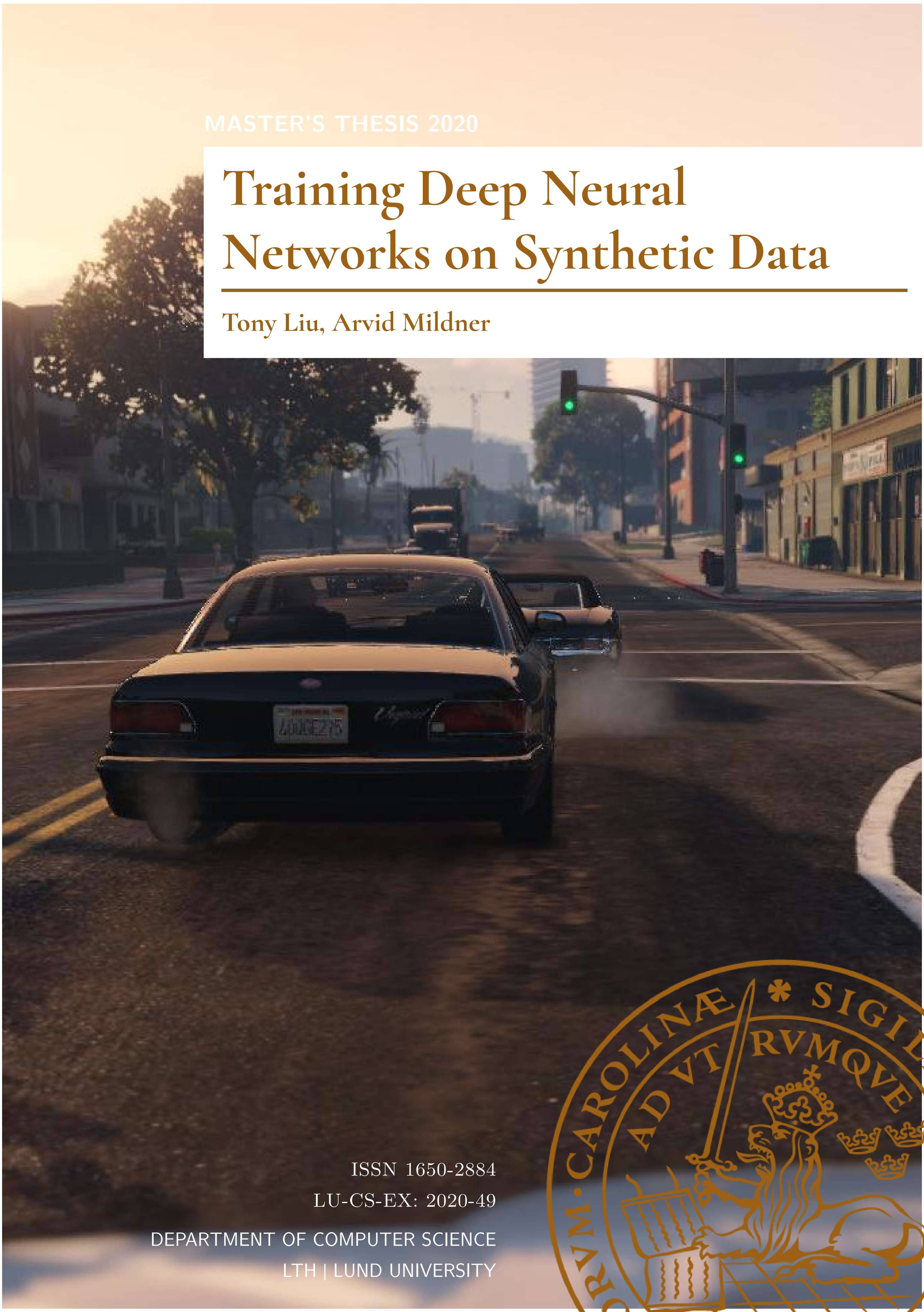


MASTER'S THESIS 2020

Training Deep Neural Networks on Synthetic Data

Tony Liu, Arvid Mildner

ISSN 1650-2884
LU-CS-EX: 2020-49

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2020-49

Training Deep Neural Networks on Synthetic Data

Tony Liu, Arvid Mildner

Training Deep Neural Networks on Synthetic Data

(Analysis of the Effect on Object Detection in Cityscapes)

Tony Liu
to5541li-s@student.lu.se

Arvid Mildner
ar3007mi-s@student.lu.se

August 19, 2020

Master's thesis work carried out at Axis Communication AB.

