Using 3D Graphics to Train Object Detection Systems

Stian Jensen
Andreas Løve Selvik

Master of Science in Computer Science
Submission date: July 2016
Supervisor: Donn Morrison, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Assignment Text

Recent advancements in machine learning, and in particular Convolutional Neural Networks (CNNs), have yielded excellent object detection and classification models. However, these techniques require vast datasets of labelled training images, which are very labour intensive to produce.

As a result, progress in new object domains with many potential applications, such as mushroom detection in forest scenes, is limited by lack of training data.

Based on a recent paper by Peng et. al. [PSAS14] and the performance of object detection algorithms on art [RDGF15], this project will explore reducing the dependence on manually collected and labelled datasets by using 3D graphics to generate datasets which can be used as training data for a model that detects real objects.

The thesis should comprise a literature review of related techniques and propose a framework for data generation, labelling, and model training for a specific class of objects. Evaluations against existing methods should also be performed. Such a framework can then be used as a step towards a system for real-time mushroom detection in forest scenes.

Abstract

Recent advancements in machine learning, and in particular deep neural networks, have yielded excellent object detection models. However, these techniques require vast datasets of labeled training images, which are prohibitively labor intensive to produce.

This thesis explores an alternative approach to obtaining labeled training data, namely using 3D models of objects and modern game engines to generate automatically labeled synthetic training data. A simple approach for generation similar to the one used by Peng et al. [PSAS14] is presented requiring minimal user input, making dataset generation virtually free.

The real-time CNN object detection model You Only Look Once (YOLO) is trained with our synthetic data to detect cars, and its performance is evaluated on real images of cars from the KITTI and PASCAL Visual Object Classes (PASCAL VOC) public datasets, with up to 11.9% and 22.2% Average Precision (AP) respectively. This is significantly lower than state-of-the-art detection systems that use natural image training data, but on par with the winner of the PASCAL VOC challenge in 2008, and we outline multiple avenues for further research that we believe could significantly boost the performance.

Performance of models trained on datasets with different features are evaluated and compared. It is found that aspect ratio, realistic background imagery, and object occlusion are important factors for performance. This is partially contradictory to the findings of Peng et al. [PSAS14] where they find their object detection system to be largely invariant to the background imagery. This discrepancy is likely caused by differences between the two object detection systems employed.

We argue that synthetic datasets can be valuable for training of detectors of novel categories where there is a lack of training data, as well as a technique for controlled experiments to get insight on how Convolutional Neural Networks (CNNs) responds to different attributes in training data.

Sammendrag

Nylige fremskritt i maskinlæring, og spesielt dype nevrale nettverk, har gitt utmerkede objektgjenkjennings-modeller. Disse teknikkene krever imidlertid store datasett med merkede treningsbilder, som er uoverkommelig arbeidskrevende å produsere.

Denne masteroppgaven utforsker en alternativ tilnærming til å skaffe annotert treningsdata, ved å benytte 3D-modeller av objekter samt moderne spillmotorer til å generere automatisk merket syntetisk treningsdata. En enkel genereringsmetode, lignende den presentert av Peng et al. [PSAS14], som krever minimalt med manuelt arbeid blir presentert og gjør datasett-generering tilnærmet gratis.

You Only Look Once (YOLO), en sanntids CNN-basert objektgjenkjenningsmodell, blir trent med våre syntetiske datasett til å gjenkjenne biler. Presisjon på opp til 11.9% og 22.2% *Average Precision (AP)* blir målt på ekte bilder av biler fra henholdsvis datasettene KITTI og PASCAL Visual Object Classes (PASCAL VOC). Dette er signifikant lavere enn state-of-the-art detekteringssystemer som benytter store datasett med naturlige bilder som treningsdata, men på nivå med vinneren av PASCAL VOC i 2008 samt andre metoder når tilgang på data er begrenset. Vi skisserer flere veier for videre forskning som vi mener kan øke ytelsen betraktelig.

Ytelsen til modeller trent på datasett med flere ulike egenskaper blir evaluert og sammenlignet. Vi finner at størrelsesforhold på bildene, realistiske bakgrunner samt tildekking av objekter er viktige faktorer for ytelsen. Dette er til dels motstridende med funnene fra Peng et al. [PSAS14] hvor de fant at deres objektgjenkjenningssystem var i stor grad ikke avhengig av realistiske bakgrunner. Dette er sannsynligvis forårsaket av forskjeller mellom de to objektgjenkjenningssystemene som benyttes.

Det argumenteres for at syntetiske datasett kan være verdifulle for å trene gjenkjenningssystemer i nye kategorier hvor det per i dag ikke finnes treningsdata, i tillegg til å benyttes i kontrollerte eksperimenter for å utforske hvordan CNN-er responderer på ulike egenskaper ved treningsdata.

Acknowledgements

This thesis work has been conducted with guidance of Associate Professor Donn Morrison at the Department of Computer and Information Science (IDI). We thank him for guidance and support throughout the writing of our masters degree.

Additionally, we want to thank Anne Elster for providing us with equipment in her laboratory which has been invaluable in our research. Also, thanks to Geir Amund Svan Hasle for helping us with setting up and configuring computers in the lab.

Finally, we want to thank Sigve Sebastian Farstad, Margaret Davis Schoening and Aleksander Vognild Burkow for valuable input on our research and writing.

Contents

List of Acronyms

ANN Artificial Neural Network.

AP Average Precision.

CNN Convolutional Neural Network.

DCNN Deep Convolutional Neural Network.

DNN Deep Neural Network.

DPM Deformable Parts Model.

fps frames per second.

GPU Graphics Processing Unit.

HOG Histogram of Oriented Gradients.

ILSVRC ImageNet Large Scale Visual Recognition Challenge.

IOU Intersection-Over-Union.

KITTI KITTI Vision Benchmark Suite.

mAP mean Average Precision.

NN Neural Network.

PASCAL VOC PASCAL Visual Object Classes.

RPN Region Proposal Network.

SGD Stochastic Gradient Descent.

SIFT Scale-Invariant Feature Transform.

SURF Speeded Up Robust Features.

SVM Support Vector Machine.

YOLO You Only Look Once.

Chapter 1
Introduction

Computer vision is the science of analyzing and understanding the content of images automatically, using computers. There are multiple types of information that can be extracted from images, including what objects are visible, detecting events happening or finding the position of objects. Good vision systems can automate various manual tasks and remove the need for human labour, or open up entirely new possibilities by enabling applications that were previously too time consuming or data intensive to accomplish. Object detection systems are already in use today in pedestrian detection systems [SKCL13], face detection [VJ01] and in industrial processes[MK91].

As an example of a novel object detection application, we envision an embedded system that uses a camera to take pictures or video of its surroundings and, employing computer vision, identifies potential mushrooms and alerts the user. Such a system could be deployed using autonomous scouting drones which would map mushroom locations to reduce the time spent of commercial harvesters to find them. Chanterelles for instance, are still commercially harvested in the wild, and with global chanterelle commerce surpassing a billion dollars in the early 2000s [PNDM03], effectivisation of mushroom harvesting would be valuable.

However, there are numerous challenges involved in constructing such a system, including building an accurate mushroom detection machine learning model and making such a model fast enough to run in real time on an embedded system. These challenges map onto three ongoing research areas that are far from solved: accurate object detection models, embedded computer vision and real-time object detection.

In the case of accurate object detection models, advances in the machine learning field of Convolutional Neural Networks (CNNs) have improved the accuracy of such systems considerably in recent years [KSH12]. However, deep neural networks require a vast amount of labeled training data to produce good detectors. For many problem domains, including our mushroom detection proposal, no such dataset exists, and creating it can be prohibitively labour intensive.

When it comes to embedded platforms, computer vision techniques are constrained by limited computational power, which is especially problematic when it comes to using Deep Neural Networks (DNNs), since they are both computationally and memory intensive. There is a lot of promise in using smart phone platforms, as they come with powerful hardware, including Graphics Processing Units (GPUs), and a camera. However, those development platforms are still immature, and we are only now starting to see systems and libraries utilizing the GPUs on smartphones for such applications [OGK$^+$15].

Finally, real-time object detection has been possible for a long time, for certain objects, using methods such as the Viola Jones face detector [VJ01]. These techniques are heavily object specific and using them for a new object type is non-trivial. Recent advances in CNN-based object detection have, however, also showed promise of real-time behavior with state of the art performance. [RHGS15, RDGF15] Additionally, efforts have been made to let CNNs take advantage of increasingly powerful mobile GPU architectures.

Of the three challenges outlined, building an accurate object detection model with the lack of good training data pose the most significant challenge, and it will have to be solved in order to realize the envisioned mushroom detection system. Therefore, this thesis will focus on exploring alternative solutions for training CNNs without pre-existing datasets.

Various approaches have been used to reduce the amount of needed training data. It has been shown that using Neural Networks (NNs) trained on a big labeled dataset and then retrained for another purpose, reduces both the amount of data needed [GDDM13, EBC$^+$10], as well as the time to convergence. Another established technique is augmenting the dataset by adding multiple versions of the same pictures with minor variations, such as cropping, and brightness and coloring changes [WG15, RDGF15, SSP03]. However, even with these methods one needs thousands of labeled images.

While these means of data augmentation help increase the performance of networks trained on small datasets, and reduce chances of overfitting, significant work is still required to compose a dataset. We are therefore interested in alternatives that can replace the data collection process altogether. Previous research [RDGF15] have found that certain object detection systems perform well when trained on natural image data and evaluated on datasets of paintings, such as the Picasso Dataset [GHBM14]. This indicates that the abstractions learned by neural networks are powerful enough to cross between real imagery and less photo-realistic images. Further, Peng et al. [PSAS14] have investigated DNN detectors trained on images rendered from 3D models of objects.

Today, a large amount of 3D models are freely available online [1]. Additionally, multiple game engines have recently allowed free use for academic and non-commercial projects. Downloading and importing models in game engines and then rendering 3D scenes to a large set of images is now a computationally trivial task, and can be accomplished on a regular laptop computer. Several game engines support custom scripting, which means that meta information such as the position of rendered objects in 2D space can be exported along with the images.

Inspired by these findings, the goal of this thesis is to explore the viability of training DNN-based object detection models on synthetic data generated with modern game engines and to explore what factors are important to the quality of synthetic datasets.

We hypothesize that if such a technique works well, it should work for any object type. Since there are no good benchmarks for testing mushroom detectors, it was decided to build a system for detecting cars in pictures and use the KITTI [GLU12] computer vision competition data to measure the results. The rest of the thesis will focus solely on exploring the viability of such an approach by conducting experiments that investigate what is required of a synthetic dataset in order to train a network that generalizes to natural images.

1.1 Structure of the Thesis

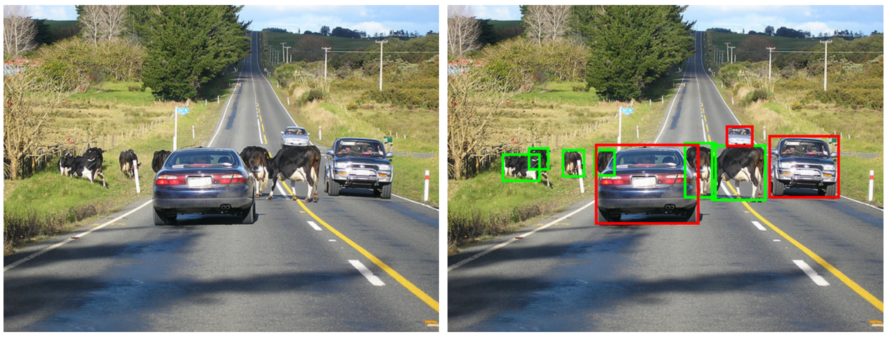
The thesis is structured as follows: The background chapter gives a brief introduction to the history of object detection and public datasets before delving into Artificial Neural Networks (ANNs) and how they are being used for object detection. It then goes on to explore and compare relevant object detection systems and the use of synthetic datasets. The methodology chapter goes into more detail on our method for dataset generation and lays out the experiments to be performed. The results chapter covers details about our setup and shows relevant results that the discussion chapter goes on to discuss. Finally the conclusion chapter summarizes the thesis and draws a conclusion followed by suggestions for further work.

[1] https://www.cgtrader.com/, http://www.turbosquid.com/, http://www.clara.io/

Chapter 2 Background

2.1 Object Detection

Object detection is a subfield of computer vision where the goal is to create a computer program that can identify and locate objects in an image. For instance, given the image in Figure 2.1a the goal would be to predict the bounding boxes around the objects as shown in Figure 2.1b.



(a) An example image

(b) Detected bounding boxes overlaid, with bounding boxes for each class in different colors. Cows are outlined in green, and cars in red.

Figure 2.1: Example of image with and without detected bounding boxes applied

Object detection is different from the similar challenge of *image classification* where the goal is to find labels that describe an image, which in the case of Figure 2.1a could be *cow*, *car* and *road*.

Machine learning algorithms have become an integral part of object detection research, as the problem is highly complex. Machine learning is the study and construction of algorithms that can learn from examples instead of being explicitly programmed. This is especially useful for high-dimensional and complex problems where it is much easier to get example data than to program a direct solution. Example applications include voice recognition, natural language processing, classification of DNA sequences, fraud detection on credit cards and computer vision.

Machine learning is often divided into supervised and unsupervised machine learning. Unsupervised learning focuses on finding structure in data where you get no extra information at all, an example is to cluster similar words given a lot of textual data. Supervised learning, on the other hand, is given a labeled training set with example pairs of input-output pairs. The goal is then to learn from this data and be able to predict the right output when given new data that was not part of the training data. For object detection, the training data would be pairs of images (input) and bounding boxes for objects (output), and after training on this data it should be able to predict bounding boxes on a previously unseen image.

2.2 Public Datasets

With the rise of the Internet, researchers started to compile large public datasets that any other researcher could use for their computer vision research. This has had a great impact for multiple reasons. Perhaps most importantly this has made it possible to compare performance for different models, as they can now be evaluated on the same data.

These datasets provide a large collection of varied images, depicting a defined set of object categories. Along with the images, they provide annotations for each image, denoting the position of these objects by listing the coordinates and sizes of the correct bounding boxes. These correct bounding boxes are often referred to as the *ground truth* bounding boxes. Because it is essential to test an algorithm on data it has not seen during training, the datasets are divided into three parts: training, validation and test data. Researchers train their models only on the training data, and then measure their performance, how good they are at predicting the right answer, on the validation data. If the performance is measured on data the model has already seen during training, the results cannot be trusted because of a common problem with machine learning known as overfitting. Overfitting is when a model learns something that was specific to data it was shown, but which does not generalize to other data. An extreme example would be if it had only seen red cars on a road during training. It then might label anything red and call it a car as that would work well for the data it was trained on. In this case it would achieve very good results on the training data, but if it was tested on a separate validation set, this overfitting would not be helpful any longer, and it might miss a gray car or label a red house as a car. The reason for another separate *test* dataset is to prevent a more subtle form of overfitting problem. Every time a researcher makes a choice of a hyperparameter – a parameter that affects performance but is not found automatically – based on what value gives the best result for the validation set, she is fitting the parameters to the validation set and might start overfitting after a while. In the case of the public dataset competitions this test dataset is used to measure the final scores and kept secret from the researchers.

2.3 Measuring Performance of Object Detection

To compare how well two different object detection systems perform on an evaluation dataset, a score for the accuracy of each of them is useful. The measure used is normally Average Precision (AP), a metric derived from two measures of false positives and false negatives: precision and recall.

