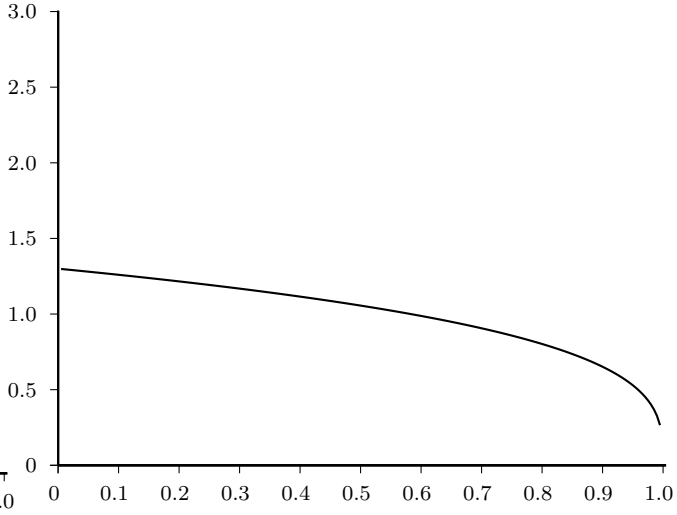
It is possible to change characteristics of the scene such as light conditions, car color, and influence of shadows over each object of the scene. These options are supplied via a configuration file, and are used by Blender [25] to generate the synthetic images. Refer to chapter 3: System Overview, section 3.4.2: Scene parameters for more information.

Random distributions

In the configuration file, a minimum and a maximum value are supplied for each property that is changed. For every image of the synthetic dataset, a random value is generated between those values (for example, light intensity or car color). In viewpoint estimation [14], the distribution used to generate random values is always uniform (same probability for every number inside the range). [14] recommends to try different distributions. In object detection, it is required that the detector can perform close up detections, but at the same time these are the most difficult ones (because the overall shape of the object isn't present). Having into account the suggestion of [14] and considering the requirements of object detection, the random distribution was changed for certain properties of the camera. For the camera distance between the camera and the 3D model, a beta distribution is used with $\beta = 0.5$ and $\alpha = 0.5$. This results in a higher probability of lower and higher values, in others words, near the 3D model or far away from it. This way the detector can gather both characteristics from close ups and the overall shape of the object. beta distribution was also used for the elevation angle (described in chapter 3: System Overview, section 3.4.2: Scene parameters). This angle determines how high the camera will be. Images of Golf are usually taken from a human height, meaning that it is not so important to have a high elevation angle. $\alpha = 1$ and $\beta = 1.3$ are chosen, which will produce lower values, which is equivalent to lower elevation of the camera.



(a) Beta distribution using $\beta = 0.5$ and $\alpha = 0.5$ used for the distance between the camera and the 3D model.



(b) Beta distribution using $\beta = 1.3$ and $\alpha = 1$ used for the elevation of the camera.

Figure 42: Different distributions were used for the camera properties.

4.2.7 Input resolution

The resolution of input images has an important influence on how the network learns. SSD supports two possible input resolutions: 300x300 and 512x512. It is important to note that having a bigger resolution impacts the experiments in different ways:

1. Batch size has to be reduced, which can negatively affect the learning (less images per batch means less images that affect the weights update, which may lead to a local minimum).
2. As images are bigger, there is more information gained by the network, which leads (usually) to better results.
3. Makes small details easier to detect.
4. Better GPU is required which leads to worse performance.
5. More time needed to generate images:
   - 300x300: 2.5 days for 200k synthetic dataset using GeForce GTX 960.
   - 512x512: 4 days for 200k synthetic dataset using GeForce GTX 960.

(a) 300 x 300 image

(b) 512 x 512 image

Figure 43: Visual example of the different image resolution used. Note that both are downscaled proportionally to fit the page, but the aspect ratio is maintained.

4.2.8 Adding Peugeot 406

It is interesting to understand what happens when another model is added. That's why an open source Peugeot 406 3D model was also used to generate synthetic images. Those images are combined with the already created synthetic datasets of Golf model. The network is then trained to either detect Golf and Peugeot as a single class (consequently, distinguishing between that combined class and background) or it is trained to detect 3 classes: background, Golf and Peugeot.

Figure 44: Images generated using uniform textures in the ground and the sky, and a Peugeot 3D model, as used in [14].

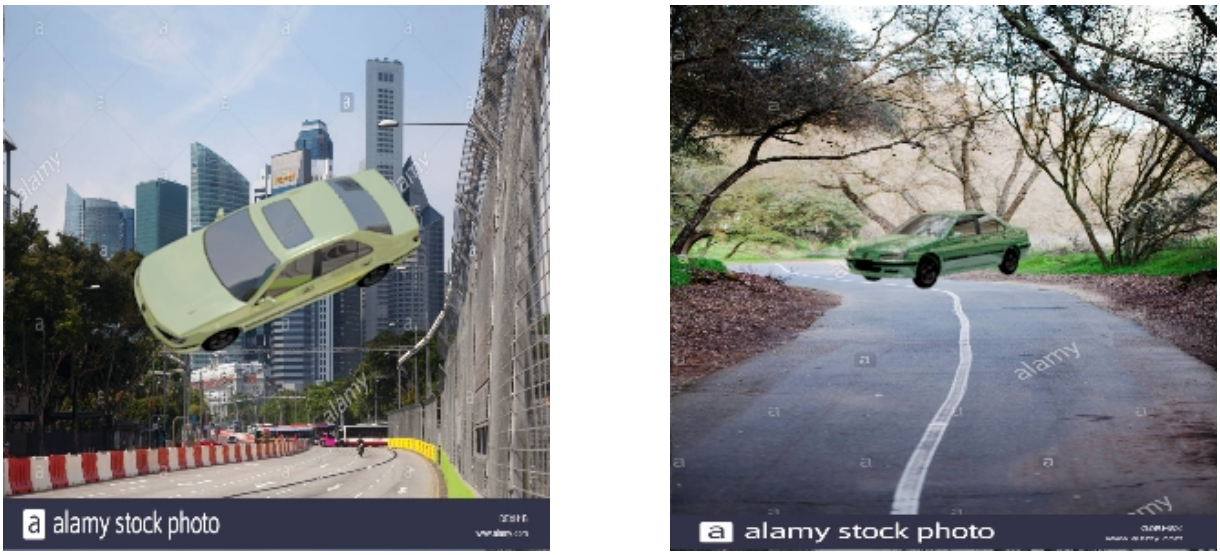


Figure 45: Images generated using uniform background images, and a Peugeot 3D model.

4.2.9 Summary

In a high level perspective, images are generated via ground and sky or whole background, and then combined with the 3D model and the scene parameters. The output is a synthetic dataset with annotations, that can be used in the train pipeline.

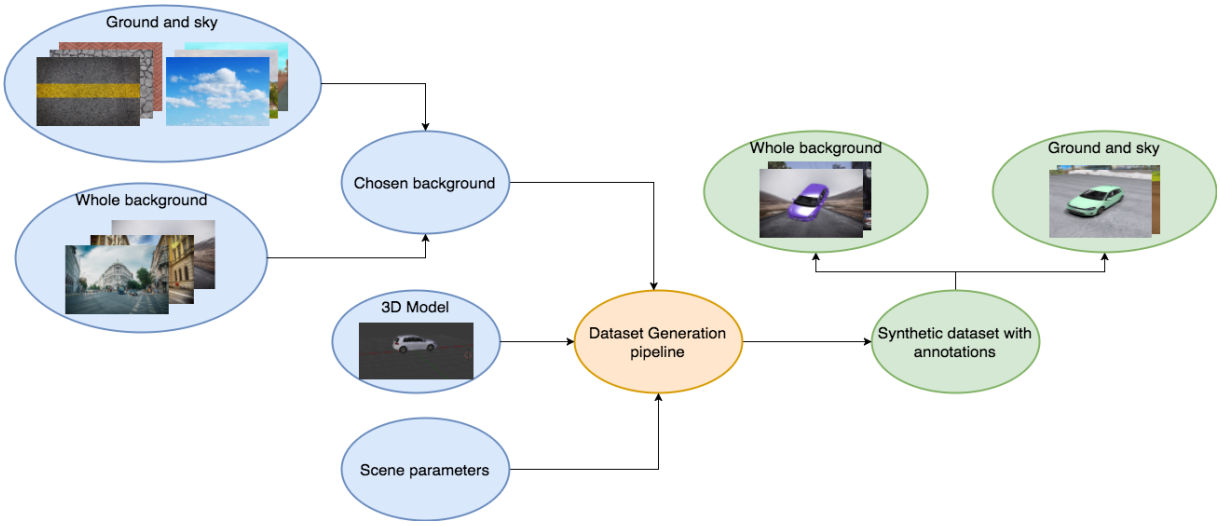


Figure 46: Overview of the synthetic dataset generation process. Blue distinguish input data, orange means that it was developed in this work and green is used for the output of the pipeline

4.3 Train pipeline

A dataset was created containing real images, downloaded using the same script that allows to obtain the background images section 4.2.4 but in this case by searching keywords containing Volkswagen Golf. The script allows to execute multiple queries by using the keyword "Golf" and concatenating different words from a list. The images downloaded are then manual checked (from 300 images, 283 were relevant) and afterwards the bounding boxes are annotated. The interesting thing is that this annotations are first suggested by reusing a trained network, so the user just needs to verify that the proposed bounding box matches the object or slightly adjust it. This results in a real test dataset of 283 images, with each of them containing one or more Golf cars.