Supervisors:
Pierre Nugues, pierre.nugues@cs.lth.se
Martin Ljungqvist, martin.ljungqvist@axis.com
Otto Nordander, otto.nordander@axis.com

Examiner: Jörn Janneck, jorn.janneck@cs.lth.se

Abstract

Artificial neural networks have been used to solve complex tasks across different application areas. To train well-behaved generalizing neural networks, sufficiently large and diverse datasets are needed. Collecting data while adhering to privacy legislation becomes increasingly difficult and annotating these large datasets is both a resource-heavy and time-consuming task. An approach to overcome these difficulties is to use synthetic data generated by a computer for training. In this thesis, we investigate this idea by training the object detector YOLOv3 on synthetic images. Moreover, we study the difference between models trained on real and models trained on synthetic data.

We present a performance comparison between leveraging synthetic data and training on real data only with different proportions of the real training set. Our results show that performance can be increased using synthetic data when small amounts of real data is available. However, this relative performance increase is not as noticeable as more real training data is introduced. Additionally, fine-tuning from a base model trained on synthetic data seems to yield higher robustness to hyper-parameters during training.

To give insights on how models trained on synthetic and real data differ, we applied different similarity and sensitivity metrics to the networks on a layer-wise basis. The results of this analysis indicate that the models leveraging synthetic makes predictions with fundamentally different weight configurations and generalizes better compared to a model trained only on real data.

Keywords: Machine Learning, Deep Learning, Synthetic Data, Computer Vision, Object Detection, Convolutional Neural Networks, YOLOv3

Acknowledgements

We would like to thank our supervisors at Axis Communications, Martin Ljungqvist and Otto Nordander for great discussions and continuous feedback throughout our work. We would also like to thank Axis Communications for giving us this opportunity as well as the colleagues at Core Technologies Analytics for helpful advice along the way.

We would like to extend our gratitude to our supervisor at LTH, Professor Pierre Nugues for guidance and great advice during our thesis project. His support and enthusiasm for our work has been much appreciated.

Contents

Chapter 1
Introduction

1.1 Background

In computer vision, object detection is a sub-discipline which involves detecting instances of objects and classifying them semantically. Object detection is widely used in various different areas such as video surveillance (Yeh et al., 2017), autonomous vehicles (Chen et al., 2016), medical technology (Jaeger et al., 2018), and face-detection (Jiang and Learned-Miller, 2016). The task is ultimately about locating objects and classifying these object instances. This problem was previously solved using traditional machine learning methods such as SIFT (Lowe, 2004) and HOG (Dalal and Triggs, 2005).

In recent years, a popular approach to solve the object detection problem is to use deep learning – more specifically *convolutional neural networks* (CNNs). A lot of effort has been put into developing accurate and fast object detectors leveraging the structure of convolutional layers (Redmon and Farhadi, 2018; Ren et al., 2015; Lin et al., 2017; Long et al., 2014). This has led to a drastic increase in performance of object detectors during the past few years. However, these models generally require massive amounts of labeled training data to achieve good performance and generalization (Nowruzi et al., 2019). Building these datasets can be both time consuming and resource heavy.

First of all, the raw data needs to be collected, often involving complex data acquisition setups and gathering schemes. Additionally, when collecting data in public spaces, privacy concerns, such as the General Data Protection Regulation (GDPR) in the European Union (European Commission, 2018), need to be taken into account. Adhering to new data protection regulations and, at the same time, ensuring the diversity and quantity of the data becomes an increasingly difficult challenge. Secondly, the data needs to be annotated. This issue has previously been tackled using, for example, crowd-sourcing such as Amazon MTurk (Amazon, 2005) or semi-supervised learning methods (Odena, 2016). However, these approaches involve some kind of manual human labour. Since the training sets often include several thousand images, the annotation process becomes an extremely mundane and time-consuming task.