Recall denotes the amount of ground truth bounding boxes that were accurately predicted:

$$\frac{\text{True positives}}{\text{Number of ground truth boxes}} \tag{2.1}$$

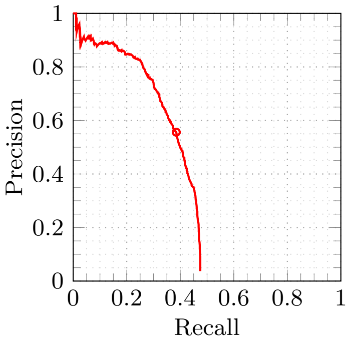
Precision denotes the number of predictions that were correct:

$$\frac{\text{True positives}}{\text{True positives} + \text{False positives}} \tag{2.2}$$

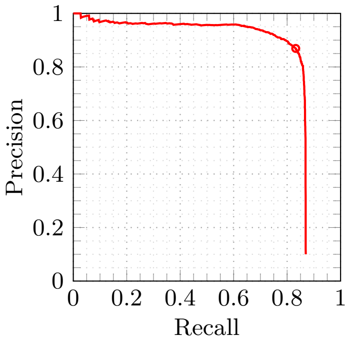
A true positive is found when there is a significant overlap of a prediction made by the detector and a ground truth bounding box. This is defined by the Intersection-Over-Union (IOU) of a prediction and a ground truth box, the intersection of the two boxes divided by the union of their areas. A typical threshold value signifying a correct detection is an IOU of 50% [EEG$^+$14].

Predictions made by the model are not absolute – each prediction has its own confidence score denoting how certain the prediction is. By choosing a minimum confidence value, the number of predictions will be reduced. Thus, the precision is increased since the number of false detections decreases. However, higher threshold values for confidence also mean that recall decreases. Selecting a good threshold value is therefore a matter of finding a good balance between high recall and good precision.

The AP measure handles the different threshold values by computing precision and recall values for a high number of different thresholds, and averaging the precision score for each recall value. This can also be seen as the area under a precision-recall curve. See Figure 2.2 for an example of two precision-recall curves, the 2.2a shows a curve with an AP of less than 35% while 2.2b gives a curve with an AP of over 82%. Notice the difference in area under the curve.



(a) A poor precision-recall curve with an AP of 34.96%



(b) A good precision-recall curve with an AP of 82.42%

Figure 2.2: Average Precision is defined as the area under the precision-recall curve. The curve of a good model should be as close to the top-right corner as possible.

2.4 History of Object Detection Systems

Early object detection systems searched the image by comparing values with a picture of the object they were looking for using some local feature descriptor like Sum of Squared Differences (SSD), Scale-Invariant Feature Transform (SIFT) [Low99] or Speeded Up Robust Features (SURF) [BTG06]. These feature descriptors were then combined with machine learning techniques. An example is the 2001 Viola and Jones system where they used a simple feature descriptor combined with a supervised machine learning technique to achieve fast and accurate face detection [VJ01]. This technique worked so well it is still in use, but it does not generalize well and is highly specific to front-facing faces. In 2005 Josef Sivic et al. used unsupervised learning to cluster SIFT features into more powerful feature [SRE$^+$05] inspired by the Bag of Words machine learning technique. In the same year Dalal and Triggs published a paper introducing a simple feature descriptor they called Histogram of Oriented Gradients (HOG) and combined with a supervised learning algorithm known as SVM [CV95] they outperformed previous algorithms for pedestrian detection [DT05]. Using the HOG feature Felzenszwalb et al. published a paper in 2008 introducing a technique known as Deformable Parts Model (DPM) where they model an object as a combination of multiple parts detectable with HOG.

In the early 2010s a machine learning technique known as Deep Neural Networks (DNNs) – a type of large Artificial Neural Networks (ANNs) – started outperforming other techniques in multiple fields, such as speech recognition [GMH13] and in 2012 DNN-based methods caught the attention of the computer vision community, when Krizhevsky et al. placed first in the ImageNet image classification challenge. With a top-5 error rate[1] of 15.3% [KSH12], compared to 26.2% achieved by the second-best entry for that same year, they showed that DNNs could outperform other methods by a significant amount. The year after, Convolutional Neural Networks (CNNs) were applied to object detection by Girshick et al. [GDDM13], achieving a mean Average Precision (mAP) of 53.3% on PASCAL Visual Object Classes (PASCAL VOC)2012, an improvement of over 30% relative to the previous best result of 35.1% mAP reported by Uijlings et al. in their 2013 paper [UvGS13]. As of June 2016 the top object detection results for PASCAL VOC, ImageNet and KITTI Vision Benchmark Suite (KITTI) are based on DNNs.

As the object detection system utilized in this thesis, You Only Look Once (YOLO), is a DNN the next section will give an overview of how they work as well as introduce object detection systems based on DNNs including YOLO.

Artificial Neural Networks (ANNs)

Based on the early Neocognitron model by Fukushima [FM82] neural networks gained popularity in the early 90s after three groups independently discovered a very effective technique for training them known as back propagation in 1985 and 1986 [Par85, Cun85, RUM86]. (Another group discovered a similar technique in 1974 [Wer74], without catching the attention of the machine learning community.) However, while it remained the state-of-the-art for reading handwritten digits [CBD$^+$90], focus shifted to other machine learning algorithms in the late 90s after the Neural Networks (NNs) failed to achieve state-of-the-art performance.

The recent resurgence of neural networks is driven by the good result of big networks that requires today's fast Graphics Processing Units (GPUs) [RMN09] and access to very large datasets, none of which were available in the 90s.

In addition to the extra data and computation, there has been a couple of key innovations that enabled even deeper networks, some of which will be covered in this section. Using *rectified linear units* as the nonlinearity instead of previously common ones like *tanh* or the *sigmoid* function made networks converge quicker, *autoencoders* allowed for unlabeled pre-training that lets you get away with less labeled data [HOT06, H06, BLP$^+$07, PCCo06], and dropout (section 2.4.1) helped reduce overfitting of big networks [HSK$^+$12, SHK$^+$14].

[1] In the ImageNet image classification challenge each image has a correct label. The competing algorithms are given five guesses per image and the *top-5 error rate* is the percentage of images where five guesses did not include the correct answer. 0% is a perfect score.

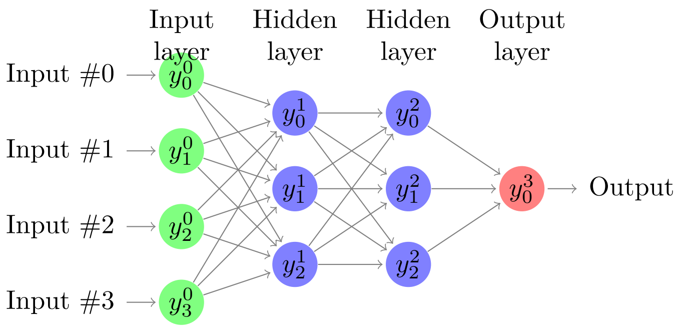


Figure 2.3: Graphical representation of an example neural network. Each circle corresponds to a node with value y_i^j. Each arrow represents a directional connection between two nodes.

2.4.1 The Structure of Artificial Neural Networks

An artificial neural network contains multiple layers of “neurons”, or nodes, where each node has connections to the nodes from the previous layer as seen Figure 2.3. The first layer is referred to as the input layer since the nodes in this layer is set to the input values. Conversely, the last layer is referred to as the output layer, since this layer will contain the output of the neural network after evaluation. Any layers in between are called hidden layers, as they are not directly observable when you run an evaluation, but calculate intermediate results. Note that the number of hidden layers as well as number of nodes in every layer, including the output layer, varies. When a neural network has multiple hidden layers it is often referred to as a deep neural network or DNN.

Each connection between nodes has an associated weight. Evaluation of the network is done by setting the values of the nodes in the input layer to the input data, and the values of the nodes in the next layer are then calculated according to the following equation

$$y_i^j = f(\sum_{i=0}^{N^{j-1}} w_i * y_i^{j-1}) \tag{2.3}$$

where y_i^j is the value, often referred to as the activation energy, of the ith node in layer j, f is some non-linear function, N^j is the number of nodes in layer j, w_i is the weight of the connection between the current node and the ith node in the previous layer that has the activation energy y_i^{j-1}.

This is done for every layer in sequence until the output layer is reached, the output of the network is the activation energies of this final layer.

To make classification predictions with a neural network one typically normalizes the values of the last layer, using a function known as the softmax function, such that the values of all the final nodes sum to one. Each node of the output layer is then interpreted as the prediction probability, or confidence, of the input being of a specific class. For instance, if the network is used to distinguish between cats, dogs, cars and mushrooms, the output of the final layer $[0.7, 0.3, 0, 0]$ might indicate a 70% confidence that the input picture was a cat, 30% that it was a dog and 0% car or mushroom.

There are different connection patterns possible between the different layers. In Figure 2.3 and equation 2.3 layers are fully connected, that is, every node is connected to every node in the previous layer, but this is not always the case. A closer look at an important connection pattern for NNs that work well on images, known as Convolutional Neural Networks (CNNs), is given in subsection 2.4.1.

Training Neural Networks; Backpropagation

The connection weights change the complex function calculated by the network as a whole and are the parameters being optimized during training. This is where the backpropagation algorithm [Wer74, Par85, Cun85, RUM86] mentioned in the introduction applies.

A cost function is defined that is to be minimized with regards to the weights over a training set of examples (x,y). This is typically a measure of how wrong a prediction is and is defined in terms of a prediction and the correct answer, which is available during supervised learning. A number of minimization algorithms can be used, a popular and effective algorithm is Stochastic Gradient Descent (SGD) [LBOM12], where you shift the parameters a tiny amount in the opposite direction of the gradient calculated over a random subset of the training data. Drawing a new random subset of the training data for each iteration is the stochastic part of the algorithm. While there are many optimization algorithms that are more advanced, SGD has shown best performance for large scale networks [BB08].

Using backpropagation to train a network, you can proceed in the following manner with a training example (x, y):

1. Set the input layer to x and do a forward pass. This gives an estimated result at the output layer y'.

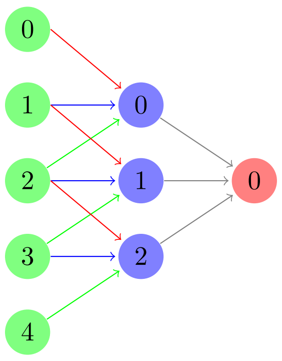


Figure 2.4: Selected units from two convolutional layers. Unit 1 in the second layer has unit 1, 2 and 3 in the input layer as its receptive field. Note how changes to node 0 and 4 do not affect this node at all. The node in the last layer has a receptive field of all three nodes in the second layer and all 5 input nodes. This is convolution in one dimension, in two dimensions, the receptive fields would be rectangles. The matching colors correspond to shared weights.

2. Calculate the gradient of the cost function of y' and y with respect to the network weights by propagating the errors backwards through the network.
3. Use the gradient to perform a step in your chosen minimization algorithm.
4. Repeat.

Convolutional Neural Networks (CNNs)

From Hubel and Wiesels work on monkey brains in 1968, we know that their visual cortex contains cells that are responsive only to a subset of the cells in the retina, and these cells are spatially close [HW68]. Cells that affect a cell in the next level of the hierarchy are called the cell's responsive field. This organization makes sense, as visual imagery is highly local, a mushroom in the center of your vision is still a mushroom even if a lot changes in your peripheral vision.

Inspired by this, Lecun et al. introduced in 1990 a technique known as convolutional neural networks, or CNNs. Their paper applied neural networks to handwritten digit recognition [CBD$^+$90], where they used the raw pixel values of a 16x16 pixel image as direct input to the network.

A CNN is a neural network that contains one or more layers with a convolutional architecture, as can be seen in Figure 2.4. In a convolutional layer each node is connected only to a spatially confined area of the layer before it, its receptive field as it were. As CNNs is an important technique for applying neural networks directly on pixel values in a 2D image, the receptive field is usually a NxN square. All nodes share the size of the receptive field, but for each node the center is shifted by a given offset. The offset and the size of the receptive field varies between architectures and affects how much the receptive fields overlap. The great innovation from [CBD$^+$90] was to group a set of nodes that cover the whole input with their receptive fields into *feature maps* that share the exact same weights, as illustrated by the colors in Figure 2.4, something that greatly reduces the amount of parameters in the model and makes training large convolutional layers feasible. A convolutional layer usually consists of multiple feature maps that each span the input with their receptive fields.

The name convolutional stems from the mathematical convolution operation where you shift a window of weights over an image. The weights are multiplied with the underlying values and summed together to produce a value in the new output image. This is very close to what the convolutional layers are doing. It is easy to see that the first few layers are performing operations, such as edge detection, which is commonly achieved using convolution in image processing.

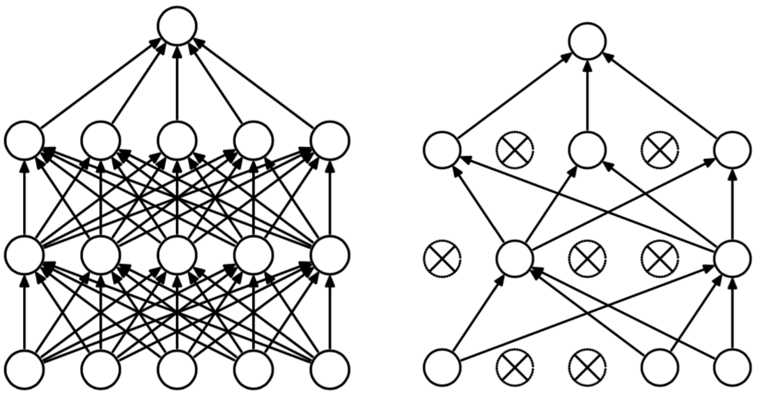
Subsampling with Max-Pooling

Convolutional layers are often followed by a subsampling layer, where the most common technique is a *max pooling* layer. The goal of subsampling is twofold. Firstly it reduces the size of any subsequent layer, reducing the complexity of the model and thus allowing substantially deeper networks. Secondly it provides some location invariance if the subsampling is performed in the right way. The most common subsampling method in modern NNs is known as max-pooling and works as follows. A node in this layer is connected to a number of nodes in the previous layer, say a 2x2 "receptive field", and gets its value by choosing the maximum value of those nodes.

While pooling layers give locational invariance which increases accuracy when detecting objects, they are also inherently losing potentially valuable location information that could be used for localization of the detected object. Some methods have been proposed to use location information directly from a deep convolutional neural network, but supplement it with a network that learns the offset [TGJ$^+$14].

Reducing Overfitting of Big Models Using Dropout

Overfitting occurs when a machine learning technique learns to distinguish classes based on random noise in the training set that happens to be useful for the classification, but which does not generalize outside the training data. When this happens, the algorithm will perform very well on the test data, but will lose a lot of accuracy on previously unseen data. The risk of overfitting increases with the ratio of



(a) A fully connected neural network. (b) The same network after applying dropout. The dropout is applied randomly for each iteration of training, but not for evaluation.

Figure 2.5: A figure from [SHK+14] illustrating the dropout technique.

$\frac{\text{number of parameters}}{\text{amount of training data}}$. Thus, because deep neural networks have a vast number of parameters, often in the millions or more, overfitting is a serious concern.

An effective technique to reduce overfitting in big networks known as *dropout* was first introduced by Hinton and Srivastava et al. in 2012 [HSK+12] and later improved in 2014 [SHK+14].

The dropout technique works by dropping random nodes and all their connections during training. See Figure 2.5 for an illustration. This makes specific features from the training set unreliable to the network as those units may be dropped, forcing it to generalize better. In their 2014 paper [SHK+14] Srivastava and Hinton et al. show significant improvement for a variety of applications, including handwritten digit recognition from the MNIST dataset [LBBH98], image classification on ImageNet and clean speech recognition on the TIMIT dataset [GLF+93]. They also showed that it takes roughly 2-3 times as long to train compared to the same network without dropout as each iteration has to calculate the gradient for a different random architecture [SHK+14].

Using Pre-trained Models

When training a deep neural network on different vision tasks with raw pixel data as input, a lot of similar functionality will be found in the first few layers, even though one network might be trained to distinguish between a thousand categories of images including dogs and cats [SLJ+14, HZRS15, ESTA13] while the other one can recreate the artistic style of an artist on arbitrary images [GEB15].

Based on this intuition, it would seem wasteful to train new networks completely from scratch each time. And experiments show that you can often save a lot of training time as well as get away with much smaller training sets by setting the weights of the first layers to the weights of a pre-trained model instead of using random initial weights before training [GDDM13]. A lot of the object detection papers covered in section 2.4.2, such as YOLO [RDGF15] and Faster R-CNN [RHGS15] use models pre-trained on the ImageNet dataset [DDS+09].