(a) Cropped Golf

(b) Different car models

(c) Two Golf models

Figure 47: Images extracted from the real dataset of 284 images.

4.4 Summary

The synthetic train dataset created (either ground and sky, whole background or a combination of both) is used as the train dataset. Combined with the test dataset and train parameters, they constitute the input of the train pipeline. There are multiple parameters involved in training, but the parameters chosen were the ones used by SSD in PASCAL VOC [10]. The important parameters that did change are:

- Gamma: 0.01. This parameter indicates how much the learning rate should change every time it reaches the next step. The value is a real number, and can be thought of as multiplying the current learning rate by said number to gain a new learning rate.
- Stepsize: 20000. This parameter indicates one of potentially many iteration counts that moves onto the next step of training. This value is a positive integer.
- Iterations: 50k. Number of train iterations.
- Learning rate policy: step.
- Momentum: 0.9 This parameter indicates how much of the previous weight will be retained in the new iteration.
- Weight decay: 0.0005. Factor of (regularization) penalization of large weights.
- Average precision: 11 point. For computing mAP, explained in chapter 2: Basics).
- SSD 300
    - Train batch size: 8
    - Test batch size: 4
- SSD 512
    - Train batch size: 2
    - Test batch size: 2

The batch size determines how many images will be fed through the network in a single iteration. It should be the highest possible, so more images will affect the new weights update (and therefore avoid overfitting). Batch size is limited by GPU. Note that when the size increases, the batch size needs to be reduced. For a complete overview of the parameters, refer to [10]. With the input provided, the train pipeline generates the networks and run caffe train, in a reproducible and automatic way. The output of the training phase is an experiment that contains the trained weights.

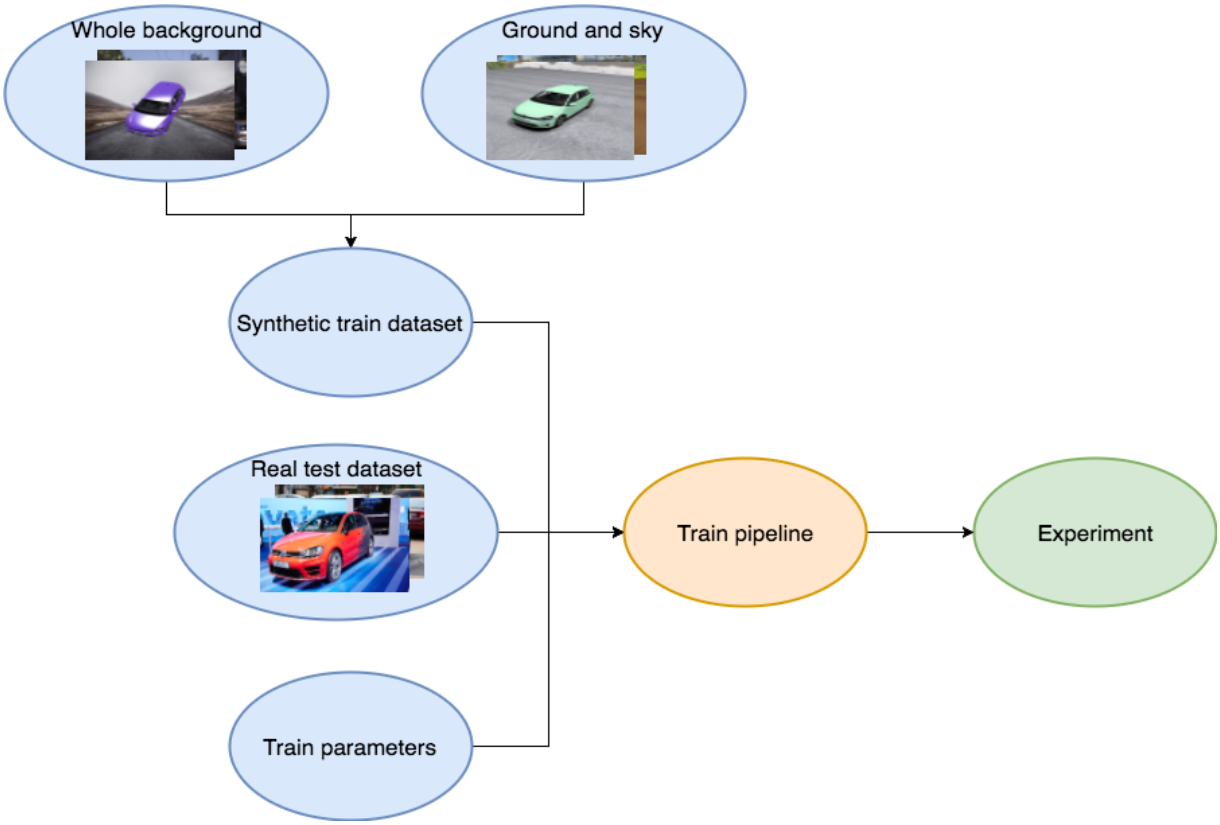


Figure 48: Overview of the training pipeline. Blue distinguish input data, orange means that it was developed in this work and green is used for the output of the pipeline. The arrows determine the information flow.

4.5 Detection pipeline

The experiments created by the train pipeline (that contain the trained weights) allows to run detections on other images. As already mentioned in chapter 2: Basics, CNNs are fast when deployed, because every image is just a forward pass through the network. After the forward pass, the output detection are written to a file. The detection pipeline allows to supply an option to create a new cropped image with the estimated bounding box and the original image, or to draw the bounding box with the confidence over the image. It also supports a video as input, for which it split the frames, detect as explained before, and then join the frames again into a video with detections.

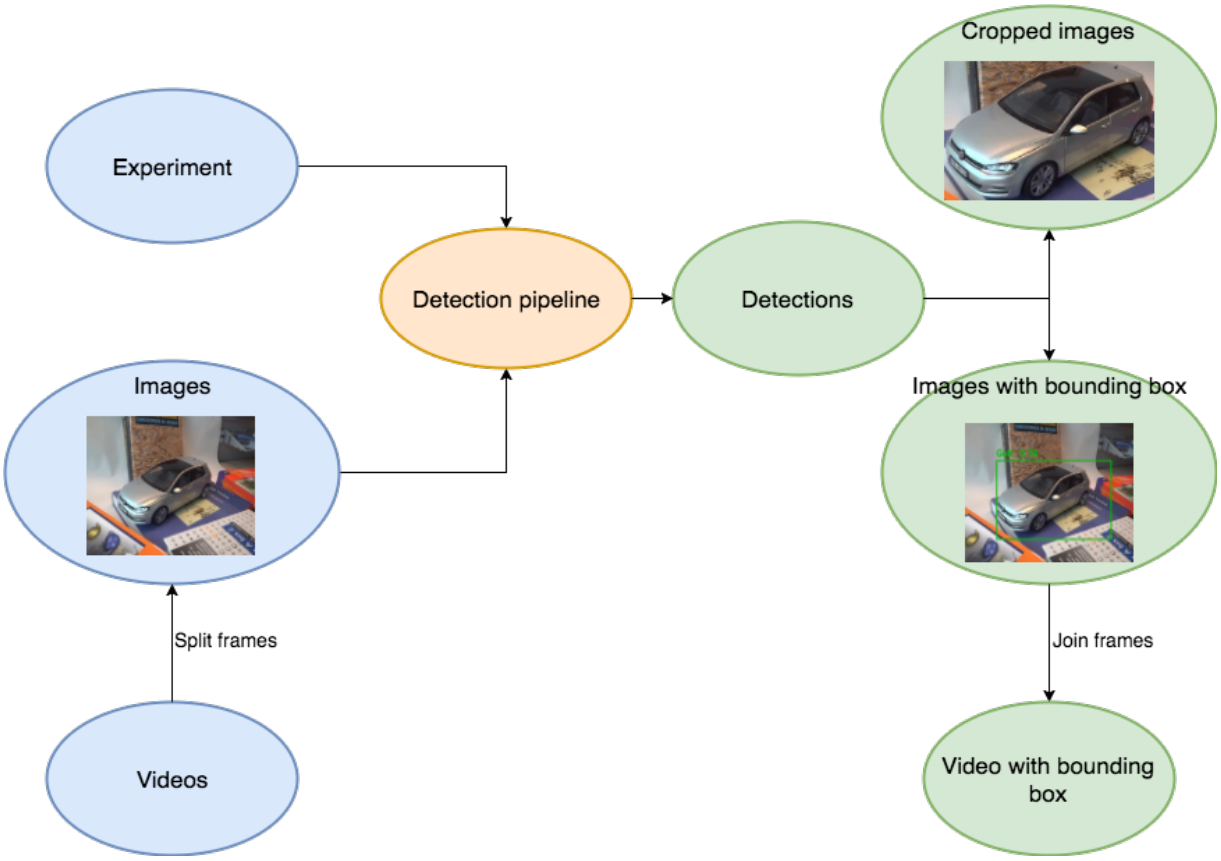


Figure 49: An experiment(output from the train pipeline) is the input of the pipeline (blue). The detection pipeline outputs the detections, and can also crop the input image or draw the bounding boxes (green, output of the pipeline).