One way of avoiding these issues is using synthetic data for training. Generated synthetic datasets are inherently scalable and labelling of the data can be done automatically. These datasets can for example be generated using a data generation tool such as Synscapes (Wrenninge and Unger, 2018), Carla (Dosovitskiy et al., 2017), and Synthia (Ros et al., 2016), or sampling videos from open-world video games like GTAV, which has been done in Richter et al. (2017) and Johnson-Roberson et al. (2016).

A general problem with deep neural networks is that their complexity makes it difficult to understand exactly why a certain prediction has been made. This has led to neural networks often being considered as *black boxes* (Fong and Vedaldi, 2017; Alain and Bengio, 2016), meaning that we only look at the input and the output, while relying on trial and error when creating a well-working system.

In this thesis, we will investigate how object detection models are affected when trained on synthetic data versus real data by exposing the inner workings of the network. One key element will be the comparison between the outputs from individual hidden layers in the models using different novel ideas of similarity and criticality measurements.

1.2 Purpose and Delimitations

The purpose of this thesis is to investigate how synthetic data affects the performance of object detection models as well as how hidden layers in the CNNs are affected by different types of training data. More specifically, our project aims to answer the following questions on the matter:

- Can we increase the performance of existing object detection models by introducing synthetic data in the training sets?
- How does a model trained on synthetic data differ from one trained on real data and which network layers are affected?

The report aims to present two major parts that should be relevant for anyone looking to integrate synthetic data into their object detection system.

1. First we will present a thorough parameter search section together with a transfer learning approach to maximize the utility of synthetic data given a limited amount of real data.
2. In the second part, we experiment with some novel ideas on how the individual layers of the models can be interpreted mathematically to find similarity between models trained on real and synthetic data compositions. A main goal here is to see if the results from the analyzes described in Kornblith et al. (2019) and Chatterji et al. (2019) apply on our use case are coherent and in line with our intuition.

1.2.1 Delimitations

We will not try to implement new network architectures to increase performance when training on synthetic data. Instead, we will focus on one of the current state-of-the-art architectures, namely YOLOv3 as described by Redmon and Farhadi (2018) and try to understand how it is affected by training on synthetic data.

As we wanted to focus on performance comparison and similarity analysis more than wrangling with different datasets, we ended up using one synthetic dataset: GTAV produced by Richter et al. (2017) and one real dataset: BDD from Yu et al. (2018) for the experiments in this thesis.

The report is also scoped in that we are not performing any experiments on what improvements can be done once similar layers have been found. The investigative part of the report is meant to give an insight on these novel similarity metrics and to see if there are some patterns to be found. Our hope is that the knowledge gathered from these experiments can be useful when designing new architectures or training methods.

1.3 Related Work

1.3.1 Object Detection

Many object detection models use a so called two-stage region-based detection process, popularized by Girshick et al. (2013). The first stage proposes a sparse set of candidate object locations and the second stage classifies the candidate as either a background or foreground object. Network architectures such as MaskRCNN (He et al., 2017), FPN (Lin et al., 2016) and Faster-RCNN (Ren et al., 2015) use this two-stage process and have successfully reached high performance on standard datasets such as COCO (Lin et al., 2014).

A limitation of these two-stage techniques is their comparatively slow inference speed making them less suitable for usage in real-time object detection in video. More recently, several one-stage methods which sample densely on the set of object locations, scales, and aspect ratios, have been proposed such as YOLO (Redmon and Farhadi, 2018, 2016; Redmon et al., 2015), Retina-NET (Lin et al., 2017), and SSD (Liu et al., 2015). These networks are significantly faster while having comparable performance to the conventional two-stage methods. Because of its speed, comparable accuracy, and relatively light-weightness, we have chosen to use YOLOv3 (Redmon and Farhadi, 2018) in our experiments.

1.3.2 Synthetic Data

There are several synthetic datasets of city environments available and several experiments of training on synthetic data have been conducted. VKITTI is presented in Gaidon et al. (2016) and Cabon et al. (2020) which is the synthetic version of the KITTI dataset (Geiger et al., 2013). Synthia (Ros et al., 2016) is another synthetic dataset of images from urban scenes. The results from Ros et al. (2016) showed increased performance when training on a mixture of real and synthesized images. Richter et al. (2017) and Johnson-Roberson et al. (2016) leverage the open-world video game *Grand Theft Auto V (GTA V)* to generate synthetic datasets. The experiments conducted in Johnson-Roberson et al. (2016) show that training a Faster R-CNN on a GTA V synthetic dataset of at least 50,000 images increases the performance compared to training on a the smaller real dataset Cityscapes (Cordts et al., 2016) when evaluated on the real KITTI dataset (Geiger et al., 2013). However, these experiments only used cars as labels, disregarding other important labels such as persons and bicycles.