Another technique, known as autoencoding, to get good initialization weights in the network before training by doing unsupervised pre-training was discovered by Hinton and Salakhutdinov in their 2006 paper [H06]. It works by training a network to produce the same output as it gets as input. This sounds trivial, but if any layer is smaller than the size of the input, the network is forced to learn a compression, which means it has to discover the structure in the images to be able to represent it with less data. This compression is a good starting point for a CNN to be trained on similar images.

2.4.2 Object Detection Networks

This section introduces a few different CNN-based object detection models leading up to the YOLO system that we have chosen for this thesis.

R-CNN

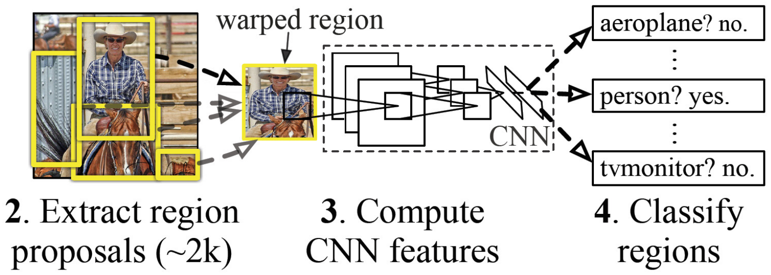


Figure 2.6: Illustration of the R-CNN architecture from the paper [GDDM13]. It illustrates how an input image is split into many regions, each region being warped into the correct input size and fed through the CNN that produces a feature vector which is fed to a number of linear Support Vector Machines (SVMs) in step 4.

In 2013, inspired by the work of Krizhevsky et al. on the CNN-based classifier that outperformed earlier classification techniques [KSH12], Girshick et al. proposed an object detection system that combined a similar CNN with region proposals [GDDM13]. In the period between 2010 and 2012, there had been little progress in object detection performance, and the introduction of Deep Convolutional Neural Networks (DCNNs) started a new era in object detection research. While the then best-performing systems had been based on more low-level features like SIFT [Low99] and HOG [DT05] combined with machine learning techniques such as SVMs, the authors showed that a detection algorithm based on a neural network could outperform existing systems. Measured in mAP, R-CNN saw a 30% increase over the previous best result in VOC 2012 [EEG+14].

The R-CNN object detection system works in three steps. An overview of the steps can be seen in Figure 2.6. The first step generates around 2000 category-independent region proposals. In R-CNN, Selective Search [UvGS13] is used for this step, but other region proposal algorithms may also be used interchangeably. The region proposals are resized and warped into a 227x227 pixel size representation, which are used as input to a CNN. The CNN then does forward propagation through five convolutional layers and two fully connected layers, and outputs a 4096-dimensional feature vector. The final step is a set of class-specific linear SVMs, that scores each feature vectors for a specific class. Based on all the scored regions for all the classes, only the highest-scoring classes for each overlapping region are kept.

What makes the algorithm efficient is the fact that the parameters in the CNN are shared across all categories. This has the benefit that the first step is executed only once for all classes, and the evaluation of the CNN is only done once per region from the first step, significantly reducing the computational cost. Only the last step, computing the dot product of features and SVM weights is done per-class. The dimensionality of these feature vectors is also relatively low compared to other approaches, at 4000 [UvGS13]. This lowers the memory demand and computation required for the dot product operation in the last step.

R-CNN still has some drawbacks. Most importantly, the training is slow, and can take up to 2.5 GPU-days[2] when training a deep network on 5000 images as described in the paper. This is caused by the complexity of the training pipeline, consisting of fine-tuning a convolutional neural network, fitting SVMs to the features of the network, and learning bounding-box regressors. Detection is also slow, taking 47 seconds per image on a desktop GPU [Gir15]. One of the major bottlenecks of the R-CNN system on evaluation time is the fact that it needs to run one full CNN forward propagation for each region proposal, which is usually in the thousands for any given image.

[2]A GPU-day is 24 hours on one GPU. For instance if a computation takes 12 hours on 4 GPUs, then that is 2 GPU-days.

Fast and Faster R-CNN

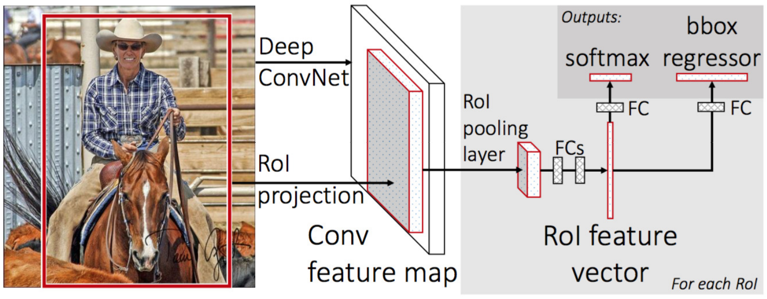


Figure 2.7: The Fast R-CNN architecture as illustrated in the paper [Gir15]. *ConvNet* is another term for a CNN, *RoI* stands for *regions of interest* and FC for a *fully connected* layer.

Fast R-CNN was proposed in 2015 by Ross Girshick [Gir15] to improve the training and testing speed over R-CNN, while also improving the detection performance in mAP from 62% to 66% on VOC 2012 [EEG+14]. The training speed is increased by a factor of 9, and the testing speed is up to 213 times faster than the original R-CNN implementation.

As shown in Figure 2.7 the Fast R-CNN network takes an entire image as input along with a set of regions of interest. Several convolutional and max pooling layers first generate a convolutional feature map. Next, a region of interest layer extracts one feature vector from this map for each object proposal. These feature vectors are finally passed through a series of fully connected layers that end in two output layers: a softmax layer producing a probability estimate for each object class, and a bounding box regressor returning refined bounding-box positions for each class.

Fast R-CNN shows significant speed-ups in detection time, but still relies on slow algorithms like selective search for generating initial object proposals, which limits its overall detection speed. Selective search [UvGS13] typically requires around 1-2 seconds per image [RHGS15], limiting the frame rate to under 0.5fps and overshadowing the benefits of speeding up the detection network further (e.g. using smaller models).

Faster R-CNN, released shortly after Fast R-CNN [Gir15], presents a new technique that does away with the largest bottleneck remaining in Fast R-CNN, namely the region proposal computation [RHGS15], while increasing the mAP from 66% to 70.4% on VOC 2012 [EEG+14]. They introduce a Region Proposal Network (RPN) which enables almost cost-free region proposals by having the RPN share convolutional features with the detection framework.

The idea for the RPN is based on the intuition that since object proposals are not needed by Fast R-CNN before after the first convolutional layers, and the feature maps outputted after these five layers can be used to generate region proposals, a lot of computation may be saved by integrating the region proposals in the same network. The RPN is constructed by adding two new convolutional layers on top of the shared convolutional layers of Fast R-CNN.

Since most of the computation in the RPN is shared with the detection network, only the cost of the two extra layers have an effect on the speed of the whole detection pipeline. The effective running time for the RPN is therefore only 10 milliseconds.

The complete pipeline achieves 5 fps even on the very deep VGG16 [SZ14]. Using the smaller ZFnet [ZF13], the frame rate increases to 17 fps, trading for a slight reduction in mAP. All the timings are measured on a K40 GPU [RHGS15].

You Only Look Once (YOLO)

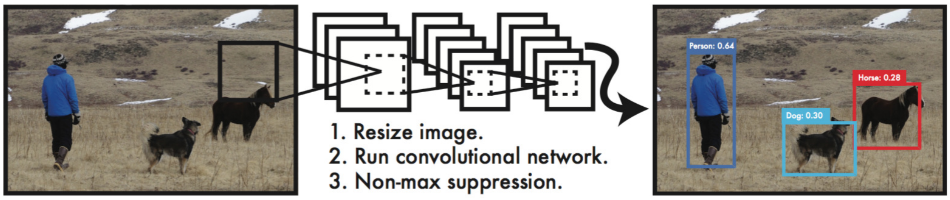


Figure 2.8: The YOLO system as presented in [RDGF15] is simple seen from the outside since all the processing is accomplished by a single deep CNN.

YOLO, published in 2015 by Redmon et al. [RDGF15], takes a different approach than the classifier-based systems we have looked at so far. Its predictions are based on the entire image and made using only a single network evaluation, as shown in Figure 2.8. This means that the NN in YOLO can be trained end-to-end directly for detection. The reported detection frame rate is 45 frames per second, which is more than 100x of Fast R-CNN and 3x of the fastest Faster R-CNN model. A smaller version of the YOLO network even reports a detection speed of 155 frames per second (fps).

YOLO works by dividing the input image into a $S \times S$ grid (Figure 2.9). Each grid cell detects objects that fall within its bounds. The grid cells each predict B bounding boxes with corresponding confidence scores, which reflect both the confidence that the prediction contains an object and how accurate the location of the box is.

The standard YOLO network architecture is very deep, with 24 convolutional layers and 2 fully connected layers. Fast YOLO uses a smaller network with only

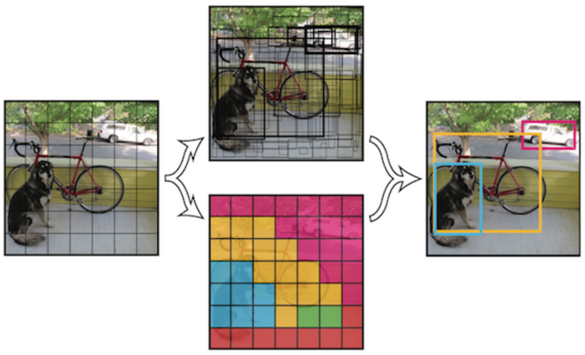


Figure 2.9: YOLO processes the image subdivided in a 7x7 grid structure. It predicts two bounding boxes for each cell. As a post-processing step, non-maximal suppression removes overlapping bounding boxes for the same class.

9 convolutional layers, and is thereby trading off prediction accuracy for higher speed and lower memory consumption. Both versions have a final output tensor with dimensionality $S \times S \times 30$ for 20 classes (S is set to 7 in the described implementation).

When images are processed by the NN, they are first scaled to a fixed resolution of 448 X 448 pixels. This means that images of differing aspect ratios will be stretched in different ways. Therefore, it is important to consider the format of the images where detection will be performed when constructing training data for YOLO.

The architecture of YOLO has some advantages, in addition to being very fast. Since predictions are based on the entire image, YOLO is less likely to predict background as objects, because YOLO sees the larger context of the image than detectors which only classify regions. The grid architecture also has some drawbacks, however, as it causes less precise localizations than region-based classifiers. In addition, a limitation of only two predictions per grid cell means it can easily miss small objects in cases where multiple objects are close together.

Another good property of YOLO that is especially interesting to us, is the way the model generalizes to more abstract representations of objects. The paper describes an experiment where the network is trained only on a subset of the VOC data [EEG+14] (natural images) and then tested on the Picasso Dataset [GHBM14] for the `person` class (paintings, see Figure 2.10). YOLO shows only a small reduction in AP from VOC 2007 to the Picasso Dataset, from 59.2% to 53.3%. The paper attributes this to the fact that YOLO models the size and shape of objects rather than low-level cues like texture and lighting, since it looks at the whole image during detection. Other object detection frameworks see a more dramatic reduction in performance, such as R-CNN. R-CNN depends on selective search for bounding box proposals,

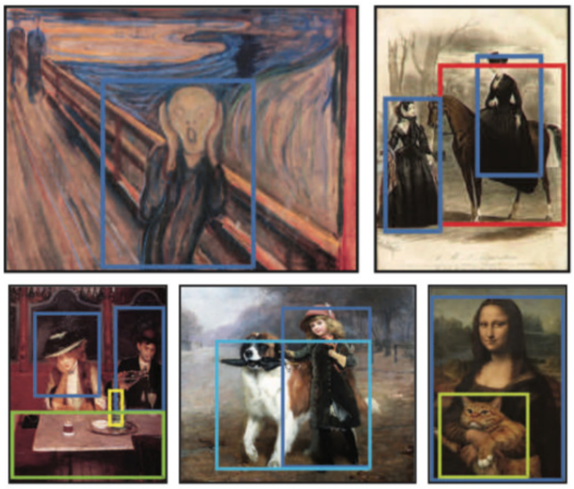


Figure 2.10: YOLO detections on the Picasso Dataset [GHBM14, RDGF15]. This shows that YOLO generalizes well to more abstract representations of the objects it has been trained to detect.

which is highly tuned for natural images, and this may explain why R-CNN and improvements based on R-CNN fail to generalize well.

2.4.3 Other Networks

Some notable systems not covered in detail here are the 2013 Overfeat network by Sermanet et al. [SEZ$^+$13] that is similar to R-CNN covered in section 2.4.2 but instead of Selective Search for object proposals it uses a sliding-window approach, leading to a much faster system with a significantly lower accuracy of 24.3% mAP on PASCAL VOC.

The 2014 SPPNet by He et al. [HZRS14] that can be viewed as an improvement of R-CNN by sharing computation between object proposals in a way that reduces training time by a factor of 3 and speeds up testing by an order of magnitude. Even so it is beat by later networks such as Faster R-CNN.

Single Shot Multibox Detector (SSD) by Liu et al. [LAE$^+$15] published in December 2015 is similar to YOLO in that it is done with a single CNN without a separate region proposal step, and achieves somewhat better accuracies than YOLO at comparable speed, but still far lower accuracy than the slower networks Fast R-CNN and Faster R-CNN.

2.5 3D Models and Synthetic Datasets

One of the big drawbacks of CNNs is that training them is expensive and a very big set of training examples is needed. For reference ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014 offers over 450 thousand labeled images for training over a thousand categories, and in this paper we are using 5000 images per training set. Producing such datasets with real photographs and manually drawn bounding boxes is very costly. As an example, the labeling effort of MIT Street Scenes, a dataset consisting of 3547 images, was done with a custom built tool and took an average of 3 minutes per picture, totaling 177 man-hours for the labeling alone. Taking the pictures was done by 13 people over 18 months, but it is unclear how much time was spent in total [Bil06].

Data augmentation, where all the pictures in the dataset are copied and altered with image processing, is related and can help increase the size of a dataset that already exists. Common changes to the pictures include cropping, stretching, adding noise and changing colors. This is easy to implement, has shown to increase performance and has become a common method when training CNNs. Chatfield et al. showed this technique to consistently give a mAP increase of 3% across multiple CNN architectures on PASCAL VOC [CSVZ14], and it has become common practice for most object detection systems [Sor14, KSH12, SZ14, SEZ$^+$13, RDGF15]. The Darknet framework that is employed in this thesis applies data augmentation automatically while training, so even our purely synthetic datasets are increased in size by augmentation techniques. Data augmentation works well to increase the performance of an already existing dataset, and might reduce the required amount of data, but it is still far away from solving the problem of needing thousands of images.

This thesis attempts to solve this problem by investigating the usage of purely synthetic data for training. This idea is not new, and this section will give an overview of how it has been applied in the past.

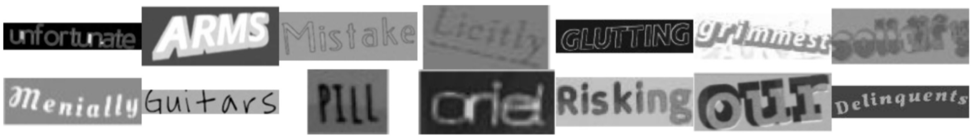


Figure 2.11: Randomly sampled data created by the synthetic text engine of Jaderberg et al. [JSVZ14]

As an example of other computer vision areas where this has been successful, Jaderberg et al. used synthetic datasets to greatly improve upon previous state-of-the-art natural scene text recognition in their 2014 paper [JSVZ14]. Natural scene text recognition is a much harder challenge than normal OCR which is aimed at reading black text on a white background. Text occurring in natural scenes has a large amount of variations in color, fonts, shadows, perspective and surroundings. Pre-existing datasets for this problem were made by cropping text from pictures of real scenes, and one of the more popular ones is made from text visible from Google Streetview [WBB11]. Jaderberg et al. generated images by writing a word in one of over 1400 fonts, adding random shadows and borders, changing colors, applying projective distortion and blending with a crop from a random natural image from a training set. This generated a virtually infinite amount of very varied and natural-looking data as seen in Figure 2.11. The sampling of random natural images from big public datasets to use as backgrounds is similar to our approach, but since we are generating cars and not text our methods for generating the foreground is very different. The significant increase upon previous results with this technique shows that even though there are datasets available for natural scene detection, the previous models were limited by lack of data.