4.6 Visualization pipeline

4.6.1 Layers Visualization

Visualizations are one of the most important ways to understand what (and why) is the CNN learning. For example, the training visualizations help to understand the required iterations for which the accuracy will not go higher, or if there is a local minimum. Visualizing the layers help to understand what is each layer learning and what components of the input images are producing the highest activation values.

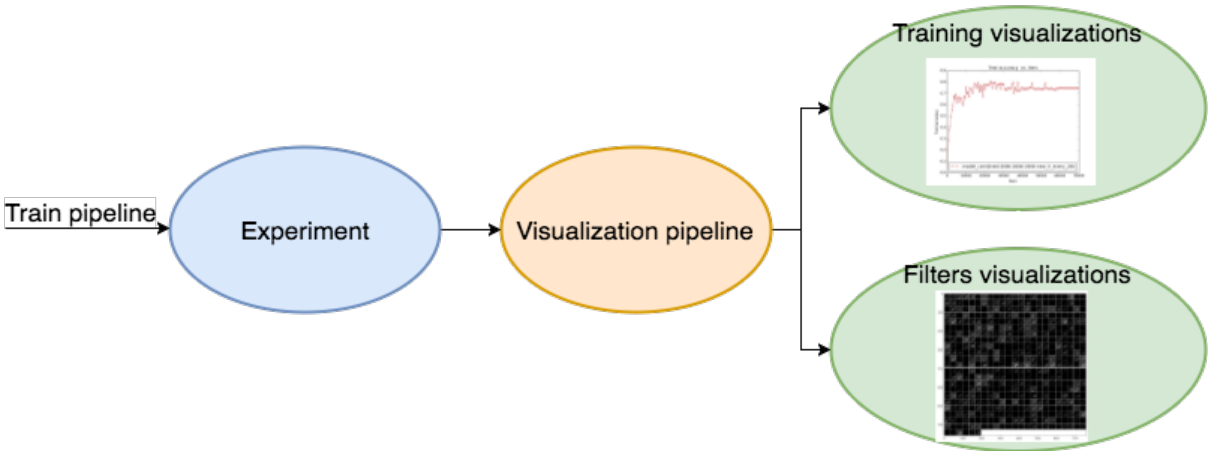


Figure 50: An experiment (output from the train pipeline) is the input of the visualization pipeline (blue). The visualization pipeline process the different visualizations and saves them to the experiment (green, output of the pipeline). Orange refers to components implemented in this thesis.

First convolutional layer

Each square of fig. 54 refers to the activation map of a filter of the first convolutional layer. An input image is feeded through the network to visualize which features produce a high activation map on these filters. White sections represent high activation values, and black sections low ones. Notice that because it is the first convolutional layer, it extracts low level features, such as edges. In addition, the images are still big because they only went through one convolution.

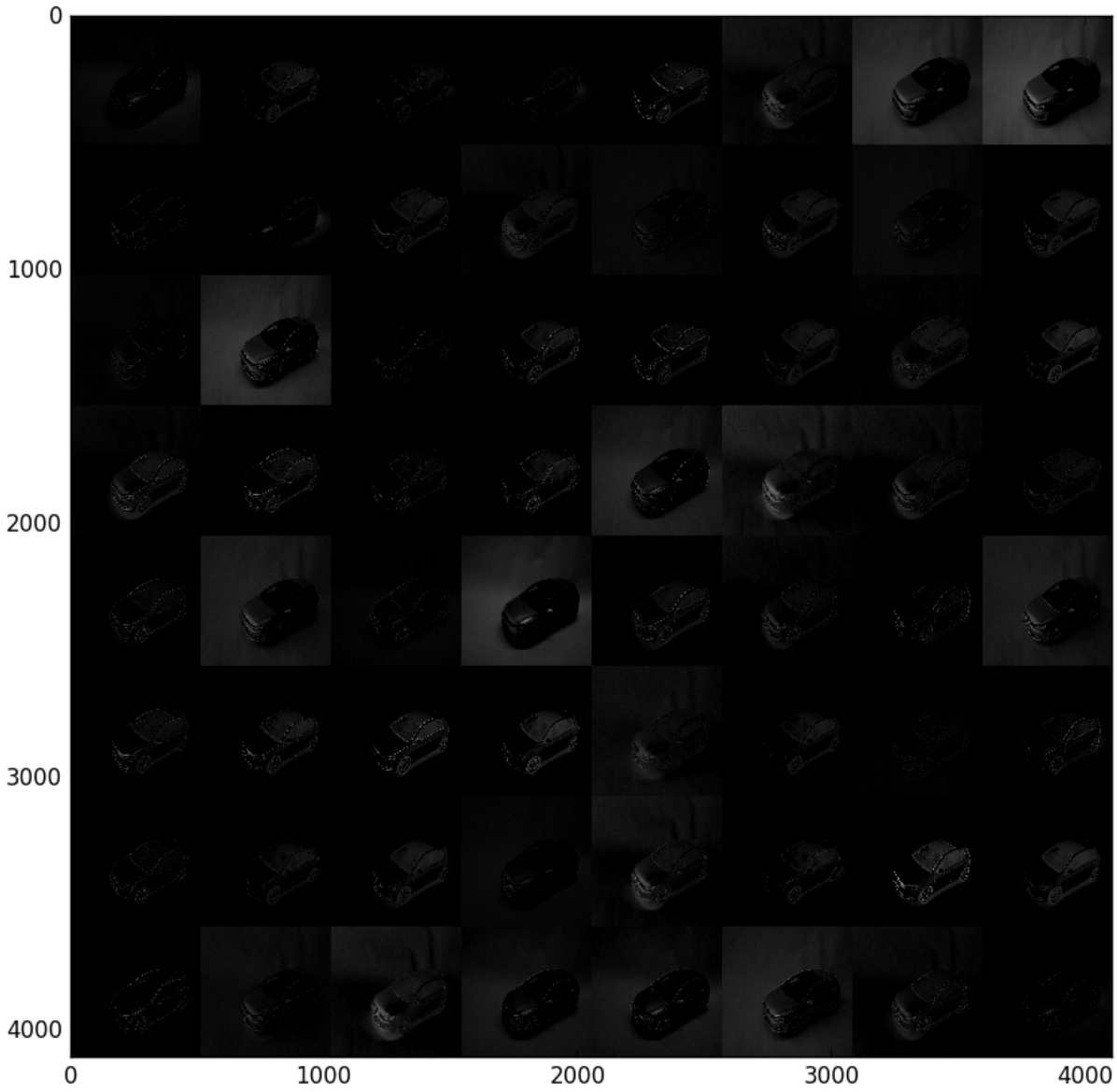


Figure 51: First convolutional layer. Each square represents the activation map of a different filter. White means high activation values, while black means lower ones.

24th pooling layer

fig. 52 Represents the activation map of the filters of a layer deep in the network (24th pooling layer). Notice how the image got reduced (because of the applications of convolutions and pooling, explained in section 2.5: Convolutional Neural Networks (CNNs)). The white sections represent high activation values, and black sections low ones. The detector is already able to detect deeper features, in this case, the Golf as a whole. Also, the results are sparse, which is a good sign while trainning a network. This is because in deep layers filters that detect instances from the same class are usually grouped together. In addition, the number of filters that detect Golf and the number of filters that detect background (black squares) is almost similar. This is

a property of accurate networks: the number of filters is approximately divided into the number of classes (in this case two, background and Golf).