Wrenninge and Unger (2018) present the dataset Synscapes, which is a synthetic version of Cityscapes. The authors claim that training on only Synscapes yields decent results, but lowers performance compared to training on real data when evaluated on Cityscapes. However, their experiments show that models trained on Synscapes outperform both models trained only on GTA V (Richter et al., 2017) and Synthia (Ros et al., 2016). Furthermore Wrenninge and Unger (2018) claim that training on a mixture of synthetic and real data can further improve performance, outperforming models trained only on real data. Results from Nowruzi et al. (2019) show that training on synthetic data and fine-tuning on real data yield better performance than training on a mixed real-synthetic dataset. The authors also conclude that photo-realism in the synthetic data is not necessarily as important as other factors in the training such as diversity. Tremblay et al. (2018) produce non-artistically generated images by *domain randomization*, where parameters such as lighting, pose, and textures are randomized. The authors show that with additional fine-tuning on real-data, their model outperforms a models trained only on real data for object detection of cars on the KITTI dataset. Furthermore, they argue that letting the backbone be trainable during training on synthetic data yielded better performance compared to freezing the backbone weights.

Lee et al. (2016) use synthetic data for pedestrian detection and pose estimation. They show that training on synthetic images only can yield a model that outperforms a model trained on real data only. However, the models are scene and location-specific meaning that they use a priori knowledge about the perspective and geometry of the location. Hinterstoisser et al. (2017) super-impose 3D rendered models of toys with different lighting and poses onto real backgrounds. As opposed to Tremblay et al. (2018), the authors argue that freezing backbone layers during training on the synthetic data is better compared to letting the backbone be trainable.

Astermark (2018) generates synthetic data of faces using Generative Adversarial Networks. The author shows that while training on large real training sets yields best performance, synthetic data can improve performance if small amounts of real data are available. Harrysson (2019) trained a YOLOv3 detector on synthetic license plates super-imposed on real background images. The results showed that mixing real and synthetic data gave better performance and only using synthetic data for training almost matches the performance of training on only real data.

These papers have shown how and where synthetic data can and can not be used. However, most of the experiments are done on two-stage models such as Faster-RCNN which is not suitable for real-time detection. Moreover, a limited set of labels is often used under very specific weather and lighting conditions. None of these papers have experimented extensively on how the real training set size from 0 to 100% affects the performance when transfer learning from a synthetic model. An understanding of the actual differences between models trained on synthetic and real data on a layer-wise basis seems to be missing. To further leverage synthetic training data, a better understanding of how it is influencing the neural networks is needed. Thus, in this thesis, we will dissect these models and show quantitative results. From these results, we will try to draw conclusions about differences in the networks.

1.3.3 Similarity of Neural Networks

The fast development and research in CNNs have yielded valuable models which perform well on previously difficult tasks. These models have been used across many application areas such as computer vision and natural language processing. However, we still have little knowledge about what the networks are actually doing *semantically* in the hidden layers. In many cases, these models have been treated as black boxes (Fong and Vedaldi, 2017; Alain and Bengio, 2016).

One idea of obtaining more insight on how the network behaves is looking at the outputs layer-wise. By comparing layer outputs from two different models, one can determine the similarity between the layers. One method of measuring the similarity of layer outputs is the *singular value canonical correlation analysis* (SVCCA) (Raghu et al., 2017). SVCCA uses *singular value decomposition* (SVD) (Golub and Reinsch, 1971) for dimensionality reduction and then *canonical correlation analysis* (CCA) (Hardoon et al., 2004) which was previously used to learn semantic representations for web images. A further improvement of SVCCA is the *projection weighted CCA* (PWCCA) (Morcos et al., 2018), which uses projection weighting to calculate the similarity measure as a weighted mean instead of a naive mean as in SVCCA.

Both metrics are invariant to invertible linear transformations which according to Kornblith et al. (2019) leads to some major issues. Kornblith et al. (2019) instead proposed a metric called *centered kernel alignment* (CKA) which, according to the authors, better captures similarity representations between network layers.