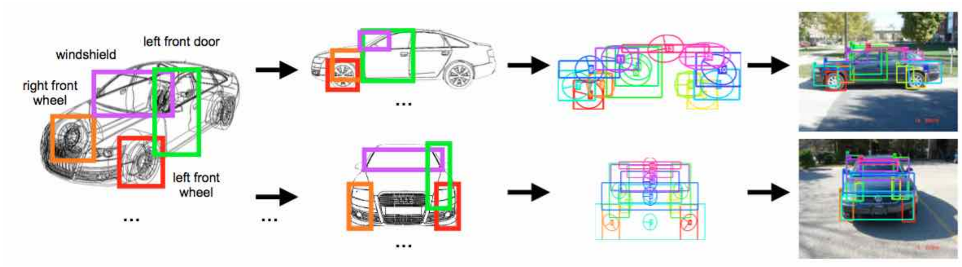


Figure 2.12: The use of 3d models to model the 3d relationship between different parts of a car for viewpoint estimation by Stark et al. [SGS10]

Much of the recent research on the use of 3D models for computer vision has been focused on pose and viewpoint estimation. Stark et al. [SGS10] used 3D models and statistics to get a representation of the 3D models that they matched to 2D images to do viewpoint estimation of cars. Using this probabilistic spatial model, as illustrated in Figure 2.12, they demonstrated superior performance to previous records on the *cars* dataset introduced by Savarese and Fei-Fei [SFF07]. This is fundamentally different from our approach, as the 3D models are directly used for their 3D data, while we only use them to render semi-realistic 2D images; our generated training data is entirely 2D.

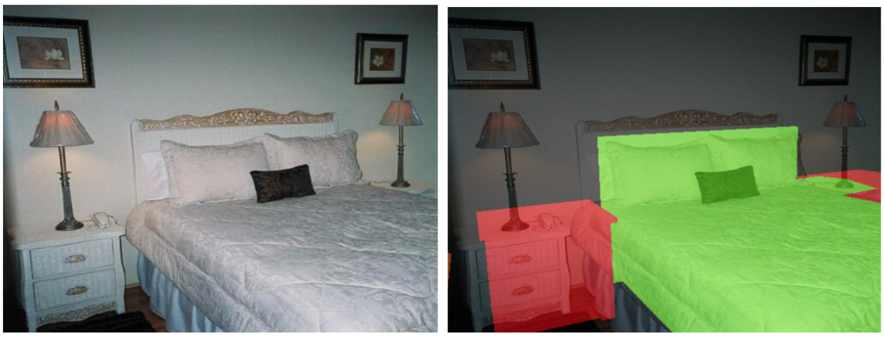


Figure 2.13: Example of the scene understanding problem solved by Satkin et al. On the right the best matching 3D objects are overlaid and color coded for type. Pictures taken from their paper [SLH12]

Satkin et al. [SLH12] used 3D models for a novel approach to scene understanding as shown in Figure 2.13. By calculating the intrinsic camera parameters[3] of the picture in question, they can render a large number of 3D objects with the right camera parameters to make them appear similar to what they would have looked like if they were in the scene. Using these rendered pictures, they compute various similarity scores that are used to pick the most similar 3D objects to the objects in the scene, and in this way recreating the scene with 3D geometry knowledge. One of the enabling factors of this technique is the utilization of the vast library of freely available 3D object models, in the same way that is important to our approach. However, while our approach was based on a few hand picked object models, they created a system that automatically downloads models based on keywords and filters out models that has a geometry that differs too much. For instance, if you search for *bed* you might get an architectural model of a *house with two beds*, but that geometry would differ widely from the geometry of an actual bed. Using this technique, they filtered 8000 models down to 2000 models of more relevance. A clustering algorithm was run on the 2000 models to discover synonym search terms that map to highly similar 3D geometry, in this way the tags *sofa* and *couch* could be applied to models found by the query *love seat*[4]. This systematic approach to gathering 3D models is something that could be applied to our approach in the future to generate large synthetic training sets across a large number of object classes, both to increase the number of classes available and increase the intraclass variability of the datasets. This is the inspiration for one of our suggestions for further work, section 6.1.

Liebelt and Schmid [LS10] and Pepik et al. [PSGS12] both embed 3D models

[3]The intrinsic parameters of a camera are important parameters of the camera itself such as focal length and image sensor format

[4]A couch with two seats

into a DPM-like detection system. While Liebelt and Schmid utilize the extra 3D information to increase their pose estimation accuracy, Pepik et al. simply shows that they can embed this 3D understanding without sacrificing detection performance, in an attempt to make it easier to extend to tasks that require more 3D reasoning, such as scene understanding and pose estimation.

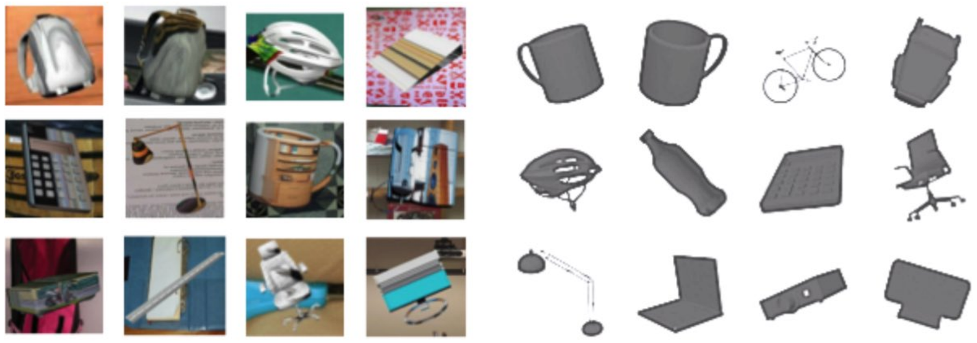


Figure 2.14: The two datasets generated by Sun and Saenko. Image taken from their paper [SS14].

All the previously covered systems using 3D models have all done so in order to explicitly utilize the knowledge of 3D geometry embedded in 3D models, which is inherently different to our approach where the 3D models is a tool to create representative 2D images with the objects of interest through modern 3D rendering techniques. Closer to our approach is the work of Sun and Saenko [SS14], who used free 3D models to render images for a training dataset of 2D pictures with automatic labeling for an object detection system. They downloaded models from the Google 3D Warehouse [5] where they selected two models for each of the 20 object categories in the Office dataset [SKFD10]. Their datasets feature a single object in the center of the image from various viewpoints, as seen in Figure 2.14. Surprisingly, and in contrast to our results, they found that using realistic backgrounds and textures did not increase performance. An important difference from our study is that they used a different object detection system; instead of a CNN, they used a DPM model with additional cross domain adaptation techniques. It is natural to believe that the invariance to cues such as background texture will vary between object detection systems, some systems might rely more on the surrounding textures, and HOG in particular uses gradients of the image which map to edges, something that is preserved even with highly unrealistic textures.

Peng et al. [PSAS14] also used free 3D models from the Internet of the 20 object classes from PASCAL VOC to generate purely synthetic dataset and evaluate on

[5]https://3dwarehouse.sketchup.com

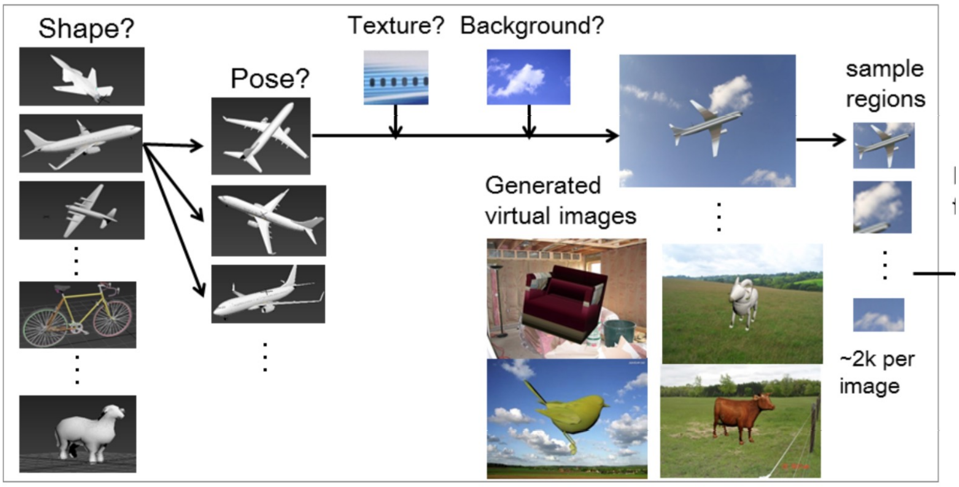


Figure 2.15: An illustration of Peng et al. synthetic dataset generation pipeline. Illustration from their paper [PSAS14]

the PASCAL VOC2007 dataset. Similar to our approach, they place 3D models in a scene with varied texture, background and pose. Their paper investigates both the feasibility and performance of such a technique and use it to test the invariances of the R-CNN object detection model cover in subsection 2.4.2. Somewhat surprisingly they find the model to be largely invariant to the background, and their datasets with a uniform color as background perform just as good, and sometimes even better than the ones with photos as a background.

Figure 2.15 shows their dataset generation pipeline. Each training image contained only one object, placed in the center of the image, with some variation in pose along one or more predefined axes. They generated datasets with and without relevant background images behind the objects. In the images that did have a texture, this did not involve photo-realistic textures that rendered the model as a real-looking car, but simply stretched a 2D car image across the texture, only giving the model somewhat more realism than it would have had without textures.

Their dataset generation technique is similar to the one we have used, but we include multiple objects, do not center them in the image, and use more realistic textures. In addition, while the object detection system employed in this thesis, YOLO, is faster, it also scores lower on accuracy than R-CNN when trained on real images, which would make it likely that our reported mAPs might be lower.

A few weeks before this thesis was submitted, Xerox Research published a paper by Gaidon et al. [GWCV16] that modeled some of the scenes from the KITTI

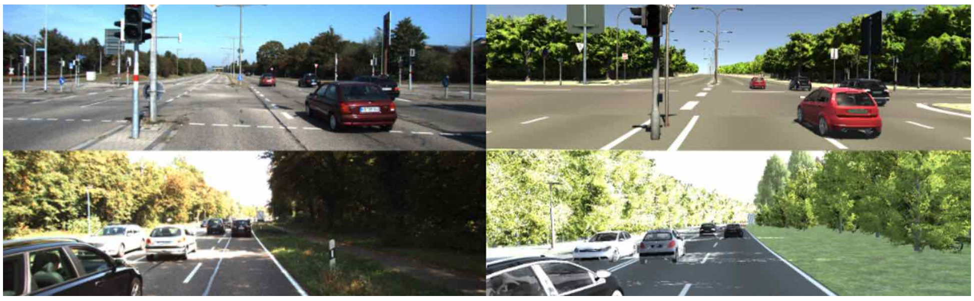


Figure 2.16: Examples from the recreated scenes from Gaidon et al. [GWCV16]. Images on the left are from KITTI while images on the right are rendered in Unity. Photo from the original paper.

benchmark in high detail using the same game engine as we used, Unity. They created virtual replicas of a few of the videos in the KITTI benchmark by utilizing the labeled training data to automatically place and move all the cars. As seen in Figure 2.16, the rendered scenes are very similar to the actual training data. In these virtual worlds they simulate different conditions, such as different camera angles or weather, and test models trained on real data on them to measure how invariant the models are to these factors. This novel use of generated data shows how synthetic datasets can be very useful in building a deeper understanding of complex CNN models by performing controlled experiments. The datasets also show the level of photorealism you can achieve with rendered data using free models and software. Generating datasets of this high fidelity is a lot more expensive than the datasets we have generated, as they have to model realistic surroundings which will vary depending on the object class. We have trained our models on their publicly available dataset to compare with the performance of our models.

Chapter 3
Methodology

This chapter starts by giving an overview of how data generation was approached using the Unity game engine, followed by our setup of the You Only Look Once (YOLO) Convolutional Neural Network (CNN) object detection model, before presenting the experiments that have been performed.

Several experiments will be performed, investigating the importance of background, texture, image sizes, occlusion and truncation. A sample of what the generated images will look like is presented in Figure 3.1.

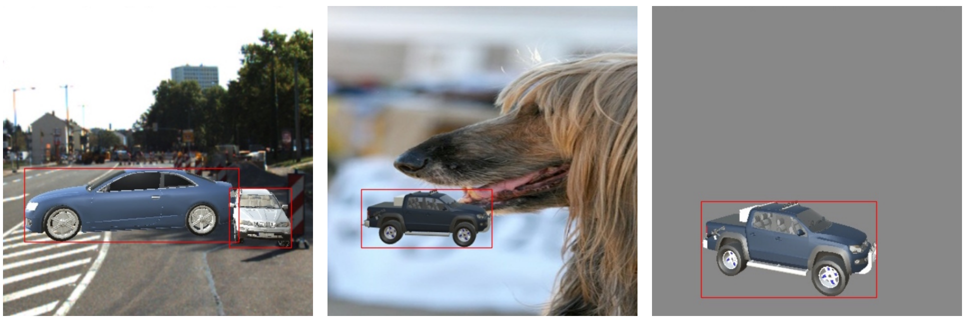


Figure 3.1: A sample of our generated images. The red box is the ground truth bounding box overlaid the image.

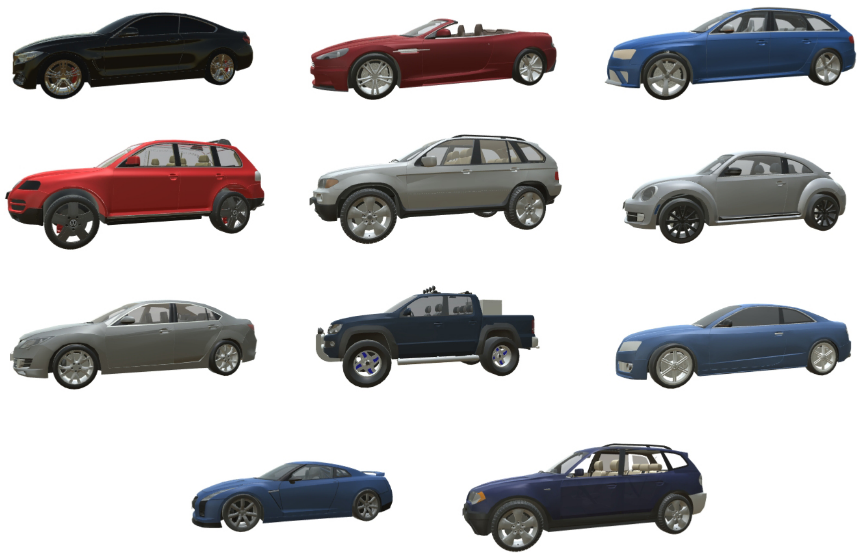


Figure 3.2: The 3D models used in all datasets downloaded for free from the Unity Store, TurboSquid and CGTrader

3.1 Synthetic Datasets

3.1.1 Car Models

Multiple resources for 3d models in a variety of file formats exist online. Examples of such sites that were used is the Unity Store[1], TurboSquid[2], CGTrader[3]. Most of the resources have a large selection of models that are free for academic and non-commercial use, and an even bigger library of cheap models. A search for 'car' on CGTrader yields more than 2,000 results when searching for free models, and around 13,000 when including paid models. 3D model prices range between $0-$400, and there are also collections of models that can be purchased for $400-$800.

3.1.2 Unity Scripts

In order to automate dataset generation we have developed a collection of Unity scripts. This section gives an overview of the scripts that were written. Unity scripts can be written in C# or JavaScript, and have access to manipulate all parts of the 3D scene, in addition to being able to interact with the filesystem. The scripts are able to render images of the scene to file through a virtual camera, calculate 2D

[1]https://store.unity3d.com
[2]http://turbosquid.com
[3]https://www.cgtrader.com

bounding boxes for objects and export the metadata in a way that maintains the references between bounding boxes and images. The full source code of our final scripts is available on GitHub [4].