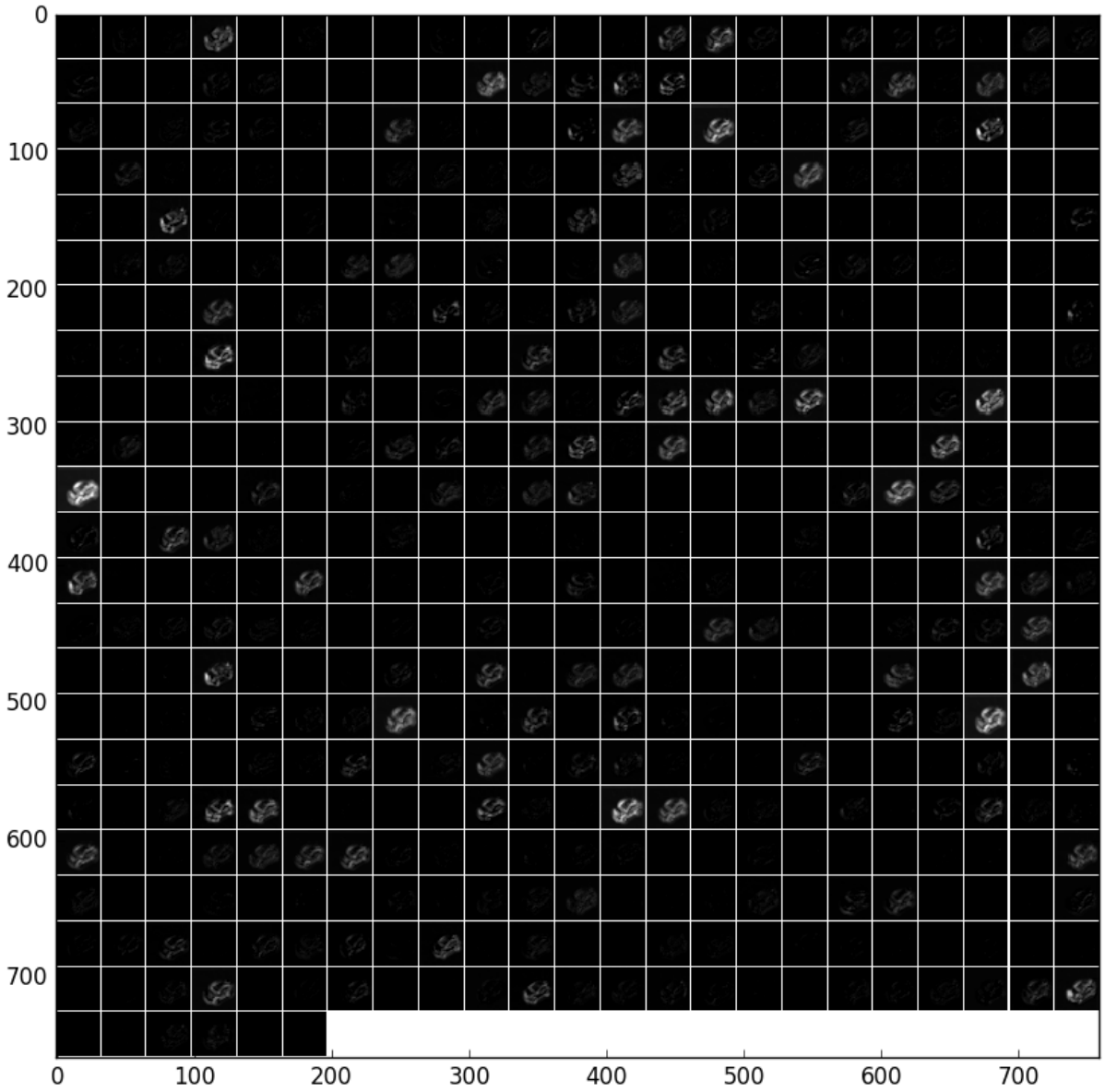


Figure 52: 24th pooling lyaer. Each square represents the activation map of a different filter. White means high activation values, while black means lower ones.

4.7 Viewpoint estimation pipeline

The object detection module belongs to a bigger system, which offers the possibility to create synthetic datasets and viewpoint estimation based on synthetically generated images. Therefore, it was structured in a way to be able to interoperate with both modules. This dataset model can also be converted to the Dataset model that the external synthetic generation tool

knows, so it is straightforward to communicate with it. Figure 53 shows a high level perspective of how the object detection module interacts with external systems. Object detection receives input images (or videos), and outputs the detections of the bounding boxes. It is flexible enough to either draw the bounding boxes on the output images or crop the images. The viewpoint estimation module [14] receives the input image and process it to obtain the viewpoint estimation. After this has been done, the viewpoint estimation is used to recreate the detected car in the 3D scene. Object detection also supports videos as input, the frames are processed the same way as explained before, and there is an extra step where the recreated frames are concatenated to obtain a video of the 3D model.

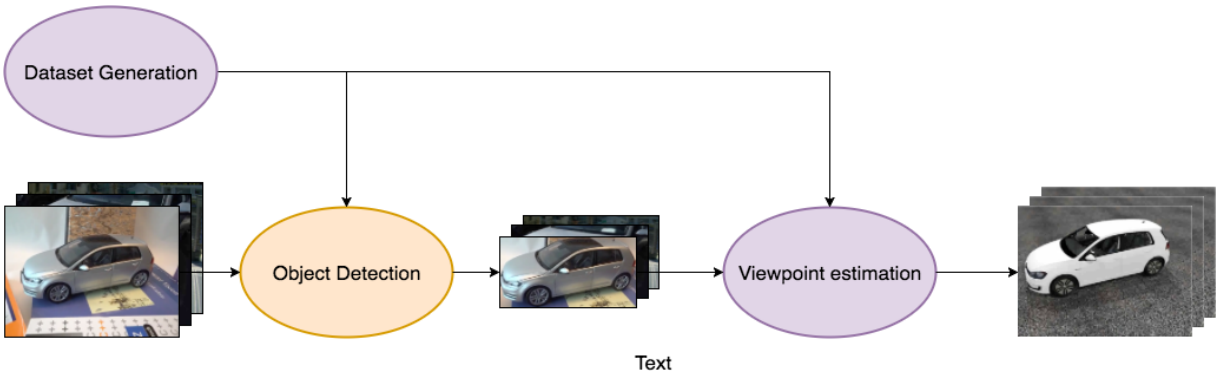


Figure 53: Shows the interaction of object detection with external systems. The synthetic dataset generated by the dataset generation tool is used to train both object detection and Viewpoint estimation [14].

5 Results

This chapter describes the results of using synthetic generated images for real world object detection. As stated in chapter 4: Experiments, the results measure the accuracy (mAP) of the object detector when using different backgrounds. Further variations where included, such as generating and adding a dataset of a Peugeot 3D model or increasing the input resolution from 300 x 300 to 512 x 512. Even though two models of cars were used (Golf and Peugeot 406), the experiments were focused on detecting Golf.

5.1 Ground and sky

Resolution	**Train dataset size**	**mAP**
300 x 300	150k	0.644
512 x 512	200k	0.128
300 x 300	50k	diverge

Table 2: Shows the accuracy (mAP) of the detector on the test dataset, according to resolution and dataset size when using only a ground and sky type of image for training.

5.2 Whole Background

Resolution	**Train dataset size**	**mAP**
512 x 512	200k	0.556
300 x 300	200k	0.510
300 x 300	150k	0.489
300 x 300	100k	0.462
300 x 300	50k	diverge

Table 3: Shows the accuracy (mAP) of the detector on the test dataset, according to resolution and dataset size when using only a whole background type of image for training.

5.3 Combined

3D model used	Resolution	Whole bg,	Ground and sky	Train dataset size	mAP
Peugeot and Golf as same class	300 x 300	300k	300k	600k	0.839
Golf	300 x 300	150k	150k	300k	0.747
Golf	300 x 300	400k	150k	550k	0.723
Golf	512 x 512	200k	200k	400k	0.718
Peugeot and Golf as different classes	300 x 300	150k	150k	300k	0.341

Table 4: Accuracy (mAP) of the detector on the test dataset, according to resolution, dataset size and a combination of backgrounds used for training. In the first case, the detector was trained with both images of the Golf and Peugeot as a single class (the output classes are still a single car and background). That setup achieved the best accuracy (0.839 mAP). In the last element, Peugeot was introduced as a different class, so the output classes are 3: Golf, Peugeot and background.

6 Discussion

6.1 Influence of dataset size

6.1.1 Minimum dataset size

Ground and sky (table 2) and whole background (table 3) show that with 50k the training diverges, meaning that the network can't learn any characteristic from the input dataset. They also show that 100k is an effective minimum dataset size for training with synthetic images generated this way. This will depend on the specific model (the 3D model size matters), and cannot be extrapolated to other problems, though it sets a good baseline for future applications using synthetic images.

6.1.2 Upper bound on incrementing dataset size

Though the previous point mentions that increasing the dataset size makes the detector better, there is an upper bound. It can be appreciated in table 4 where bigger combined datasets were produced but didn't increase the detector accuracy as it happened with ground and sky and whole background. This relays on that the detector already learned the features that both type of images can represent, so adding more images don't contribute to a better learning. The ideal case would be to design a new scene in a different way, just how whole background was thought of. In addition, the need of introducing variations is important here: adding more textures for the background or adding new 3D models of cars (like Peugeot) would be beneficial.

6.1.3 Relation between dataset size and type of background

It is interesting to note the relation of the dataset size to the type of background. As can be seen in whole background (table 3), datasets were created by adding 50k images that produced datasets of 50k, 100k, 150k and 200k. For 50k, the trained model couldn't learn (the learning diverged). Increasing dataset size (100k, 150k, 200k) made the model more accurate, which reinforced the fact that (for CNNs) the availability of data is more relevant than the specific network architecture being used (refer to chapter 2: Basics, section 1.2: Why weren't CNNs introduced before?). It is also important to notice that the nature of the images (whole background) is highly suitable for increasing the dataset size, because more information is gained from the background than ground and sky.