While there exist several papers that attempt to answer how initialization, model complexity, or dataset size affect the similarity between models (Kornblith et al., 2019; Raghu et al., 2017; Morcos et al., 2018), no attempts have been made to compare the difference between models trained on synthetic and real data. As CKA gives a layer-wise similarity of hidden layers within the network, it can give insights of how such networks differ from each other on a layer-basis. These insights could be leveraged for example during training to target specific layers inside networks to improve performance. Therefore, in our thesis, we chose to use CKA to analyze similarity between network layers.

1.3.4 Criticality

Another approach to understand the hidden layers of a neural network is to look at the layer-wise criticality. This can either be defined as robustness to re-initialization or re-randomization and is explored in Zhang et al. (2019) and further developed in Chatterji et al. (2019). Zhang et al. (2019) investigate how much impact each layer in the network has on the performance by resetting the layer weights and evaluating the network. The mean criticality of a network is also supposedly a good measure for generalization compared to previous metrics such as generalization error, parameter counting and distance to initialization.

We use the metric re-randomization sensitivity (RR-sensitivity) inspired by the criticality measure to investigate the differences between networks trained on synthetic and real images. Moreover, we further develop the idea of re-randomization by using layer-swapping, swapping the layer weights of two models on the same layer as another measurement of similarity between network layers.

1.4 Contribution

The contribution of this thesis to the field can be summarized in the following points:

1. Show how synthetic data can increase performance for object detection and how this scales as more real data is available;
2. Show other positive side-effects of using synthetic data such as stability against hyper-parameters;
3. Give insights on how models trained on synthetic and real data differ from each other through layer-wise analysis.

1.5 Work Distribution

Arvid Mildner was responsible for most of the CKA analysis part while Tony Liu worked on the criticality and layer swapping part of the thesis. The rest of the work was carried out together, including researching previous work, building the code base, model training, hyper-parameter search, creating plots, analyzes, and report writing.

Chapter 2
Approach

2.1 Data

2.1.1 Definition of Synthetic Data

We will often refer to synthetic and real data in this thesis. Therefore, it is important to clarify the difference between them. Image data will be considered real if it is captured with an actual camera in the real world. Synthetic data is data generated by a computer and in our thesis is either one of the following:

- A rendered scene from a custom simulation tool such as Carla (Dosovitskiy et al., 2017), Synscapes (Wrenninge and Unger, 2018), or Synthia (Ros et al., 2016).
- An image captured from a video game such as *GTA V* (Richter et al., 2017; Johnson-Roberson et al., 2016).

2.1.2 Datasets

Berkeley Deep Drive

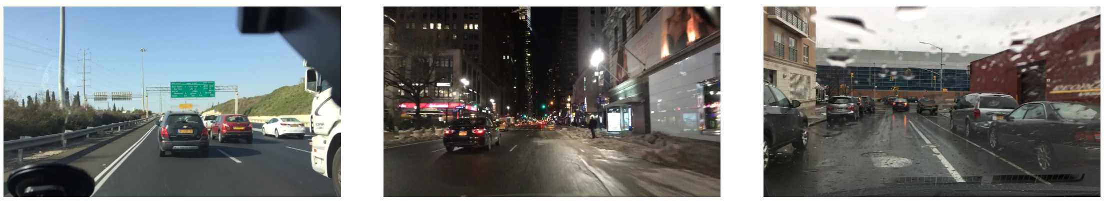


Figure 2.1: Images from the BDD dataset (Yu et al., 2018).

The Berkeley Deep Drive (BDD) dataset (Yu et al., 2018) is based on 100,000 driving videos collected from 50,000 rides mostly from dash-cams i.e. images with driver's perspective with 720p resolution. The videos are collected from New York, San Francisco, Berkeley, and the Bay Area with diverse scenes such as cities, residential areas, and highways. Moreover, the videos are recorded during different hours of the day and in different weather conditions.

One frame from each of the 100,000 videos are extracted and annotated with bounding boxes and class label. The classes in the dataset are shown in Table 2.2. Three sample images from different time of day and weather conditions are shown in Figure 2.1. 20,000 out of the 100,000 images are reserved for the test set. However, since the labels for the test set are unavailable, we use the remaining 80,000 images for our experiments divided into 60/20/20% for training, validation and testing.