Object Placement

To generate images with objects in different positions in each image, we use a stochastic approach where we randomly assign positions to all objects every frame. In order to achieve this, a custom component is written that moves the objects, by giving each of them a new sampled position from a user defined 3D volume. While more deterministic means of object placement are possible, like simulating video capture of a car driving through a virtual environment, this placement scheme requires far less setup and gives directly control of distributions in the dataset, such as the probability of having multiple cars in the same image.

In addition to random translation we also apply a random rotation around the Y-axis. Other objects than cars should be rotated around more axes, but almost every car in the evaluation sets, as well as in the real world, have all wheels on the ground, so rotations around any other axis is highly unusual.

Camera and Scene Setup

Each dataset is generated by placing a camera in a scene that contains multiple 3D models of cars, and applying the placement script to each object described in section 3.1.2. Figure 3.3 shows a top-down view of the scene setup in Unity. The camera has a custom script that takes a picture every frame and writes it to disk together with a file containing the bounding boxes of all relevant objects in the picture as described in 3.1.2.

In order to achieve a varied number of visible cars in the pictures, we define the 3D volume in which the cars are uniformly distributed every frame to be larger than the field of view of the camera. This leads to a variable amount of cars being outside the field of view. If there is a need for more control over the distribution of the number of visible cars, it is trivial to do this explicitly by adding or removing cars from the scene instead.

Picture Backgrounds

To achieve various backgrounds behind the objects of interests, a large plane which covers the camera's viewport is placed behind the bounding volume of the randomly placed objects. A collection of images is imported into Unity and applied as a texture to this plane, with a different one for every frame.

[4]https://github.com/lionleaf/mastertools/tree/master/unityscripts

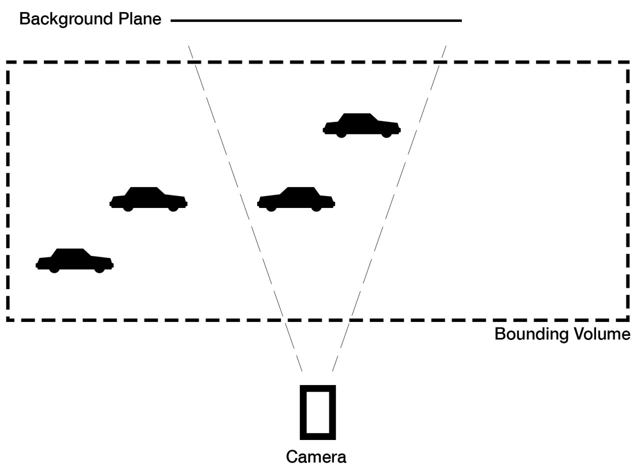


Figure 3.3: The bounding volume for the possible positions of car models is larger than the field of view of the camera, ensuring a random sample of cars to be visible in each image.

Experimentation with full 3D-scenes of a city were considered, but deemed too complex and domain-specific to represent a general approach for many object categories.

Lighting

Lighting conditions is one factor that varies a lot in the real world, so in order to generate a dataset with varying lighting conditions we have used the following setup. The scenes are rigged with a skybox including a Hemisphere Light, a Point Light and a Directional Light. The Hemisphere Light is completely static, and simulates light reflected by the ground as well as sunlight from the sky. Next, the Point Light, a light that shines in all directions, is positioned above the visible view volume. It is switched on and off for different pictures with a defined probability. Finally, the Directional Light rotates randomly around the Y axis to ensure that the models are not always lit from the same direction. Together, the three lights give varying reflections and amounts of light on models in different positions and rotations and across different images.

Generating Bounding Boxes from 3D Models

The big advantage of using computer generated images as training data, in addition to replacing manual photographing and/or collection, is the ease of generating accurate object bounding boxes. In Unity these bounding boxes can be generated by projecting all vertices of a 3D model into 2D screen space, and then extracting the minimum and maximum x and y position of all the vertices.

Producing a dataset using Unity can be done on a conventional laptop. Depending on the complexity of the images (number of models, type of background), a dataset consisting of 5000 images can be generated in under two hours. All our datasets have been generated on a MacBook Pro 13″ with an integrated Intel Iris Graphics 6100 GPU. Between 1 and 5 images are generated per second, depending on the complexity of the images, mainly type of background and number of 3D models. Using a more powerful desktop GPU would significantly speed this up.

3.2 Evaluation Datasets

In this thesis we have used two public datasets, KITTI Vision Benchmark Suite (KITTI) [GLU12] and PASCAL Visual Object Classes (PASCAL VOC) [EEG+14].

3.2.1 KITTI

The KITTI [GLSU13] object detection dataset is a road and car specific benchmark designed for autonomous vehicles. It contains annotated data for cyclists, pedestrians and cars. KITTI also provide a public highscore list where methods are ranked based on their performance on the official KITTI test set.

The dataset consists of 7481 images with labels for the three object categories *car*, *pedestrian* and *cyclist*. The labels are marked with a difficulty score of easy, moderate and hard based on size and visibility, but the official KITTI benchmark is ranked on the moderately difficult results. We base our car-only KITTI dataset on the labels that are marked as moderately difficult, following the official ranking of the KITTI results.

All images in the KITTI dataset are taken with the same camera and settings, and have the resolution of 1242 X 375 px. A sample of the KITTI images is shown in Figure 3.4.

The KITTI dataset was chosen as the main benchmark because of its realistic scenes taken on real roads under real conditions, and performance on this benchmark is expected to be close to what you would be able to get in a real system.

3.2.2 Pascal VOC

Pascal PASCAL VOC is a large object detection competition that ran from 2005 to 2012. The datasets from later competitions, 2007, 2010 and 2012 are still being used and can be combined to a dataset with 20 classes over 27090 images. While the

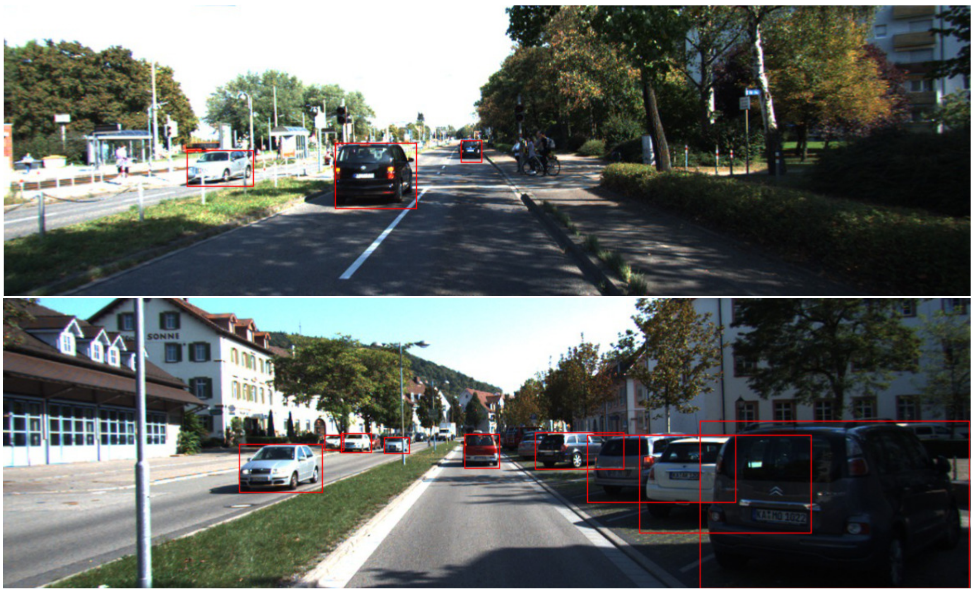


Figure 3.4: Example images from the KITTI dataset with ground truth bounding boxes for the car category overlaid

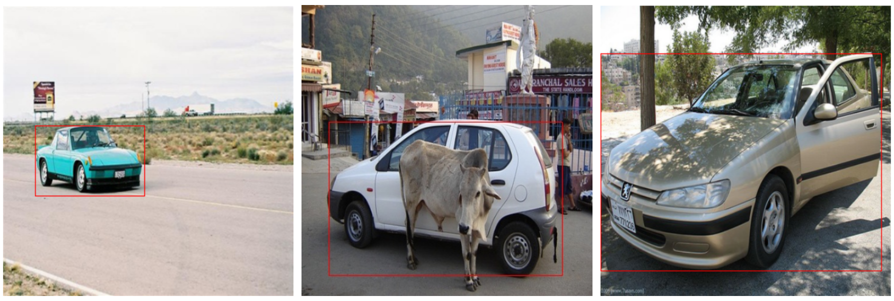


Figure 3.5: Sample images from the VOC dataset with ground truth boxes for the car category overlaid

competition is over it is still regularly used to benchmark systems for papers and as of 2016 there is still a live leaderboard with new results. We extract all the images containing labels for cars from the training and validation data of VOC 2007 and VOC 2012, which results in a dataset of 2595 labeled images containing one or more cars. A sample of the images in VOC containing cars is provided in Figure 3.5.

The PASCAL VOC dataset consists mainly of pictures where the cars are easier to detect than in KITTI. In many of the pictures the cars are closer to the camera, under better lighting and framed alone in the middle of the picture.

3.3 You Only Look Once

To evaluate and compare the performance of different types of computer-generated datasets, we have chosen to use one specific neural network architecture, YOLO. All experiments are performed using this network, so that the detection accuracies can be measured against each other. YOLO [RDGF15] was chosen for its high accuracy while still being real-time and small enough to possibly fit on an embedded device. At the time of our choice, YOLO was the best performing real-time object detection system available [RDGF15]. The paper [RDGF15] additionally mentions generalizing detections to art and other abstract forms, as discussed in section 2.4.2 which made it an interesting choice to investigate whether the generalization property also holds when training on synthetic data. The downside with YOLO is that it is implemented on custom Neural Network (NN) software that is not widely supported: Darknet.

3.3.1 Darknet

The reference implementation for the YOLO network presented by Redmon et al. [RDGF15] was written in the neural network framework called Darknet [5]. Darknet is open source, written in C, and takes advantage of CUDA for GPU acceleration. The CUDA requirement means it can not directly run on non-NVIDIA hardware, although alternative implementations of the necessary parts of YOLO are available in other frameworks as well.

3.3.2 Training Time and Pre-Trained Weights

As explained in the background chapter (section 2.4.2) the YOLO-network is pre-trained on ImageNet. This training takes multiple weeks on powerful hardware according to Redmon et al. [RDGF15]. The resulting weights are distributed along with the YOLO source code. This allowed us to use these weights already trained on ImageNet instead of random initialization when training on our datasets. Because this means the network does not have to learn basic image features from scratch, this reduces the convergence time to around 13 hours for Fast YOLO and around 67 hours for the full YOLO-network. This allows for faster iteration faster than if you have to spend multiple weeks per experiment.

3.3.3 Network Architecture Changes

The reference implementation of YOLO had a few hardcoded values that had to be changed. This was mainly the size of the output layer, as it was implemented for detection of 20 different categories, while we are only detecting a single class. For a one-class detection objective, the final layer needs to be customized. It predicts

[5]http://pjreddie.com/darknet/

bounding boxes within a 7×7 grid, and each grid cell predicts two bounding boxes in addition to a class probability. Each bounding box consists of 5 parameters: x, y, w, h, and confidence. This results in a $7 \times 7 \times 11$ output tensor on the final layer.

3.3.4 Variation in training

Training CNNs with stochastic gradient descent is an optimization search that is stochastic in nature and there are no guarantees to find the global optimum or to find the same result each time. Additionally, built-in data augmentation features in YOLO cause the training to result in a differently trained model each time. Two types of data augmentation is done automatically by YOLO. First, a random crop of the training images is selected by clipping a small part off of some of the edges. Second, a small random amount of saturation and/or exposure distortion is applied to some of the images. In order to draw any conclusions on our results we need to ensure that the differences in results are not just caused by random variance.

We perform an experiment where we use the same dataset for training three times, to investigate how much the resulting Average Precision (AP) varies. The experiment is done using the KITTI training data on the Fast YOLO network, and evaluated on the KITTI validation data.

3.3.5 Fast YOLO vs YOLO

The YOLO detection system comes with several pre-defined network structures. The reference network (hereafter YOLO), with 24 convolutional layers and two fully connected layers; and Fast YOLO, with 9 convolutional layers and three fully connected layers. The Fast YOLO model is designed to trade off detection accuracy for lower memory usage and increased speed. Fast YOLO only uses about 611 MB of GPU memory, which means it will fit on an embedded GPU like the Nvidia Tegra K1 [Nvi14]. The reported mean Average Precision (mAP) for YOLO on VOC 2007 is 63.4%, while the mean Average Precision (mAP) for Fast YOLO is 52.7%, roughly 11 percentage points less.

We seek to investigate whether the difference is similar between the two networks in terms of resulting detection performance when using synthetic datasets. To do this, we use the same dataset as training data for both networks and compare their results on several evaluation datasets. Additionally, we measure the training time and the time they need to perform detections.

Using Fast YOLO instead of YOLO is interesting for several reasons. Foremost, lower memory usage and faster detection speeds means it is more suitable for embedded systems and much more likely to be able to give real-time results. Second, reduced training time means it allows for running a higher number of experiments in the same amount of time as YOLO.

This experiment is aimed at exploring whether the trade-offs for using Fast YOLO can be justified or if the performance is too detrimental. By running this over multiple training datasets we will see if results on the Fast YOLO network can be used as a proxy for how the full YOLO network will perform; that is, if training on a particular dataset is better for Fast YOLO is it likely to be better for full YOLO as well? If that is the case, we propose using Fast YOLO to test performance on datasets as this allows much faster iteration.

3.4 Experiments

Several experiments were set up in order to investigate the viability of our proposed approach. There are many parameters that can be tested and varied, but in order to keep the focus on datasets and comparing as many different dataset strategies as possible, only one set of parameters have been used for training networks. This means using the same learning rate, dropout rate for all experiments, and limiting the network architectures to the two reference networks provided by YOLO.

3.4.1 Aspect Ratio

As mentioned in the background, YOLO converts all input images to a 448x448 RGB image before processing them, which means that various aspect ratios lead to different shapes of objects observed by the NN. As noted in the YOLO paper on page 4, YOLO struggles to generalize to objects in new or unusual aspect ratios. Further, the KITTI dataset contains only images of the same aspect ratio, which is very wide (3.31:1), whereas the PASCAL VOC dataset contains images of varying aspect ratios.

This experiment is designed to investigate how much the aspect ratio of images affects detection performance. We generate pairs of datasets where all parameters are equal except for the dimensions of the images. The groups of images are square (448x448 pixels) and wide (1484x448 pixels).