6.2 Training image resolution

6.2.1 Whole background

Image resolution has a big impact in detectors. For real world images, the increase of image resolution will make the detector more accurate, as shown in SSD [10] and YOLO [9]. It is straightforward to understand why this happens: if the image resolution is higher, the detector can learn better the low level features (and reinforce the high level ones). For synthetic images, it doesn't behave this way. When trained with lower images resolution, objects contained in synthetic images resemble real images: it is simple to identify if there is a car (just by learning the features that makes it a car). At the same time, as the experiments contain a specific model of car (Golf), distinguishing between it and other type of cars is extremely difficult. The detector isn't able to learn those low level features.
Furthermore, this increase of accuracy highly relates with the background type: in whole background, in contrast to ground and sky, there is more information gained from the background when training the detector. It it worth to mention that the detector learns to distinguish between two classes: Golf or background. In this case, the background information makes the detector better not by proposing more positive cases for the Golf class, but by gathering more negative cases that belong to background class.
As image resolution increases, more details are added. That extra information has two effects:

1. The detector is better when differentiating between different type of cars (Less false positives).

2. The detector predicts more precisely the bounding box.

6.2.2 Ground and sky

While analyzing the quantiative data, one interesting aspect arises: higher resolution images that were created using ground and sky background do not make the detector better. In fact, the 0.128 mAP score of the test dataset shows that it is the other way round: incrementing the input resolution of the training images negatively affect the detector. This is related to the information that the ground and sky background provides: as it is always two different textures (with changes in lighting), by increasing the resolution there isn't a gain in the accuracy. In other words, there is no information gain by increasing the resolution: instead, it is more probable that the network learns details from the synthetic images that aren't in the real ones.

6.3 Combining different backgrounds

Table 4 shows that when combining different backgrounds, better accuracy is achieved. This is what was expected: by using different backgrounds, more information is added to the detector.

When using just ground and sky, the detector would incorrectly learn that every Golf needs to be over a uniform background. However, it also would learn that a Golf should always be over a surface. In the case of whole background, the detector learns more details by the differences in the background, but it cannot be guaranteed that the model would be correctly positioned over a plane.

6.4 Size of the 3D model

The size of the 3D model plays an important role in the accuracy (mAP) that SSD [10] can achieve (and region based CNNs in general). According to [10], SSD is very sensitive to the bounding box size. In other words, it has much worse performance on smaller objects than bigger objects. This is not surprising because those small objects may not even have any information at the very top layers. Increasing the input size (e.g. from 300 x 300 to 512 x 512) can help improve detecting small objects, but there is still a lot of room for improvements. On the positive side, SSD [10] performs really well on large objects, so that is also a reason of why object detection for Golf obtained a high accuracy.

6.5 Detecting a specific model

Training a model on a specific model of car rather than training it to detect cars in general didn't result at all. In order to understand what would happen when training with an additional car model, an open source Peugeot 406 model was added, but in two ways:

- Single class: The Peugeot 406 and Golf together constitute a single class. This means that a correct detection would refer to the proper bounding box of the car and the car being a Peugeot or a Golf.
- Different classes: Peugeot was added as a new class. The detector then distinguishes between 3 classes, Golf, Peugeot 406 and background (or none of the others).

Table 4: Accuracy (mAP) of the detector on the test dataset, according to resolution, dataset size and a combination of backgrounds used for training. In the first case, the detector was trained with both images of the Golf and Peugeot as a single class (the output classes are still a single car and background). That setup achieved the best accuracy (0.839 mAP). In the last element, Peugeot was introduced as a different class, so the output classes are 3: Golf, Peugeot and background. shows the results for both of the alternatives. It can be seen how beneficial it was for the single class case, obtaining the best accuracy (0.839 mAP). Detecting cars results in an easier task than detecting a specific model, and by adding the characteristics of the Peugeot, the detector gets better. This shows that it is really difficult to train a detector in specific models: bigger dataset are needed. Using synthetic datasets is even worst because of the difficulty to replicate those low level features that different cars exhibit.

In the other hand, having the Peugeot as a different class critically reduces the performance (0.341 mAP), the worst accuracy obtained by combined datasets. The explanation for this is that it is really difficult for the network to distinguish from Golf to Peugeot because they exhibit similar features. The first layers, that extract low level features, will have similar activation maps when faced with Golf or Peugeot. Distinguishing between them is done in the last layers. To effectively train a network to this level of detail, more complex synthetic images than the ones generated in this work are needed (which will probably require more time to create).
The detector knows just about Golf and background. While analysing other type of car, it is highly probable that it will classify it as Golf. This would happen even to human vision: if you only to know to distinguish a Golf and background, you will probably decide that a Peugeot 406 is a Golf. In other words, the network should be also trained to detect other type of car models (by supplying the corresponding 3D model).

6.6 Analyzing precision-recall curve

The precision-recall curve is the most important metric to asses how good the detector is. It is used in most of the datasets. There is always a tradeoff between choosing the precision-recall curve and the threshold. An analysis must be done by modifying the threeshold, and evaluating how does the precision-recall curve vary. This analysis depends on the specific problem (explained in section 2.7: Accuracy). Figure 54 shows how precision-recall vary in the experiment of Golf and Peugeot as a single class (0.839 mAP). The ideal is to be at the upper right as possible (as it means a higher precision and recall), but the threshold will be lower. So the threshold value should be decided as the highest possible while still achieving expected precision and recall.

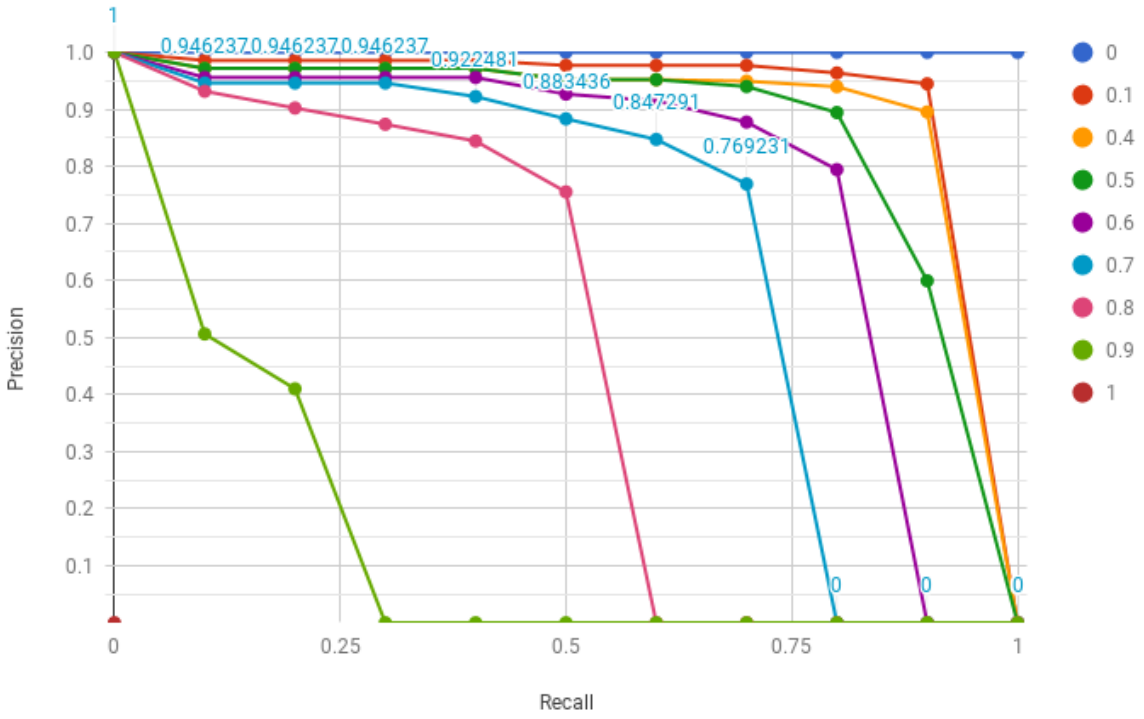
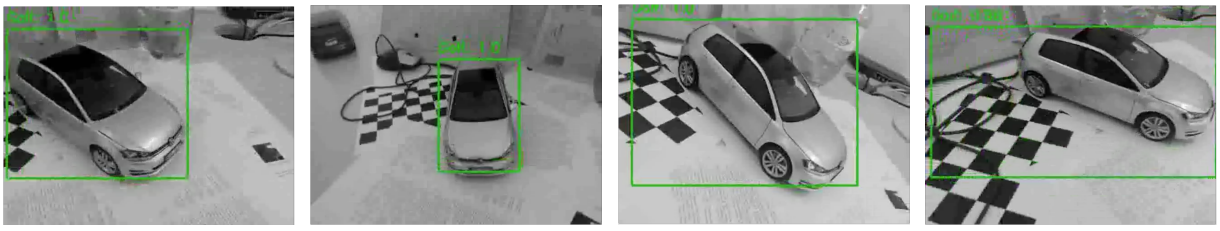


Figure 54: Visualization of the precision-recall curve for the best entry of table 4 (Golf and Peugeot as a single class) which achieved 0.839 mAP. This figure shows how do precision and recall change according to the chosen threshold. The treshold values are listed on the right columns, and they are changed from 0 to 1 by steps of 0.1.