Grand Theft Auto V

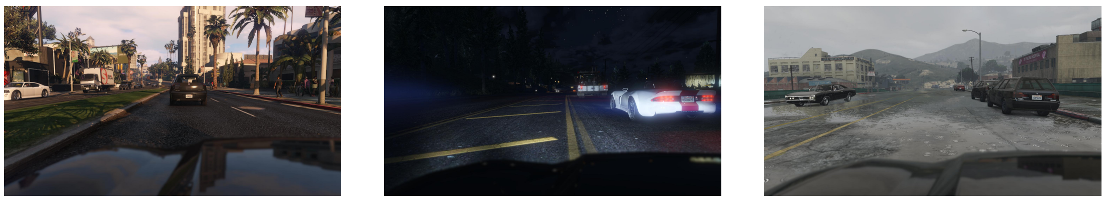


Figure 2.2: Images from the GTAV dataset (Richter et al., 2017).

The *Grand Theft Auto V (GTAV)* dataset (Richter et al., 2017) consists of 1080p images sampled from video sequences from the video game *Grand Theft Auto V*. The images and 3D object information are extracted using an injected middle-ware which receives all rendering commands. Using this information, the object information and position as well as the rendered image can be retrieved.

The training set consists of 134,000 images which are collected on different time of day, in different weather conditions in the fictional city of Los Santos. Images were divided into 60/20/20% for training, validation, and testing for the experiments. Table 2.3 shows the classes in the dataset and three sample images from different time of day and weather conditions are shown in Figure 2.2.

Since the object position information in the GTAV dataset is gathered before the actual rendering, there exist a lot of tiny bounding boxes in the ground truth that are not actually visible. These labels can be objects very far away or persons inside vehicles which makes them very hard or sometimes impossible to spot, even for a human. To eliminate these bounding boxes, we filtered out small bounding boxes with an area smaller than 100 pixels. This area was chosen by empirical visual inspection of the ground-truth bounding boxes.

Furthermore, in the GTAV dataset, the hood of the driving car is labeled while it is not labeled in the BDD dataset. Therefore, we also removed the hood labels from the dataset. This filtering of bounding boxes is visualized in Figures 2.3 and 2.4. Filtered small and hood bounding boxes are shown in red and the remaining labels are shown in green. The number at each bounding box represents the class index of the object. These images are scaled to 416×416 pixels which will be the resolution of the input to the YOLOv3 object detector.

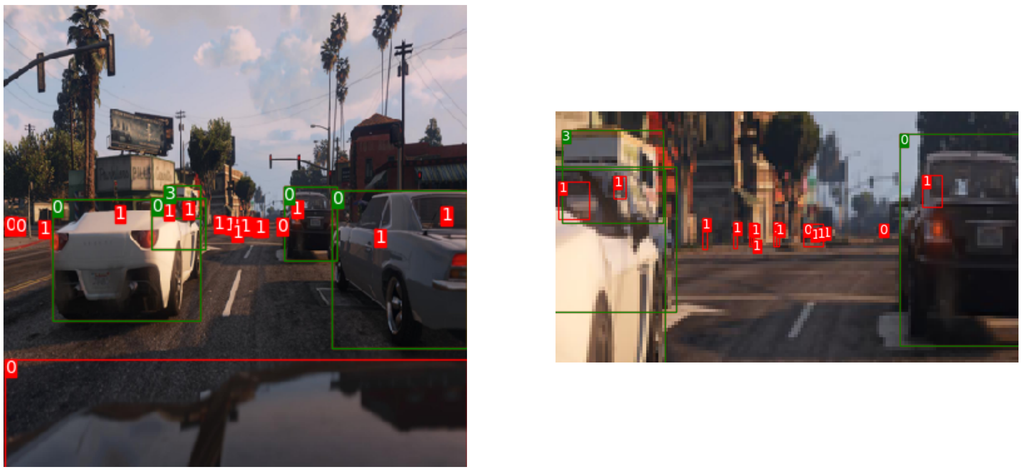


Figure 2.3: Filtered and saved bounding boxes in the GTAV dataset. Red bounding boxes are removed while green are kept. Numbers represents the class indices. For example: 0 is car, 1 is person.