3.4.2 Image Backgrounds

We want to test the importance of realistic environments around the target object in the training data. Is only the object of interest itself needed in order to train a good detection model, or is it necessary to fully imitate real images? Previous work has shown that when using HOG features, natural backgrounds does not help in gaining better detection accuracy [SS14]. Others have shown that Deep Neural Networks (DNNs) are also largely invariant to backgrounds [PSAS14]. But because we are using a different object detection system, and the other systems showed some

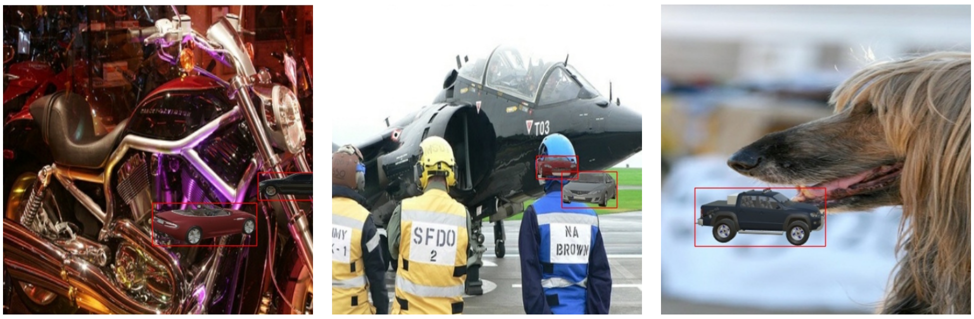


Figure 3.6: Dataset images with backgrounds from ImageNet, labeled bounding boxes for cars are highlighted as red rectangles

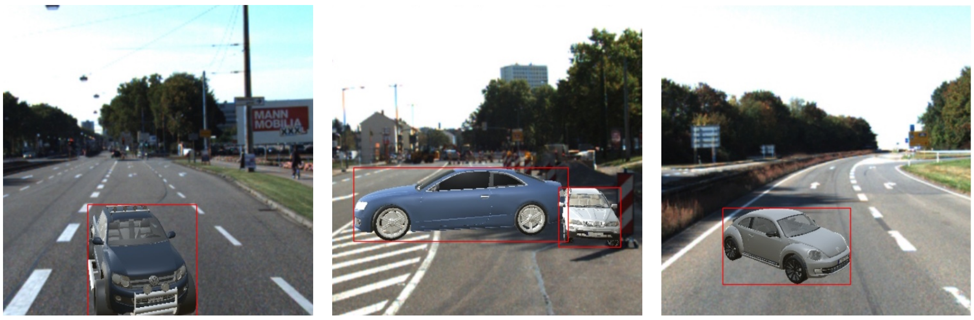


Figure 3.7: Dataset images with backgrounds from the KITTI Road/Lane Detection Evaluation dataset. Labeled bounding boxes for cars are highlighted as red rectangles.

differences in results, we want to evaluate to what degree backgrounds matter, and if so which ones are better.

The ImageNet classification challenge provides a dataset of images[DDS$^+$09], and we pick 1000 images that are labeled as not containing cars. For each image that is generated, a random image is applied as a background, as can be seen in Figure 3.6. These images do not necessarily depict scenes where cars would normally occur, but might still represent the variation of environments in which they should be detected.

We also create datasets where we use background images that are more relevant to the object of interest: in this case road images from the KITTI Road/Lane Detection Evaluation [GLSU13]. 43 of the images in the Road dataset are found to not contain any cars, and are therefore selected as suitable background images representing natural car environments. The datasets with KITTI-based backgrounds are shown in Figure 3.7.

To examine the importance of natural image backgrounds, we generate datasets

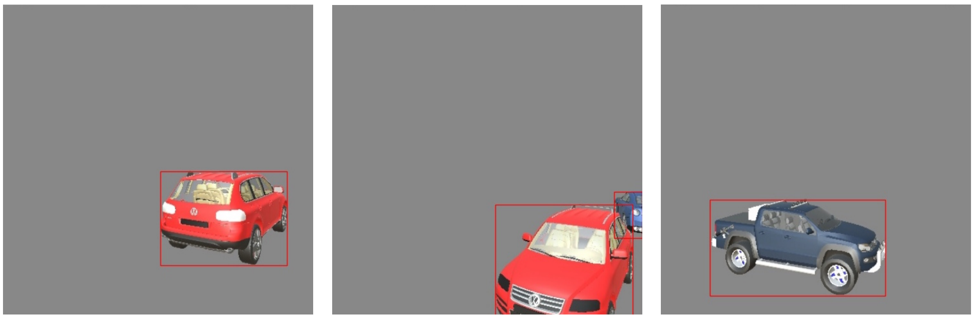


Figure 3.8: Dataset images with gray backgrounds

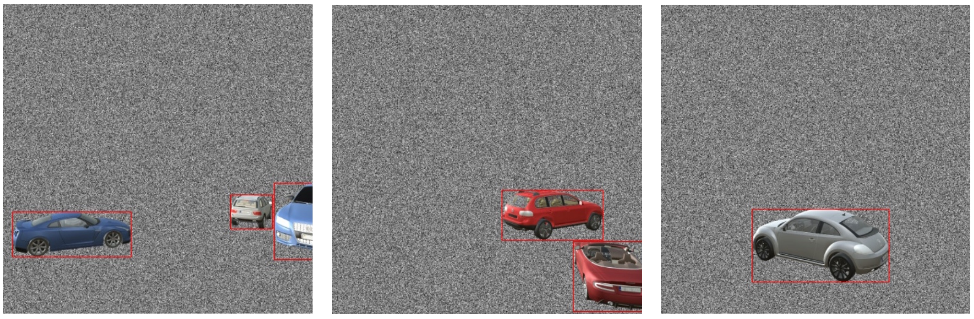


Figure 3.9: Dataset images with random noise backgrounds

where, instead of natural images, we use the simplest background possible: uniform gray (Figure 3.8).

As another test of alternative background approaches, we apply random grayscale noise in a separate dataset, as shown in Figure 3.9. This is different from the single gray color in that the background will be different in all images. We will compare the two alternative background strategies to see which produces the best results.

3.4.3 Texture Invariance

We have chosen to select only 3D models which comes with photorealistic textures. However, many free 3D models come with no such option. Thus, if realistic textures are unnecessary or have little impact on a model's performance, finding 3D models to use will be easier by relaxing this requirement. Notably, Peng et al. show that, for their detection system, some object classes are highly invariant to texture, while others depend heavily on it [PSAS14]. Stark et al. [SGS10] do not use textures when training their viewpoint detector, as they explicitly model a shape representation of objects. We generate a dataset with no textures on the models to investigate

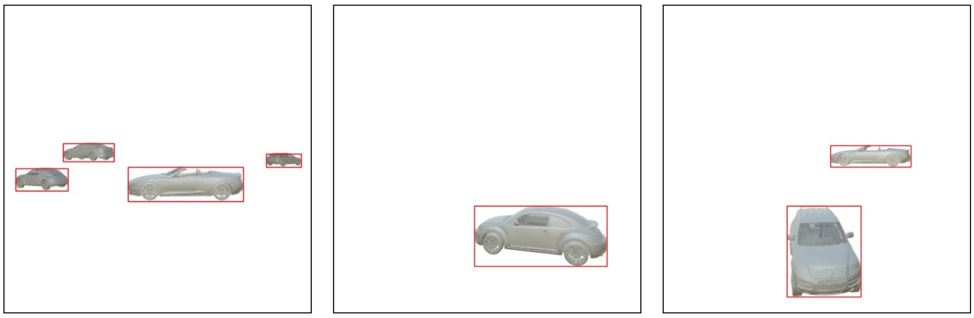


Figure 3.10: Dataset images with no texture on the 3D models

whether the NN will learn to detect cars in natural images based on shape alone (Figure 3.10).

3.4.4 Occlusion and Truncation

Different datasets have different rules for how to label objects, and for how much of the object must be visible in order to qualify for an annotation. If only a small part of the object can be seen in the image, it might be considered very hard to detect or not even necessary to detect, depending on the use case. For official benchmarks, which provide both training and test data, the differences in when objects are labeled and not are often the same for both datasets, thus object detection networks are typically trained on data with roughly the same labeling style as the data their performance will be measured on.

To understand how performance varies with differing levels of occlusion and truncation in the training data we generate several datasets. First, we use one baseline dataset containing only cars, where cars that are more than 65% out of frame are ignored. Images where two cars overlap more than 25% are discarded from this dataset. Overlap is measured by calculating the Intersection-Over-Union (IOU) of the two bounding boxes. Second, a dataset where cars were labeled as long as at least 5% of the object is visible. Here, images were kept as long as two objects overlap less than 75%. Finally, a dataset where several new objects were introduced. The same rules for overlap and truncation as in the previous set were used. Bounding boxes for the cars do not take into account whether a car was occluded by a non-car object. This means that for a lot of the objects, they are occluded by part of a tree, or an abstract 3D shape, while the label still covers all of the object.

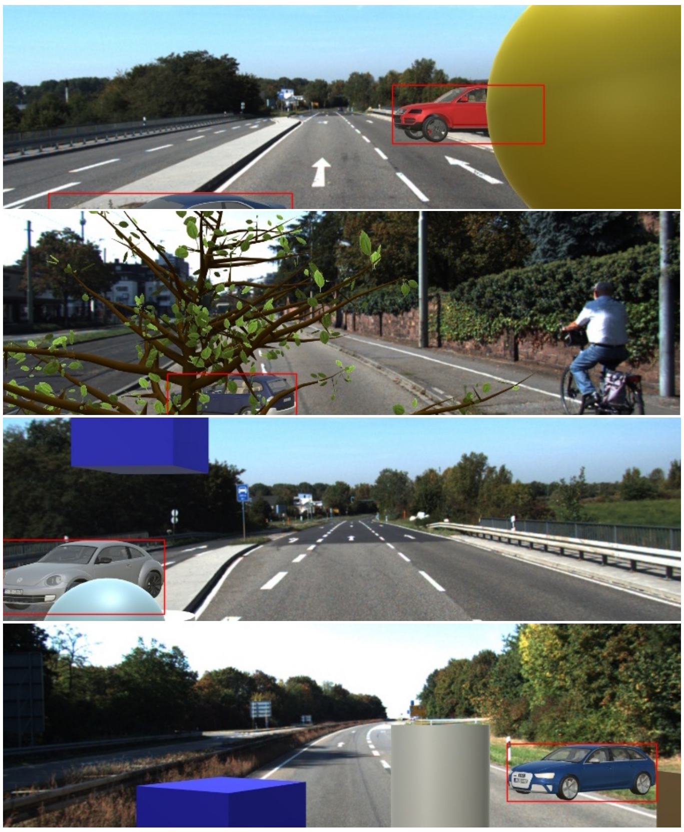


Figure 3.11: Dataset images with occlusion and truncation. Labels, highlighted in red, still outline the complete car shape within the bounds of the image.

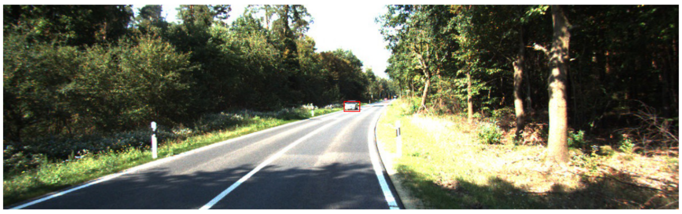


Figure 3.12: An example of a small bounding box from the KITTI dataset.

3.4.5 Small Objects

In detailed scenes, many current object detection algorithms are better at detecting larger objects than smaller objects, this is evident in that detection accuracies on bottles, which are often small, is significantly worse than bigger classes on the PASCAL VOC challenge as noted by Uijlings et al. in their Selective Search paper [UvGS13]. YOLO struggles even more with small objects than comparable systems for two major reasons. Firstly, the model can only predict two bounding boxes per grid cell, so if there are multiple small objects close together it is simply not possible to correctly detect all of them. Secondly, in the error analysis of the original paper [RDGF15] they show that localization errors, that is a bounding box that is further away from the ground truth than a set threshold, accounts for more than half of the errors, and smaller bounding boxes are harder to hit since a miss by a fixed amount of pixels will lead to a much lower overlap percentage than on a larger object.

The KITTI benchmark has a number of small bounding boxes, as well as multiple bounding boxes close together as seen in Figure 3.12

To investigate the impact of object sizes on our performance we create a filtered version of the KITTI evaluation set where all small bounding boxes have been removed and compare the performance to the full benchmark. We expect some difference here, but the magnitude of this difference would offer some insight as to how big this problem is.

3.4.6 Comparison to Virtual KITTI

The Virtual KITTI dataset, which was released during the final stages of this project, consists of 3D models of cars placed in complete 3D environments, and is made as a virtual replica of a subset of the KITTI training set. The dataset differs from our datasets in that its cars are positioned, lit and occluded more similar to KITTI. Therefore it will be interesting to compare the performance of a model trained on Virtual KITTI with the performance of models trained on our datasets.

3.4.7 Variability of Training

Because the training process of YOLO is stochastic in nature, every training will result in a different network. To measure this effect we re-run training with the same parameters three times in a row and calculate the standard deviation.

Chapter 4
Results

4.1 Setup

4.1.1 Hardware & Frameworks

For running training and experiments, we use two desktop computers with Nvidia GTX 980 graphics cards, which run Ubuntu 14.04. Datasets are generated using Unity version 5.3.4f1.

Darknet

All experiments have been executed with the same version of Darknet. We have forked the original Darknet library on GitHub [1], and the experiments have been run on commit `228464c`. Darknet has been compiled with CUDA support, using CUDA version 7.0 and cuDNN version 3.0.

4.1.2 Running Experiments

To prepare datasets for training, a line-separated list of filepaths for images in the dataset must be created. The images must be stored in a folder named `images`, and all labels must be stored with filenames equal to their corresponding images substituting `.jpg` for `.txt` in a folder called `labels` next to the image folder.

To start training, the following command is executed:

```
./darknet yolo train cfg/singleclass.cfg singleclass.weights
    > output.txt 2>&1
```

This will use the images references from the file train.txt in the same folder, use the configuration file `singleclass.cfg` and the pre-trained weights stored in `singleclass.weights`.

[1] https://github.com/stianjensen/darknet

To evaluate the performance of a fully trained model, run the following:

```
./darknet yolo valid cfg/singleclass.cfg singleclass_final.
    weights evaluation_dataset.txt
```

`evaluation_dataset.txt` should contain one image path per line, the same way as the list of training images. This will output a text file containing all predicted boxes in the format specified by PASCAL Visual Object Classes (PASCAL VOC) [EEG+14].

4.2 Dataset reference

An overview of all synthetic datasets that we will be looking at results from in this chapter is given in Table 4.1 for reference.

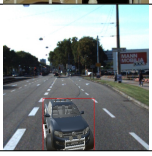
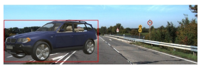
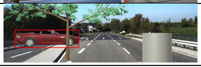
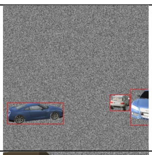
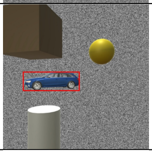
Name	Description	Square	Wide
Gray-BG	Gray Background		
IN-BG	ImageNet Background		
K-BG	KITTI Background		
K-BG (t)	KITTI Background, w/Truncation	N/A	
K-BG (t&o)	KITTI Background, w/Truncation and Occlusion	N/A	
Noise-BG	Noise Background		N/A
Noise-BG (o)	Noise Background, w/Occlusion		N/A
No-Tx	White Background, no texture on models		N/A

Table 4.1: Several of the datasets are generated in both square and wide variants. We will refer to the datasets by the names defined in column one throughout out the rest of the thesis.

4.3 Results

KITTI	Gray-BG	IN-BG	K-BG	K-BG (t)	K-BG (t&o)
Square	0.1	0.5	2.0	-	-
Wide	0.1	1.4	5.1	5.9	**11.9**
KITTI Filtered	Gray-BG	IN-BG	K-BG	K-BG (t)	K-BG (t&o)
Square	0.8	0.6	15.5	-	-
Wide	0.7	10.4	20.5	25.9	**38.0**
PASCAL	Gray-BG	IN-BG	K-BG	K-BG (t)	K-BG (t&o)
Square	1.6	6.7	7.4	-	-
Wide	10.4	13.6	19.6	**22.2**	16.3

Table 4.2: The full YOLO network results measured in Average Precision (AP) on the KITTI Validation set, KITTI Filtered (ignoring all bounding boxes smaller than 3% of the image) and the PASCAL VOC dataset.

KITTI	Gray-BG	IN-BG	K-BG	Noise-BG	Noise-BG (o)	No-Tx
Square	0.0	0.1	0.5	0.0	0.3	0.0
Wide	0.1	1.3	**2.6**	-	-	-
PASCAL	Gray-BG	IN-BG	K-BG	Noise-BG	Noise-BG (o)	No-Tx
Square	0.5	3.0	0.9	1.5	1.4	0.1
Wide	6.3	2.3	**8.6**	-	-	-

Table 4.3: Fast YOLO results measured in AP on the KITTI Validation set and the PASCAL VOC dataset.