6.7 Color invariance

Black and white images and videos were used to test the color invariance of the trained model. The model worked correctly in all of them (with correctly meaning that the IOU between the prediction and the ground truth bounding box was at least 0.5 in all of the cases, metric proposed by [4]).



(a) Correct detection from the side.

(b) Correct detection from the front.

(c) Correct detection when camera is rotated.

(d) Notice the influence of the ground pattern at the left.

Figure 55: black and white images showing the correct detection of Golf.

However, there is still influence of the color that can be appreciated for the stability of the bounding box. The colors that affect the detector are those of the background and not the Golf itself. While thinking about this on a semantic level, the behaviour is not incorrect. It depends on which the problem is, because there are some cases that it would make sense to extract information about colors. Just to exemplify this, the yellow color of the sun helps humans to detect that it is precisely the sun, so in that case color is a valuable information.

6.8 Detections from different angles

The detector proved to support detections from different angles, by analyzing a Golf exhibition. Also notice how the precision is related to the quality of the picture: the detections are very good (more than 0.5 IoU).

Figure 56: Shows correct detections from different angles. Detector applied to a Volkswagen Golf exhibition.

6.9 Close up detections

One important aspect to consider is about closeups on the Golf. The shape of the Golf itself may be lost, so it is a difficult task to still detect it correctly. Even for humans it would be difficult to differentiate between a Golf and other model by visualizing just a part of the car, like the door or a window. The detector outputs correct predictions from close up images in general, as seen in this frames taken from a Volkswagen Golf exhibition.

Figure 57: Shows correct detections with a medium distance close up. Detector applied to a Volkswagen Golf exhibition.

Figure 58: Shows correct detections when the viewer is close. Detector applied to a Volkswagen Golf exhibition.

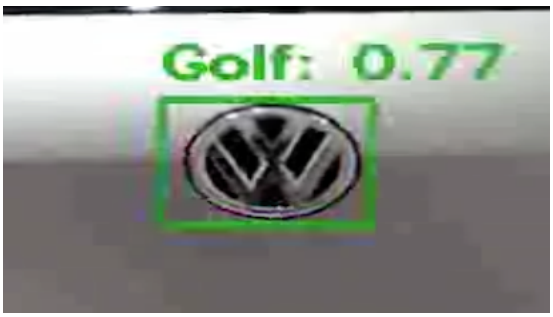
6.10 Influence of subcomponents

By learning hierarchical features, CNNs are able to detect an object via the presence of certain characteristics of its components. In other words, there are some filters that will have a high activation map when encountered with certain aspects of the object. This is the case for wheels: every car has wheels, so the CNN learns this characteristic. This is why the presence of wheels affects detecting objects which also contain wheels: it is difficult to distinguish between different objects that contain the same component. When more of these components are shared by both objects, it will be more difficult for the network to distinguish between them. Though this seems to be counter intuitive, it reflects a valid point with how we learn. Suppose that you were taught just about two type of objects: cars and multiple backgrounds. If you were asked to classify a bicycle, it is probable that the answer would be car, because a bike is more similar to a car than to the background. The same happens with the trained CNN: when it encounters some characteristic present in a Golf, it would probably think that it is indeed a Golf (not completely because it detects a non-linear combination of features, but still there is a high influence of the wheels). There are three ways to reduce the influence of specific components of an object for detection.

1. Provide different models with that component for training (different cars in this case).
2. Introduce variations on the training images, like occlusions (occluding the wheel in some images).
3. Add images to the background which contain that component (add backgrounds with cars).

With all of these steps, the influence of the component is reduced. The problem arise in that it may not be advisable to do this, because it will also negatively affect the precision of the model, it will require more training (because of the need of more detailed detection) and training (and more detailed) data has to be increased.

(a) Example of detection of the wheel. The bounding box should surround everything from the car, not just the wheel.



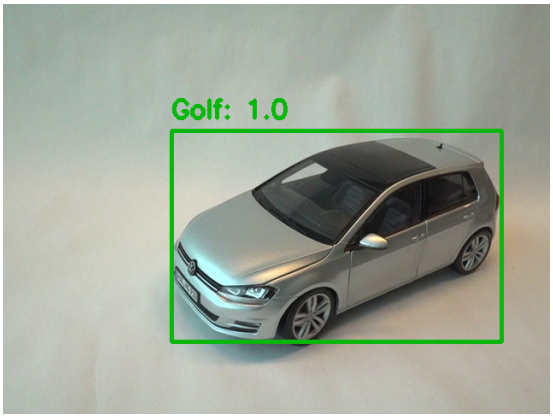
(b) Incorrect detection because of the Volkswagen logo. Every training image contained a logo, that's why sometimes the detector misdetects the logo as the car.

Figure 59: Shows that the CNN can learn that specific subcomponent of an object must be present in a car (which is not completely incorrect). Training images should also be generated by occluding those subcomponents in some of them.

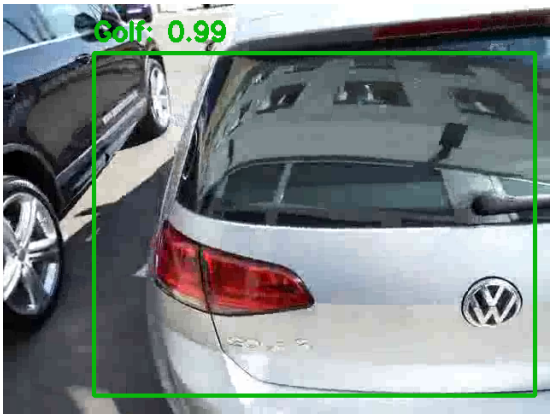
Figure 60: It is hard for the network to distinguish between a Golf and other type of models, as share similar subcomponents(like wheels).

6.11 Accuracy of the bounding box

From the qualitative analysis made on videos, it can be seen that the bounding box isn't generally stable. For videos that contain no background as fig. 61a, the predicted bounding box is usually stable between the different frames. It is worth to mention that frames are processed separately, so a new estimation is made for each frame. For other videos the predicted bounding box usually varies, but it it still consider a good detection (good refers to an IOU greater that 0.5).



(a) Simple detection (as the background is uniform). The detector shows stability on the bounding box when processing this kind of images.

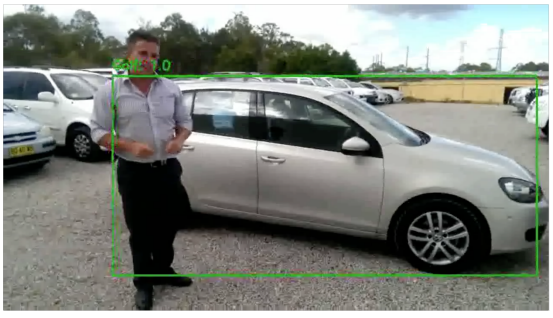


(b) Complex detection: a close up of a Golf with a part of other car contained in the image. The low quality of the image also affects negatively the detector, although it achieves a good detection.

Figure 61: Shows a comparison of the accuracy of the bounding box. While it still achieves a good detection in both images, the one of the right is not so accurate as the one on the left. This affects specially videos, because of the missmatch between the bounding box for each frame.

6.12 Resistance to occlusion

The detectors show resistance to occlusions, that can be appreciated in the qualitative images and videos. It is important to note that in the training datasets there are no occlusions, so the detector is not explicitly trained to detect them. It can still detect cars in these circumstances because there are details of the car being visible (like the wheels, windows and overall shape).



(a) Detection is correct even though there is a person in front of the car. The Golf model is also an older one, and the detector still is able to correctly predict the bounding box.

(b) Image taken from an exhibition and then fed through the network. The detector correctly predicts the bounding box, even though there is a person in front the car.

Figure 62: Examples of detections when there are occlusions.

6.13 Unexpected mistakes

There are some mistakes for which it was difficult to find an explanation. This may be related with the combination of colors, or some aspect of the images that resemble Golf cars. What is hapenning is that the detector understands that these images are more similar to the Golf class rather than the background one. This suggests that it would be a good idea to add more random images (noise) to the training dataset with negative examples (to be part of the background category).

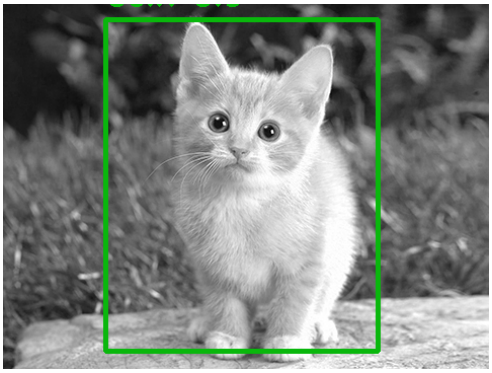


Figure 63: Incorrect detection of a black and white cat image.