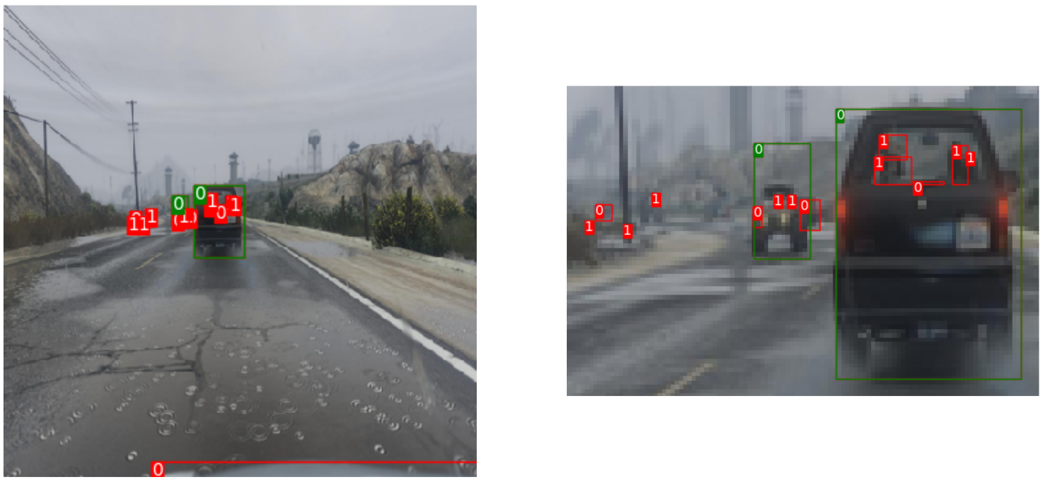


Figure 2.4: Filtered and saved bounding boxes in the GTAV dataset. Red bounding boxes are removed while green are kept. Numbers represents the class indices. For example: 0 is car, 1 is person.

2.1.3 Intersection of Class Labels

The GTAV (Richter et al., 2017) and BDD (Yu et al., 2018) datasets use different class labels. The number of classes in the label set can vary, for example GTAV has 32 classes while BDD has 10. Moreover, label names in the datasets also differ. GTAV has the *bicycle* and *motorcycle* labels, while BDD uses the names *bike* and *rider*. Problem arises when we want to train our model using one dataset and evaluate on another, or when transfer learning between models trained on different datasets. This problem could be solved at test time using a prediction mapper between the predicted labels and the target label space. However, since our model detection head is dependent on the number of classes in the dataset (see Section 2.2.2) transfer learning with YOLOv3 would not be possible using this approach.

Our approach to this problem is to define a common intersection of all class labels of the datasets with a fixed number of classes. We call this label space the *common* labels. This forces us to surjectively map some of the more detailed labels such as *van* and *trailer* into less descriptive common labels such as *car* and *truck*. The upside is that we now have a common interface between the model and the datasets which is independent on which dataset the model is trained on. The common class label space defined by us is described in Table 2.1. The surjective mappings from BDD and GTAV label indices to the common label index are shown in Tables 2.2 and 2.3 respectively.

Table 2.1: Commonly defined labels which cover an intersection of the various class names between the datasets used.

Common label index	
Index	**Label**
0	Car
1	Person
2	Cycle
3	Truck
4	Bus

Table 2.2: The surjective label map from *BDD* labels to the simplified *common* labels. Note that one prominent side effect of this definition is that rider and bike are now both considered as cycle.

BDD to common index mapping	
BDD	**Common**
traffic light	-
motor	-
train	-
truck	truck
rider	cycle
car	car
bus	bus
bike	cycle
traffic sign	-
person	person

We also made a slight mistake here which was noticed too late to change it. In the BDD dataset the label *motor* was simply dropped, while it should have been mapped to *cycle*. This happened because we did not realize it meant motorcycle. Since the number of motorcycles in BDD is small compared to the other classes and the mistake was made for all experiments, this error should have negligible effect on the general trend and conclusions of our results. We also mapped *rider* to *cycle*. Since *bike* was also mapped to *cycle*, a rider on a bike would give us 2 bounding boxes for the cycle where we would rather have wanted one bounding box. However, this will be partially compensated for by the non-maximum suppression step in the detection head. We conducted a few experiments to somewhat verify that this had little effect on our results. For further detail of these experiments, see Appendix A.