Table 4.2 shows the results for the full YOLO network trained on our synthetic datasets while Table 4.3 displays the results obtained from training with the smaller Fast YOLO network. Recall that the experiments run on Fast YOLO is done for comparisons between datasets, not in order to achieve the highest possible AP, because running an experiment on Fast YOLO is much faster. Wide datasets scored consistently better than their square counterparts for both networks. It is also evident that all the models trained on the datasets with natural image backgrounds, `K-BG` and `IN-BG` significantly outperformed the datasets with gray or noise as background, `Gray-BG`, `Noise-BG` and `No-Tx`. Nonetheless, `Gray-BG Wide` performed better on `PASCAL` than any of the square datasets with natural backgrounds.

The introduction of occlusion in `K-BG (t&o)` doubled the performance on `KITTI` from 5.9% to 11.9%, but decreased performance on `PASCAL`, while the truncation increased performance for both datasets.

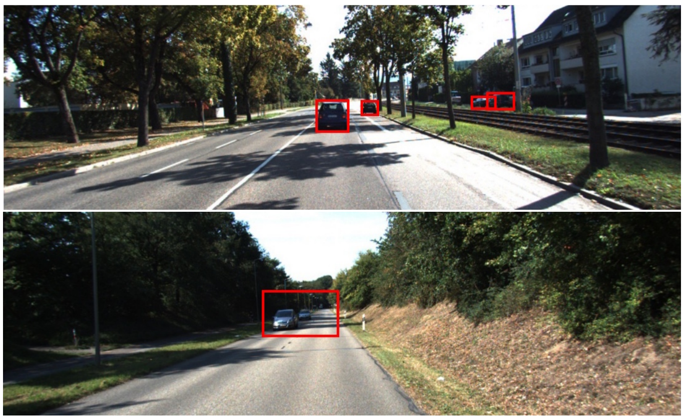


Figure 4.1: Detections on KITTI Validation by K-BG (t&o)

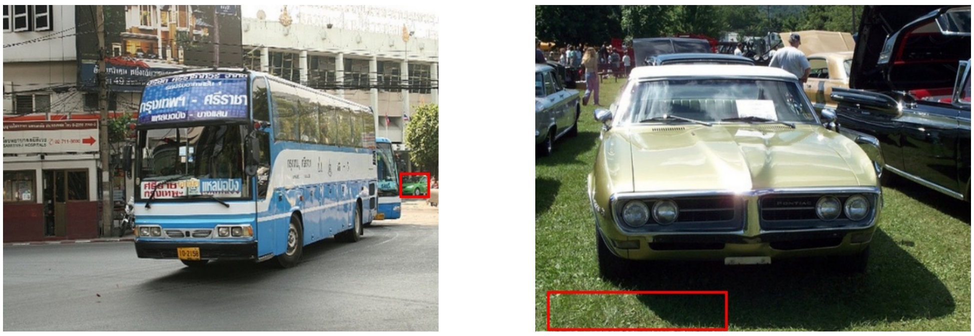


Figure 4.2: Detections on VOC by K-BG (t)

In `KITTI Filtered` we have removed all bounding boxes that cover less than 3% of the area of the image as explained in section 3.4.5. This increased the AP of all the experiments by a wide margin, and in many cases more than quadrupling it, showing that a disproportionate amount of the errors are on small objects.

4.3.1 Example Detections

Figure 4.1 shows an image where our best-performing model, `K-BG (t&o)`, on `KITTI` has accurately detected four cars in the KITTI dataset, and an image where two small cars are incorrectly misclassified as one. Note that for this second picture this detection will be counted as a total miss for the purpose of AP calculation. In Figure 4.2 we show detections made by the model we have trained with best performance on `PASCAL`. In the image on the left, a car in the background is correctly detected, and none of the buses in the foreground are misclassified as a car. In the image on the right, the predicted bounding box completely misses the car in the center of the frame. A larger random sample of detections performed by both of these models are presented in Appendix A.

4.4 Comparison to Virtual KITTI

	Virtual KITTI
KITTI Validation	20.6%
PASCAL	0.5%

Table 4.4: Results from training YOLO on the Virtual KITTI dataset.

Table 4.4 shows the results for YOLO trained on the Virtual KITTI dataset [GWCV16]. We see that performance on KITTI Validation is almost twice as high as for our best-performing dataset. On VOC however, Virtual KITTI performance is very poor, lower than all YOLO models trained on our datasets.

4.5 Variation

	First training	Second training	Third training
Kitti validation performance	54.24%	53.76%	52.92%

Table 4.5: AP on the KITTI validation data on three independently trained Fast YOLO-based models using the same training data.

We present the results in Table 4.5 showing the performance of a Fast YOLO-network when trained on the same training data three times. The resulting mean is 53.64% with a standard deviation of 0.67%.

We have not trained on all datasets multiple times to calculate the mean and standard deviation, but we use this calculation as a basis for the expected amount of variation in the performance of all models.

Chapter 5 Discussion

We set out to explore the viability of using synthetic datasets to train object detection models and explore what factors are important for the performance of such datasets. We argue that our results suggest that this method could be viable in cases where access to good training data is limited.

The results show a top Average Precision (AP) of 22.2% on the PASCAL Visual Object Classes (PASCAL VOC) dataset and 11.9% on KITTI Vision Benchmark Suite (KITTI). State of the art detection results on these benchmarks are 87.8% AP[1] and 90.06% [2] respectively. However, these results are using different object detection models, so it is more natural to compare with the results of You Only Look Once (YOLO) trained on the data from the benchmarks. YOLO achieves 82.8% AP[3] on KITTI and 55.9%[4] on PASCAL VOC. So it is evident that training on our synthetic datasets in place of high quality photographic datasets leads to significant drop in performance. This is not surprising, as training a model on a different domain than the one you are evaluating on is known to cause significant performance drops [KFS16]. Our performance on VOC is on par with the winner of the PASCAL VOC challenge in 2008[5] and close to the recent 100 fps DPM implementation of Sadegy and Forsyth [SF14] that got 25% mean Average Precision (mAP).

Comparing our results with that of Peng et al. [PSAS14] it would seem that their datasets are superior, as they report AP on car detection up to 36.0%, a big improvement over our 22.2%. However, when the `RR-RR` dataset from their paper which achieved 33.0% when trained on by R-CNN was trained on by YOLO instead, it only got 15.8%, significantly lower than our results, which is more in line with the lower sophistication of the image data. This illustrates how important the object

[1] http://host.robots.ox.ac.uk:8080/leaderboard/displaylb.php?challengeid=11&compid=4
[2] http://www.cvlibs.net/datasets/kitti/eval_object.php
[3] Trained and evaluated by us.
[4] http://host.robots.ox.ac.uk:8080/leaderboard/displaylb.php?challengeid=11&compid=4
[5] The old highscore site is no longer available, but a graph of the results was found here: https://people.eecs.berkeley.edu/~rbg/slides/rcnn-cvpr14-slides.pdf

detection network itself is, and makes it natural to believe that replacing YOLO by the the top performing networks would lead to a significant boost in accuracy.

Interestingly, the drop in performance of `RR-RR` from the Peng et al. paper when trained on YOLO is disproportionately large compared to the difference between YOLO and R-CNN on the VOC benchmarks. Hence R-CNN appears to be better at generalizing from synthetic to real data, opposite of what we thought when choosing YOLO over R-CNN based on their performance on synthetic data when trained on real data. The YOLO paper suggests that the reason R-CNN performs poorly when trying to detect objects in synthetic datasets is because its object proposal algorithm, Selective Search, is fine-tuned for natural images [RDGF15]. This might actually be a benefit the other way around; when trained on synthetic data it is only the CNN part that is retrained for the objects from the synthetic dataset and Selective Search maintains its tuning towards natural images and will still suggest high quality bounding boxes. The R-CNN network used by Peng et al. has been significantly improved in later versions known as Faster R-CNN [RHGS15] and testing our datasets on this network would be a natural next step.

5.1 Aspect Ratio

The KITTI dataset contains only images of the same very wide aspect ratio (3.31:1), whereas the PASCAL VOC dataset contains images of various aspect ratios. Looking at the results, we see a strong indication that performance of the models increases when the aspect ratio of the training data is similar to that of the evaluation data. This is in line with what can be expected from the design of YOLO, where every input image is scaled to the same resolution and aspect ratio, something that changes the shape of objects. In YOLO individual predictors in the grid structure specialize on objects in certain aspect ratios [RDGF15], making the network especially sensitive to aspect ratio changes.

5.2 Backgrounds

Datasets with natural images as backgrounds show better performance than datasets with gray and random noise backgrounds. The difference between single-color and natural image backgrounds is in contrast to what has been found by Peng et al. [PSAS14] and Sun et al. [SS14]. They showed that for several object categories performance was better when training on images with white background, and concluded that Convolutional Neural Networks (CNNs) are largely invariant to backgrounds. For the car category, their best accuracy was produced by training on images with white background and no texture on models. Their overall accuracy across 20 classes, the mAP, was however somewhat higher for natural image backgrounds and with image textures on models. Our dataset with white backgrounds and no textures has the lowest performance of all our datasets. One of the main differences between YOLO and R-CNN, the YOLO CNN processes the whole image in one pass, while R-CNN starts with around 2000 region proposals which are individually scored by a CNN as covered in section 2.4.2 and 2.4.2. The initial region proposals are generated by Selective Search [UvGS13], which is already highly tuned for proposals in natural images. A possible explanation might be that since the Selective Search proposes bounding boxes that are then fed to CNN, this leads to a much more localized approach, while YOLO does more affected by the global structure. Another explanation is that since we are retraining the whole YOLO network, overfitting becomes easier than with R-CNN where the object proposal step is not retrained.

5.3 Occlusion and Truncation

We saw that adding occlusion using other 3D models than cars increased performance on the KITTI Validation dataset, while reducing performance on PASCAL VOC. We hypothesize that this is due to KITTI images containing high amounts of occlusion, and that training with occlusion is therefore important. However, there may be one more reason why other 3D models than cars in training data might increase performance, even when the models do not actually occlude any car models. Especially in the case where backgrounds are simple, a CNN that is trained on images which contains only cars can achieve good performance on the training data simply by detecting the part of the image that is not uniformly gray. It does not necessarily have to learn any features specific to cars in order to detect them. For datasets with more complex backgrounds, it is not sufficient to detect non-gray areas, however. Still, it is possible that the CNN mainly learns to detect the difference between a natural-image background and a 2D rendering of a 3D model, regardless of specific features of the object. Our results do show that most trained models have learned to detect actual cars in natural images to some degree, although it is possible that the effects of training with only one class of 3D models have reduced the need for learning and prioritizing important object features such as wheels and windows in favor of general attributes of rendered 3D models.

Introducing non-car 3D renderings in the training data, as in the K-BG (t&o) dataset, might therefore have increased performance not only because it forced the CNN to recognize occluded cars, but because it needed to prioritize car-specific features in order to increase performance during training.

5.4 Small Objects

From the big AP boost of removing the smaller bounding boxes from KITTI it is evident that detecting small objects is a substantial issue for models trained on all of our datasets. As discussed in section 3.4.5 this is a weakness of YOLO, and our experiment showed how important this is for the KITTI benchmark. Using a model with better performance on small objects would likely give a significant performance boost on KITTI.

5.5 Comparison to Virtual KITTI

We saw that the model trained on Virtual KITTI [GWCV16] performed almost twice as good as the result of our best dataset on KITTI Validation. This shows that higher performance than what we achieved using our datasets is possible with purely synthetic datasets. Not all scenes from the original KITTI training data have been replicated, and the dataset does therefore not contain as much variation as the full KITTI training data. Creating a synthetic dataset with properties similar to the Virtual KITTI dataset with more variation may therefore be able to produce even higher accuracy than what we achieved here.

5.6 Variability of Training

As seen in Table 4.5 retraining the same network multiple times gives very similar results. While this is far from a thorough analysis of the variability of training, it should be enough to give some confidence in our results. Running every experiment multiple times was simply not feasible when one experiment took between one and three days, and a total of 39 models were trained during the course of this thesis[6]

5.7 Pretraining on ImageNet

We have presented the use of synthetic training datasets as a method of training detectors for new object classes which currently do not have natural image datasets. In our evaluation we have used cars as a benchmark of the possible performance of this approach. Since the YOLO network is already pre-trained on ImageNet, however, the weights of the network are already tuned to detect the 1000 object classes in the ImageNet dataset, one of which is cars. We are not certain how much this has affected the results, and whether training on synthetic datasets for a category not present in the ImageNet dataset will perform worse. It is possible that pre-training on ImageNet will yield an initialization of the network that is helpful not only for the specific ImageNet categories, since the first layers learn very basic features that are present in all natural images.

[6]Many of which were early calibration, minor bounding box annotation errors and exploration

Chapter 6 Conclusions

In this thesis the use of synthetic datasets for training CNN-based object detection systems has been investigated. Our results on the PASCAL Visual Object Classes (PASCAL VOC) and KITTI Vision Benchmark Suite (KITTI) datasets, while far from state of the art, show promising accuracy that indicates that this method can be viable and is an exciting area for further research.

Compared with two other papers employing synthetic datasets, we have achieved high performance and are to the best of our knowledge the only one with a randomly generated synthetic dataset system that places additional objects in the scene for occlusion, something that doubled our performance.

Somewhat surprisingly we observed different behaviour with regards to background images from earlier research. Previous research has shown that their CNN-based object detection systems were largely invariant to different types of backgrounds in their synthetic datasets [PSAS14], while datasets with realistic backgrounds performed consistently better in our experiments.

Only a first exploratory step has been taken into this field and it is believed that there is potential for significant improvements with further research, which would make this technique a valuable tool for creating object detection systems where training data is lacking. There are also many cases, such as the mushroom detection system discussed in the introduction, where high accuracy is not too important, and in such a case our approach works as a close to free off-the-shelf solution to get a reasonable object detection system.

Further, we believe that the use of synthetic data to explore what factors are important for object detection systems can prove very valuable, because they enable controlled experiments with large datasets that are simply not possible to do with real data.

6.1 Further work

Having shown that there is a lot of promise in generating synthetic datasets, there are many paths that can be followed for future research, and this section will suggest some of the ones the authors believes to be most promising.

For our initial proposal of creating an object detector for mushrooms, we have shown that it is possible to generate fully synthetic datasets using only a few 3D models. As mentioned in the discussion we have used an initialization of the You Only Look Once (YOLO) network pre-trained on ImageNet data, which in addition to cars do contain some images of mushrooms. Since both cars and mushrooms are present in the pre-trained data, any possible advantage of pre-training should remain for the mushroom category. Our results on detecting small objects accurately do however pose a challenge for mushrooms. Investigations into other object detection models should perform the same evaluations with respect to small objects as we have done with our filtered KITTI dataset.

Generating synthetic datasets for mushroom detection would be the next step in the pursuit of a mushroom detection system, followed by research into getting the system to run on an embedded device. At this point, a handheld device that alerts the user when it sees mushrooms could be evaluated before moving on to drone flight.

The current top-performing object detection model on PASCAL VOC is Faster R-CNN, the most recent, fastest and best-performing variant of the R-CNN system [RHGS15, GDDM14]. While Peng et al. used R-CNN in their experiments [PSAS14], and we employed the YOLO system [RDGF15], it would be interesting to see how well Faster R-CNN performs with different types of synthetic datasets. Although Faster R-CNN is not as real-time as YOLO with a reported 5fps vs. 45fps, Faster R-CNN also provide a smaller version of the network which trades off some accuracy for an increase in speed to 17fps. This could be useful for many real-time and near-real-time applications.

Synthetic datasets introduce a whole new range of hyperparameters that can be adjusted to boost the detection performance, such as occlusion, truncation, object density and distance to camera, number of different objects of the same class, backgrounds and rendering parameters such as lighting among others. It is therefore possible to create a very large collection of datasets with small variations in properties, and thereby boost the performance of the detector on a validation set. While generating datasets is relatively fast, training a new model on each of them is costly, thus limiting the granularity of parameters that may be tested with constrained time. However, it is possible to prematurely end the training and get a network that might still be useful as an indication of how good the dataset will perform. Using this combined with a smaller network such as Fast YOLO, it is possible to systematically test these hyperparameters one step at a time. Plotting the AP performance with small changes for any number of these hyperparameters could prove valuable.