Figure 64: Incorrect detection of a panda.

6.14 Testing with synthetic dataset

Viewpoint estimation [14] proposes to generate a synthetic test dataset for each training dataset, in the same way as training images are generated but using different characteristics (such as textures and light conditions).

This approach has some fundamental flaws: if test datasets are generated synthetically, there is no certainty that a detector which learns to detect synthetic images is also effective in real images. Even worst, if test dataset is generated the same way as training images, there is a great probability that the detector learns the features that those type of images describe (overfits on the training data). It is true that, if the color and light conditions are varied, the detector would be invariant to those characteristics. However, it is incorrect affirm that when faced with different images, it will continue to have the same results. This is mainly because in the image space (space that contains all possible images), the training and testing images will be near because they were created the same way. Last of all, using a different synthetic test dataset for each training dataset makes it impossible to reliably compare accuracies.

Resolution	Train dataset size	Synthetic test dataset mAP	Real test dataset mAP
300 x 300	200k	0.94	0.51
300 x 300	150k	0.93	0.489
300 x 300	100k	0.91	0.462
300 x 300	50k	diverge	diverge

Table 5: Results of testing images with background type whole background on a synthetically generated dataset. Notice the difference between the accuracy according to the synthetic test dataset and real test dataset. This clearly shows that test dataset must not be synthetic.

Resolution	Train dataset size	Synthetic test dataset mAP	Real test dataset mAP
300 x 300	150k	0.98	0.644
512 x 512	200k	0.99	0.128
300 x 300	50k	diverge	diverge

Table 6: Results of testing ground and sky on a synthetically generated dataset. Notice the difference between the accuracy according to the synthetic test dataset and real test dataset. This clearly shows that test dataset must not be synthetic.

6.15 Differences with pose estimation

6.15.1 Minimum dataset size

For the initial experiments, 50k was initially chosen as the number of synthetic images generated for the training dataset (as recommended in [14]). As shown in in both type of backgrounds, the learning diverged (in other terms, the model couldn't learn anything). It is interesting to understand why this happens. There is one main point that differs between object detection and viewpoint estimation [14]: the influence of the background is greater in object detection. In viewpoint estimation, the cropped image is the input of the system, but in object detection the whole image is the input. Consequently, the percentage of background that will affect the detector is higher. This makes the task of object detection harder, and therefore bigger datasets are needed to counter the challenges of different background conditions.
In [14], images generated that via ground and sky backgrounds turned out to have good results on estimating the pose of real objects [14]. While at first it seems that those type of images could also be used to train an object detector, there are significant differences between the task of estimating pose and detecting an object.

6.15.2 Object detection is harder

The domain of pose estimation is different to the domain of object detection. Both have images as their input, but because of the first point (presence of background) object detection has a bigger input space. In others words, there is more variation for the input images of object detection. This makes object detection a harder task that pose estimation.
There is another critical point, which is based in how object detectors work. When an input image is used for training, a common technique is to sample negative elements from the background, which is called hard-mining. This is because every object detector need a class (usually called background or none of the others) to default in case that there isn't a detection. In others words, a region of the image that doesn't contain the object of interest should map to the background class. In viewpoint estimation, there is no background class (there are always three angles that describe the pose of the object).
Last of all, the problem with using ground and a sky is that object detection algorithms would learn that every car should be over this uniform ground and a sky. In the real world, there are usually other type of objects present, such as building, cars, people, streets and signs. This would make the detector fragile for detecting on real images.

7 Conclusions

This work describes an approach to train an object detector with synthetically generated images. To be able to compare the results, the detectors were evaluated with a dataset of 284 real Golf images, which contained one or more objects. As mentioned in chapter 4: Experiments, the architecture was fixed, focusing on the dataset size, the input resolution and the type of synthetic generation used (whole background or ground and sky). Using synthetically generated images proved to be a successful approach for object detection. The results show that the best detector achieved 0.839 mAP. In average, the combined detectors could achieve 0.75 mAP, while ground and sky and whole background achieved 0.6 mAP and 0.56 mAP respectively. These are high values for object detection tasks (SSD [10] and YOLO [9], considered the best alternatives, score around 0.7 mAP in PASCAL VOC [4]). This work demonstrates that an object detector, trained solely with synthetically generated images, learns to detect images with a high accuracy in the real world. This work also shows that it is important to generate images in different ways and then combining them. To picture this, whole background obtained 0.6 mAP and ground and sky 0.56 mAP. When combining both of them, the information provided by both type of images incremented the results to 0.839 mAp. The analysis over the dataset size established that there is a minimum size required (100k in object detection) and that there is an upper bound when using synthetic images. Regarding training image resolution, even though in real images greater image resolution makes a detector better, this work exhibits that for synthetic images it is not always true. The network proves to be resistance to different color conditions of the 3D model, correctly detected cars from different angles and still detected details in close up detections. The detector showed to be accurate when a Golf was present in the image, but still made mistakes in some cases, specially when other cars were present. This work also showed that it is more difficult to detect a specific model of car (Golf) rather than the more generic car class. Adding synthetic images of a Peugeot model (with both ground and sky and whole background) achieved great results when detecting a single class (Golf + Peugeot). On the other hand, when trying to differentiate between those models, the detector obtained the worst results, showing the difficulty of distinguishing similar objects. Finally, the detector was combined with pose estimation, therefore obtaining objects bounding box and their viewpoint estimation, and using them for recreating the Golf in a virtual scene.

7.1 Relevance

The results here obtained impacts not only object detection, but computer vision tasks where any variatons of neural networks are used. This work demonstrates that simple ways of synthetically generating images obtain accurate results when used in the real world ("Theory of the Learnable" [12]). These techniques can be applied to different problems where data is insufficient or is not available. What this work intends to do is to supress the dependancy on the availability of data for training a CNN. Theoretically, if the images exhibit the same features as real images, the CNN that given an architecture computes ($f: hardware, annotateddataset \rightarrow Trainednetwork$) can supress the annotated dataset parameter, resulting in $f: hardware \rightarrow Trainednetwork$. This work explore the fact that availability of annotated datasets (which need manual annotations) increases in a slower pace than performance. The size of the generated dataset is arbitrarily decided by the researcher.

[14] showed, with qualitative results, that synthetically generated images using a single type of background are effective for viewpoint estimation. This work demonstrates, both qualitative (detections in images and videos) and quantitative (detections in a real world dataset of 284 images) that synthetically generated images work also for a harder task as object detection. In addition, a new way of generating images with whole background was introduced (chapter 4: Experiments, section 4.2: Dataset generation), which achieved a remarkable accuracy of 0.84 when combined with ground and sky dataset (considering that for known dataset, accuracies in object detection are around 0.75mAP [4]). Furthermore, this work presents the novelty of combining both types of images, demonstrating that it provides better results and should be the path to follow.

This work also shows that the number of images is highly dependant on the nature and difficulty of the problem. This has an impact on which is the minimum size of the generated datasets (50k for viewpoint estimation [14], 100k for object detection), and how much it takes for image generation and training. To picture this, generating a dataset of 200k images takes 2.5 days when using 300x300 images and 4 days when using 512x512 images. Training the network takes 1 day for 300x300 images and 2 days for 512x512 images, though it depends on the amount of iterations (50k used for the experiments in this work).

7.2 Limitations

Section 6.15.1: Minimum dataset size discusses the number of images required for training and shows that there is an upper bound when using synthetic images. This is an interesting point in the sense that it happens only for synthetic images but not for real ones. This occurs because synthetic images represent a subset of the features exhibit by real images, and once the detector learned those features, adding more images will not provide better results.

It is worth to mention that even though performance (FPS) was mentioned in the analysis of SSD [10] vs YOLO [9], it was out of the scope for this thesis. This is because performance is

highly related with which algorithm is used, in this case SSD [10]. This work intentionally avoids using different detection algorithms to be able to reliable compare the effect of the synthetic datasets.

[23] showed that for small objects, it is more difficult to use synthetic images for training, because there is a greater influence of details (they propose to blend the object onto the scene). The detection algorithm used (SSD [10]) also mentions that detection work better for bigger objects. As the detectors were trained to detect Golf models in this work, it being a big object has a possitive influence, but it is not assured that good results will be obtained for smaller objects.

7.3 Future work

This work focused on using only synthetic datasets for object detection task, and to evaluate the results obtained. As it showed that the model could learn, it would be very interesting to combine synthetic images with real world images for the train dataset. It is true that there are data augmentation techniques widely used, but trying to generate them for scratch is an innovative proposition.

[14] showed that synthetic dataset generation provided good results for viewpoint estimation, and this work exhibited high accuracy for object detection. There is a key factor which wasn't analysed deeply: performance (FPS). Having to train two CNNs takes more time and needs more parameter to tune. The next step should involve evaluating a CNN that is trained to both detect the object and the pose.