A deeper understanding of the connections between the hyperparameters of the datasets and the statistical distributions of the evaluation data is needed, and insights here could help remove much of the guesswork from choose these hyperparameters. We envision a future system that enables a two-stage approach to building object detection systems for novel objects. First a small but representative dataset is captured, and then statistics are calculated on this small dataset that guides the hyperparameter choice for synthetic dataset generation. Such an approach would be a huge reduction in cost, possibly even enabling the use in new domains where it is currently too expensive.

Looking at the training images generated it is evident that they do not look realistic. One obvious way to increase realism is to model a whole 3D environment, like Gaidon et al. did for Virtual KITTI [GWCV16]. This is far more labor intensive, especially if you want to generate datasets for hundreds of different objects that occur in different kind of scenes, but it could boost detection performance and hopefully lead to insights as to what factors are important.

A venue that has been left unexplored by this thesis is the effect of adding simple filters to the generated images, such as noise or compression artifacts. In the same category is variations on the textures of the objects, such as random colors or strong colored lighting.

To get a better understanding of the importance of pre-training when using synthetic data, experiments could be done without using a pre-trained network to compare the performance. We would expect these networks to be inferior, as the basic features learned from the neural network might be severely overfitted for simple synthetic data, but it is still worth attempting. Additionally, generating synthetic datasets for a category not present in ImageNet would show how much the pre-training helps for completely new categories. It would however be difficult to directly compare the results of a new category with the results on existing categories since they can not be evaluated on the same benchmark.

Chapter A

Detections

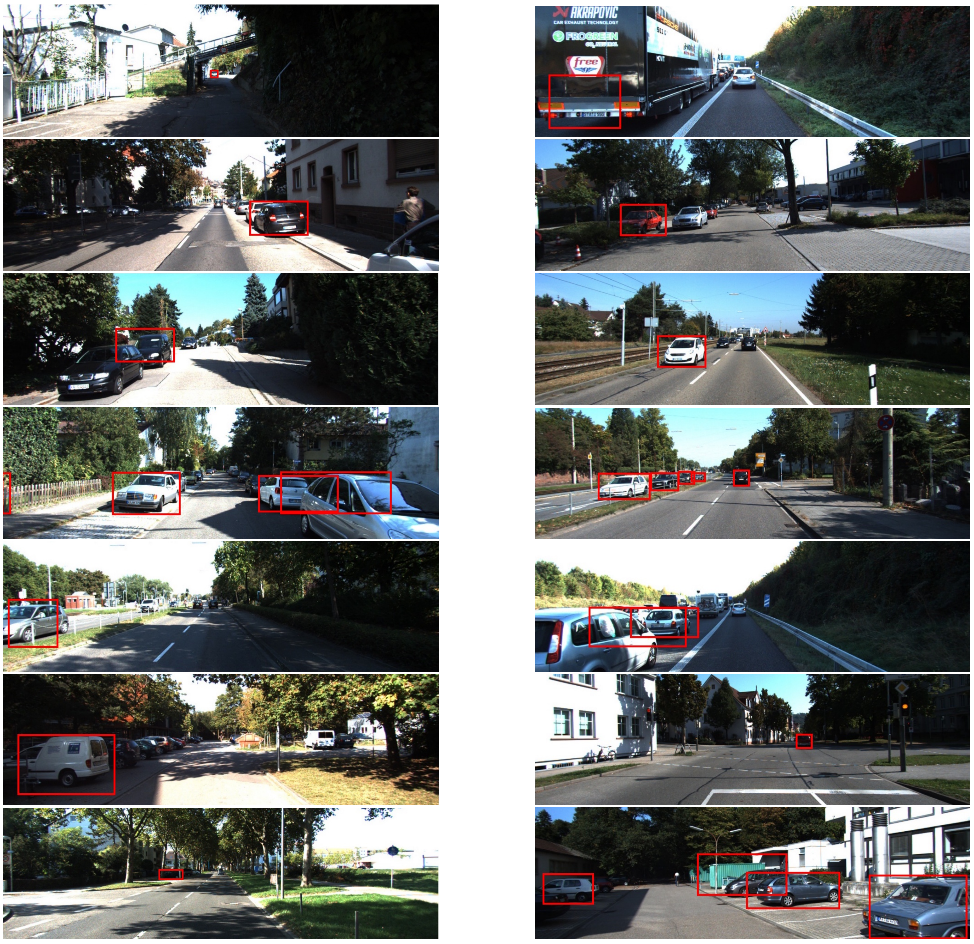


Figure A.1: Randomly sampled detections made by our best performing detector on KITTI Validation K-BG (t&o), from the KITTI Validation dataset.

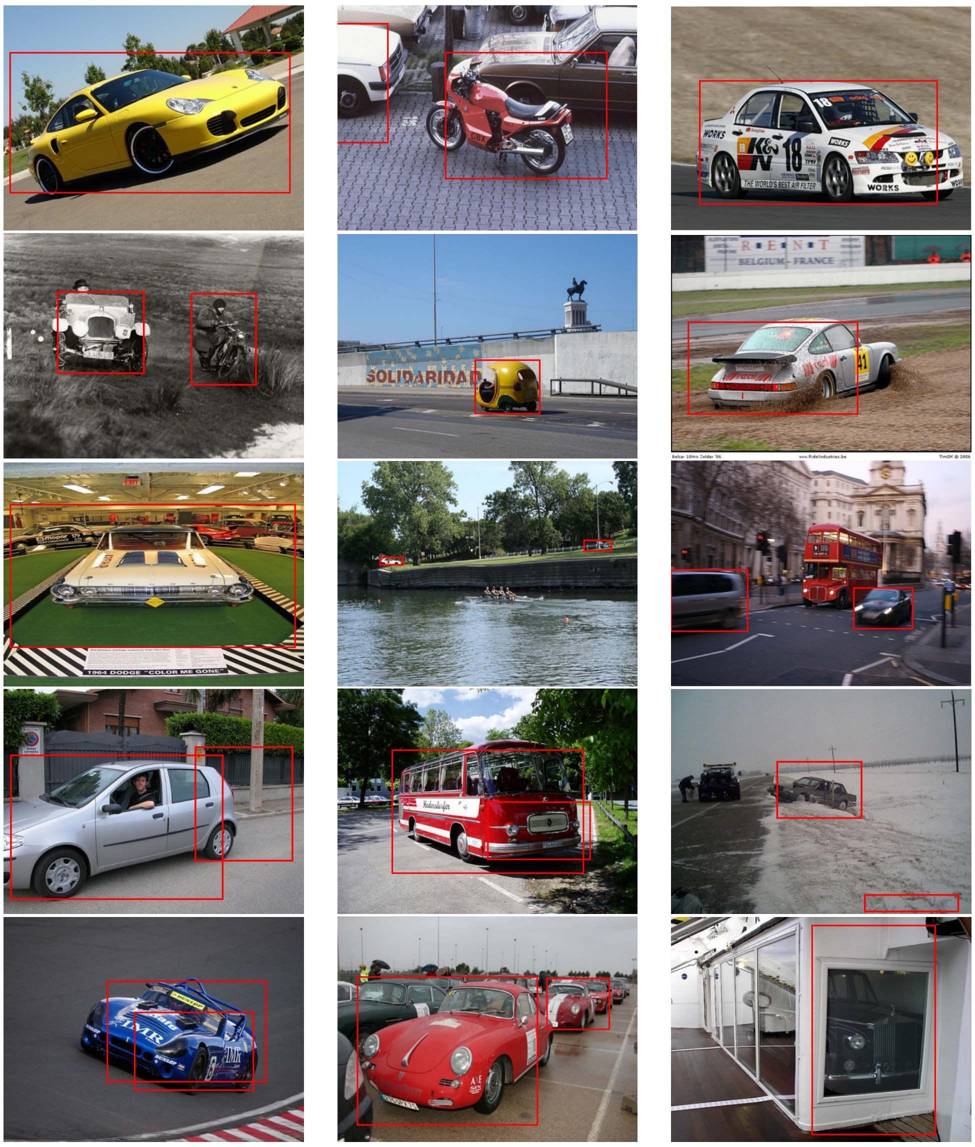


Figure A.2: Randomly sampled detections made by our best performing detector on Pascal VOC, K-BG (t), from the VOC dataset.

References

[BB08] Olivier Bousquet and Léon Bottou. The Tradeoffs of Large Scale Learning. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 161–168. Curran Associates, Inc., 2008.

[Bil06] Stanley Michael Bileschi. *StreetScenes: Towards scene understanding in still images*. PhD thesis, Massachusetts Institute of Technology, 2006.

[BLP+07] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, and others. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[BTG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded Up Robust Features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, number 3951 in Lecture Notes in Computer Science, pages 404–417. Springer Berlin Heidelberg, May 2006.

[CBD+90] Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten Digit Recognition with a Back-Propagation Network. In *Advances in Neural Information Processing Systems*, pages 396–404. Morgan Kaufmann, 1990.

[CSVZ14] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. *arXiv:1405.3531 [cs]*, May 2014.

[Cun85] Y Le Cun. Une procedure d'apprentissage pour reseau a seuil assymetrique. *Proceedings of Cognitiva*, 85:599–604, 1985.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[DDS+09] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[DT05] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005*, volume 1, pages 886–893 vol. 1, June 2005.

[EBC$^+$10] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[EEG$^+$14] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*, 111(1):98–136, June 2014.

[ESTA13] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable Object Detection using Deep Neural Networks. *arXiv:1312.2249 [cs, stat]*, December 2013.

[FM82] Kunihiko Fukushima and Sei Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern recognition*, 15(6):455–469, 1982.

[GDDM13] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv:1311.2524 [cs]*, November 2013.

[GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[GEB15] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A Neural Algorithm of Artistic Style. *arXiv:1508.06576 [cs, q-bio]*, August 2015.

[GHBM14] Shiry Ginosar, Daniel Haas, Timothy Brown, and Jitendra Malik. Detecting People in Cubist Art. In Lourdes Agapito, Michael M. Bronstein, and Carsten Rother, editors, *Computer Vision - ECCV 2014 Workshops*, number 8925 in Lecture Notes in Computer Science, pages 101–116. Springer International Publishing, September 2014.

[Gir15] Ross Girshick. Fast R-CNN. *arXiv:1504.08083 [cs]*, April 2015.

[GLF$^+$93] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett. DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1. *NASA STI/Recon Technical Report N*, 93:27403, February 1993.

[GLSU13] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets Robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[GLU12] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, June 2012.

[GMH13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech Recognition with Deep Recurrent Neural Networks. *arXiv:1303.5778 [cs]*, March 2013.

[GWCV16] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual Worlds as Proxy for Multi-Object Tracking Analysis. *arXiv:1605.06457 [cs, stat]*, May 2016.

[H06] G. E. Hinton and R. R. . Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, July 2006.

[HOT06] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, May 2006.

[HSK+12] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580 [cs]*, July 2012.

[HW68] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, March 1968.

[HZRS14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *arXiv:1406.4729 [cs]*, June 2014.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852 [cs]*, February 2015.

[JSVZ14] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition. *arXiv:1406.2227 [cs]*, June 2014.

[KFS16] V. Kalogeiton, V. Ferrari, and C. Schmid. Analysing domain shift factors between videos and images for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP(99):1–1, 2016.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[LAE+15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, and Scott Reed. SSD: Single Shot MultiBox Detector. *arXiv:1512.02325 [cs]*, December 2015.

[LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[LBOM12] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*, number 7700 in Lecture Notes in Computer Science, pages 9–48. Springer Berlin Heidelberg, 2012.

[Low99] D.G. Lowe. Object recognition from local scale-invariant features. In *The Proceedings of the Seventh IEEE International Conference on Computer Vision, 1999*, volume 2, pages 1150–1157 vol.2, 1999.

[LS10] J. Liebelt and C. Schmid. Multi-view object class detection with a 3D geometric model. In *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1688–1695, June 2010.

[MK91] M. Marefat and R. L. Kashyap. Image interpretation and object recognition in manufacturing. *IEEE Control Systems*, 11(5):8–17, August 1991.

[Nvi14] Nvidia. Whitepaper NVIDIA Tegra K1. Technical, January 2014.

[OGK$^+$15] Seyyed Salar Latifi Oskouei, Hossein Golestani, Mohamad Kachuee, Matin Hashemi, Hoda Mohammadzade, and Soheil Ghiasi. GPU-based Acceleration of Deep Convolutional Neural Networks on Mobile Platforms. *arXiv:1511.07376 [cs]*, November 2015.

[Par85] DB Parker. Learning logic. Technical report, 1985.

[PCCo06] Christopher Poultney, Sumit Chopra, Yann L Cun, and others. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, pages 1137–1144, 2006.

[PNDM03] David Pilz, Lorelei Norvell, Eric Danell, and Randy ; Molina. *Ecology and management of commercially harvested chanterelle mushrooms.* 2003.

[PSAS14] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Learning Deep Object Detectors from 3D Models. *arXiv:1412.7122 [cs]*, December 2014.

[PSGS12] B. Pepik, M. Stark, P. Gehler, and B. Schiele. Teaching 3D geometry to deformable part models. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3362–3369, June 2012.

[RDGF15] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *arXiv:1506.02640 [cs]*, June 2015.

[RHGS15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv:1506.01497 [cs]*, June 2015.

[RMN09] Rajat Raina, Anand Madhavan, and Andrew Y. Ng. Large-scale Deep Unsupervised Learning Using Graphics Processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 873–880, New York, NY, USA, 2009. ACM.

[RUM86] D. E. RUMMELHART. Learning representations by back-propagation errors. *Nature*, 323:533–536, 1986.

[SEZ+13] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *arXiv:1312.6229 [cs]*, December 2013.

[SF14] Mohammad Amin Sadeghi and David Forsyth. 30Hz Object Detection with DPM V5. *European Conference on Computer Vision*, 2014.

[SFF07] S. Savarese and Li Fei-Fei. 3D generic object categorization, localization and pose estimation. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8, October 2007.

[SGS10] Michael Stark, Michael Goesele, and Bernt Schiele. Back to the Future: Learning Shape Models from 3D CAD Data. In *BMVC*, volume 2, page 5, 2010.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[SKCL13] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun. Pedestrian Detection with Unsupervised Multi-stage Feature Learning. In *2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3626–3633, June 2013.

[SKFD10] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting Visual Category Models to New Domains. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, number 6314 in Lecture Notes in Computer Science, pages 213–226. Springer Berlin Heidelberg, September 2010.

[SLH12] Scott Satkin, Jason Lin, and Martial Hebert. Data-Driven Scene Understanding from 3D Models. *Proceedings of the 23rd British Machine Vision Conference (BMVC)*, September 2012.

[SLJ+14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *arXiv:1409.4842 [cs]*, September 2014.

[Sor14] Paul Benjamin Soren Goyal. Object Recognition Using Deep Neural Networks: A Survey. 2014.

[SRE+05] J. Sivic, B.C. Russell, A.A. Efros, A. Zisserman, and W.T. Freeman. Discovering objects and their location in images. In *Tenth IEEE International Conference on Computer Vision, 2005. ICCV 2005*, volume 1, pages 370–377 Vol. 1, October 2005.

[SS14] Baochen Sun and Kate Saenko. From Virtual to Reality: Fast Adaptation of Virtual Object Detectors to Real Domains. pages 82.1–82.12. British Machine Vision Association, 2014.

[SSP03] Patrice Y Simard, David Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

[SZ14] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, September 2014.

[TGJ$^+$14] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christopher Bregler. Efficient Object Localization Using Convolutional Networks. *arXiv:1411.4280 [cs]*, November 2014.

[UvGS13] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective Search for Object Recognition. *International Journal of Computer Vision*, 104(2):154–171, April 2013.

[VJ01] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2001. CVPR 2001*, volume 1, pages I–511–I–518 vol.1, 2001.

[WBB11] Kai Wang, B. Babenko, and S. Belongie. End-to-end scene text recognition. In *2011 International Conference on Computer Vision*, pages 1457–1464, November 2011.

[Wer74] P Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, 1974.

[WG15] Haibing Wu and Xiaodong Gu. Towards dropout training for convolutional neural networks. *Neural Networks*, 71:1–10, November 2015.

[ZF13] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. *arXiv:1311.2901 [cs]*, November 2013.