In this work, the approach taken for running detections on the videos was to split it into frames, run the detector for each frame and then recreate a video with the detection output. While this works, it has some efficiency issues: It cannot be done in real time, specially because blender takes too long to generate the images (the images that are concatenated into the output video). It would be recommendable to try a game engine, like UnrealCV [22], because the network showed that the importance lay on the size of the datasets rather than on the specific details of the images (so the details that Blender provides but that requires more performance are not needed).

Finally, instead of recreating the object by using the viewpoint estimation, a tool could be created which projects the points onto the input image.

It is really promising that the network achieved this results (0.8 mAP) trained only with synthetically generated images from scratch. Here arise the most interesting aspects. One of them is that new synthetic image generation means could be explored, and will make the object detector better (like whole background improved the results of ground and sky). On the other hand, instead of training the network from scratch, an already trained network that detects cars could be used, fine-tuning it with synthetically generated datasets of the desired model. This way, the general characteristics of a car would already be assimilated by the network (by transfer learning), making it possible to learn the details by retraining only last layers.

Bibliography

[1] Adit Deshpande. A beginner's guide to understanding convolutional neural networks. *[Online] Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks [Accessed: 20.11.2017]*, 2016.

[2] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[4] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[6] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[7] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[8] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[9] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[10] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[11] Manikandasriram Srinivasan Ramanagopal, Cyrus Anderson, Ram Vasudevan, and Matthew Johnson-Roberson. Failing to learn: Autonomously identifying perception failures for self-driving cars. *CoRR*, abs/1707.00051, 2017.

[12] L. G. Valiant. A theory of the learnable, 1984.

[13] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[14] Emilio Tylson. Domain-specific object viewpoint estimation based on convolutional neural networks trained on synthetically generated data. Master's thesis, Fachhochschule Technikum Wien, Höchstädtplatz 5, 1200 Wien, 2017.

[15] Juan C. Caicedo and Svetlana Lazebnik. Active object localization with deep reinforcement learning. *CoRR*, abs/1511.06015, 2015.

[16] Leonardo Araujo Santos. Artificial intelligence, 2017. [Online] Available: https://www.gitbook.com/book/leonardoaraujosantos/artificial-inteligence/details [Accessed: 21.12.2017].

[17] Justin Johnson Fei-Fei Li, Andrej Karpathy. cs231n, lecture 8 - slide 8, spatial localization and detection. *[Online] Available: http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf [Accessed: 22.11.2017]*, 2016.

[18] Ehab Salahat and Murad Qasaimeh. Recent advances in features extraction and description algorithms: A comprehensive survey. *CoRR*, abs/1703.06376, 2017.

[19] Wikimedia Commons. File:typical cnn.png — wikimedia commons, the free media repository. *[Online] Available: https://commons.wikimedia.org/w/index.php?title=File:Typical_cnn.png&oldid=266002912 [Accessed: 21.11.2017]*, 2017.

[20] Wikimedia Commons. File:max pooling.png — wikimedia commons, the free media repository. *[Online] Available: https://commons.wikimedia.org/w/index.php?title=File:Max_pooling.png&oldid=204200335 [Accessed: 20.11.2017]*, 2016.

[21] Sybil P. Parker, editor. *McGraw-Hill Dictionary of Scientific and Technical Terms (5th Ed.)*. McGraw-Hill, Inc., New York, NY, USA, 1994.

[22] W. Qiu and A. Yuille. Unrealcv: Connecting computer vision to unreal engine. *ArXiv e-prints*, sep 2016.

[23] Georgios Georgakis, Arsalan Mousavian, Alexander C. Berg, and Jana Kosecka. Synthesizing training data for object detection in indoor scenes. *CoRR*, abs/1702.07836, 2017.

[24] G. Georgakis, A. Mousavian, A. C. Berg, and J. Kosecka. Synthesizing Training Data for Object Detection in Indoor Scenes. *ArXiv e-prints*, feb 2017.

[25] Blender. [Online] Available: https://www.blender.org/ [Accessed: 27.9.2017].

[26] D. H. MLA Hubel and T. N. Wiesel. Binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology 160.1*, page 106–154.2, 1962.

[27] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, R. E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Handwritten digit recognition with a back-propagation network. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 396–404. Morgan-Kaufmann, 1990.

[28] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[29] J. Matas, O. Chum, U. Martin, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *Proceedings of British Machine Vision Conference*, volume 1, pages 384–393, London, 2002.

[30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[31] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[32] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *ArXiv e-prints*, nov 2013.

[33] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[34] LMDB Documentation. [Online] Available: http://www.lmdb.tech/doc/ [Accessed: 12.9.2017].

[35] Justin Johnson Fei-Fei Li, Andrej Karpathy. cs231n, visualizing what convnets learn. *[Online] Available: http://cs231n.github.io/understanding-cnn/ [Accessed: 22.11.2017]*, 2016.

List of Figures

List of Tables

Abbreviations

CNN Convolutional Neural Networks

SSD Single Shot MultiBox Detector

YOLO You Only Look Once

ILSVRC ImageNet Large Scale Visual Recognition Challenge

VOC Visual Object Classes

mAP Mean Average Precision

FPS Frames per second

TP True positive

FP False positive

FN False negative

AI Artificial inteligence

Appendices

A Background categories

A.1 Categories for whole background synthetic images

- field
- pedestrian area
- grassland
- room
- city
- hiking trail
- street
- highway
- city empty
- downtown empty
- bike trail empty
- crosswalk empty
- landscape empty
- woods
- sky
- street empty
- buildings

– people

– parking lot

– urban

– downtown

– car park

– parking cars

–

A.2 Categories for gathering real images

– volkswagen Golf

– volkswagen Golf parking

– volkswagen Golf exhibition

– volkswagen Golf park lot

– volkswagen Golf city

– volkswagen Golf downtown

– volkswagen Golf buildings

– volkswagen Golf people

B Scene parameters

– Camera:

– DistanceMin: 3

- DistanceMax: 15
- AzimuthMin: 0
- AzimuthMax: 359
- ElevationMin: 1
- ElevationMax: 89
- InPlaneMin: -5
- InPlaneMax: 5
- CameraFocus:
- Name: TargetCamera
- DistanceMin: 0
- DistanceMax: 1
- AzimuthMin: 0
- AzimuthMax: 359
- ElevationMin: 1
- ElevationMax: 89
- ProbMovement: 0.2
- Lights:
- TargetLight:
- Name: TargetLight
- ProbMovement: 0.2
- DistMin: 0

– DistMax: 1

– AzimuthMin: 0

– AzimuthMax: 359

– ElevationMin: 1

– ElevationMax: 89

– LightsNames:

– - Sun

– - Point1

– - Point2

– - Point3

– - Point4

– Sun:

– Name: Sun

– Prob: 0.5

– SkyBlendMin: 0.1

– SkyBlendMax: 0.4

– EnergyMin: 0

– EnergyMax: 0.7

– DistMin: 40

– DistMax: 500

– AzimuthMin: 0

– AzimuthMax: 359

– ElevationMin: 0

– ElevationMax: 20

– PointLamps:

– Name: Point1

– SunPresent:

– EnergyMin: 0

– EnergyMax: 0.3

– EnergyMin: 0

– EnergyMax: 0.9

– DistMin: 20

– DistMax: 30

– AzimuthMin: 0

– AzimuthMax: 89

– ElevationMin: 15

– ElevationMax: 18

– Name: Point2

– SunPresent:

– EnergyMin: 0

– EnergyMax: 0.3

– EnergyMin: 0

- EnergyMax: 0.9
- DistMin: 20
- DistMax: 30
- AzimuthMin: 90
- AzimuthMax: 179
- ElevationMin: 15
- ElevationMax: 18
- Name: Point3
- SunPresent:
- EnergyMin: 0
- EnergyMax: 0.7
- EnergyMin: 0
- EnergyMax: 0.9
- DistMin: 20
- DistMax: 30
- AzimuthMin: 180
- AzimuthMax: 269
- ElevationMin: 15
- ElevationMax: 18
- Name: Point4
- SunPresent:

- EnergyMin: 0
- EnergyMax: 0.3
- EnergyMin: 0
- EnergyMax: 0.9
- DistMin: 20
- DistMax: 30
- AzimuthMin: 270
- AzimuthMax: 359
- ElevationMin: 15
- ElevationMax: 18
- Materials:
- Name: car paint
- Color:
- RedMin: 0
- RedMax: 1
- GreeMin: 0
- GreenMax: 1
- BlueMin: 0
- BlueMax: 1
- Diffuse:
- IntensityMin: 0.2

- IntensityMax: 0.9
- Specular:
- IntensityMin: 0.1
- IntensityMax: 0.3
- HardnessMin: 80
- HardnessMax: 200
- Mirror:
- ReflectanceMin: 0
- ReflectanceMax: 0.9
- Name: ground
- Specular:
- IntensityMin: 0.05
- IntensityMax: 0.1
- HardnessMin: 50
- HardnessMax: 100