Table 2.3: The surjective label map from *GTAV* labels to the simplified *common* labels. Note that one prominent side effect of this definition is that motorcycle and bicycle are now considered the same.

GTAV to common index mapping	
GTAV	**Common**
unlabeled	-
ambiguous	-
sky	-
road	-
sidewalk	-
railtrack	-
terrain	-
tree	-
vegetation	-
building	-
infrastructure	-
fence	-
billboard	-
trafficlight	-
trafficsign	-
mobilebarrier	-
firehydrant	-
chair	-
trash	-
trashcan	-
person	person
animal	-
bicycle	cycle
motorcycle	cycle
car	car
van	car
bus	bus
truck	truck
trailer	truck
train	-
plane	-
boat	-

2.2 Object Detection Training and Evaluation

The task of object detection is to mark each object instance in an image with a 2D bounding box and classify the object correctly which is shown in Figures 2.3 and 2.4. The annotations therefore consist of bounding boxes and class labels for each object in an image. How well a model predicts these bounding boxes and class labels can be measured in numerous ways. A standard metric is using the so called mean average precision (mAP) popularized by Lin et al. (2014). To understand the mAP metric, we first recap some basic metrics used to evaluate performance.

2.2.1 Evaluation Metrics

Precision and Recall

To understand the metrics properly, we first have to introduce the following metrics known as true conditions. They can be defined as follows:

- **True postive (TP)**: Number of correctly predicted objects.
- **True negative (TN)**: Number of correctly predicted non-objects.
- **False positive (FP)**: Number of wrongly predicted objects.
- **False negative (FN)**: Number of missed objects during prediction.

The precision P is given as the ratio between true positives (TP) and the total number of predicted positives i.e. TP plus false positives (FP). It can be described as:

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \, . \tag{2.1}$$

The recall R is the ratio of TPs and the total number of ground truth positives and is given by

$$R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{2.2}$$

where FN is the false negatives. There is a trade-off between precision and recall in a classification model. Increasing one often leads to a decrease in the other. In object detection, these two metrics are combined to define the mAP for measuring the overall performance which we will come to shortly.

Intersection over Union

In object detection, the predictions and ground-truth labels are bounding boxes i.e. rectangles drawn over each object instance. The *intersection over union* (IoU) between two bounding boxes is defined as:

$$IoU = \frac{A_{\text{overlap}}}{A_{\text{union}}}, \tag{2.3}$$

where A_{overlap} is the area of the intersection and A_{union} is the union of the bounding boxes. A visual representation of this equation is shown in Figure 2.5.

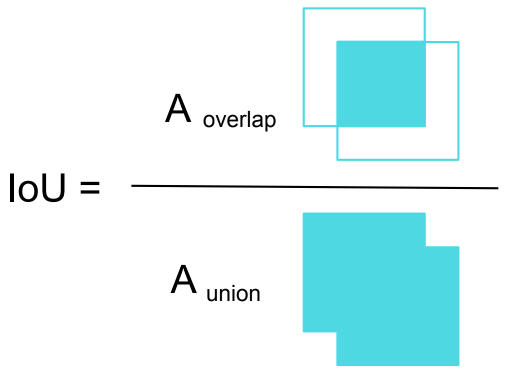


Figure 2.5: Visualization of Intersection over Union (IoU).

Mean Average Precision

With precision and recall properly defined above, we are ready to describe how we can calculate *average precision*. We pass our validation images through the network and collect the detections for all the objects. We are now able to create the precision-recall curve for each class by keeping track of the precision and recall values. This curve is created by plotting the highest scoring precision for each recall value and interpolating between the values. The average precision is then defined by the area under this precision-recall curve. When the average precision (AP) for each class has been calculated in this manner, we can finally create the mAP which is the mean AP over all the classes:

$$\text{mAP} = \frac{\sum_c^C \text{AP}(c)}{C} \tag{2.4}$$

where C is the total number of classes and AP(c) is the average precision for a given class c over the images.

Counting TP, FP and FN to calculate the precision and recall can be done at different levels of IoUs. Thus, the mAP is given at a specific IoU denoted as mAP@IoU. For our experiments, we use the IoU of 0.5. Henceforth, when referring to the mAP, we actually mean mAP@0.5.

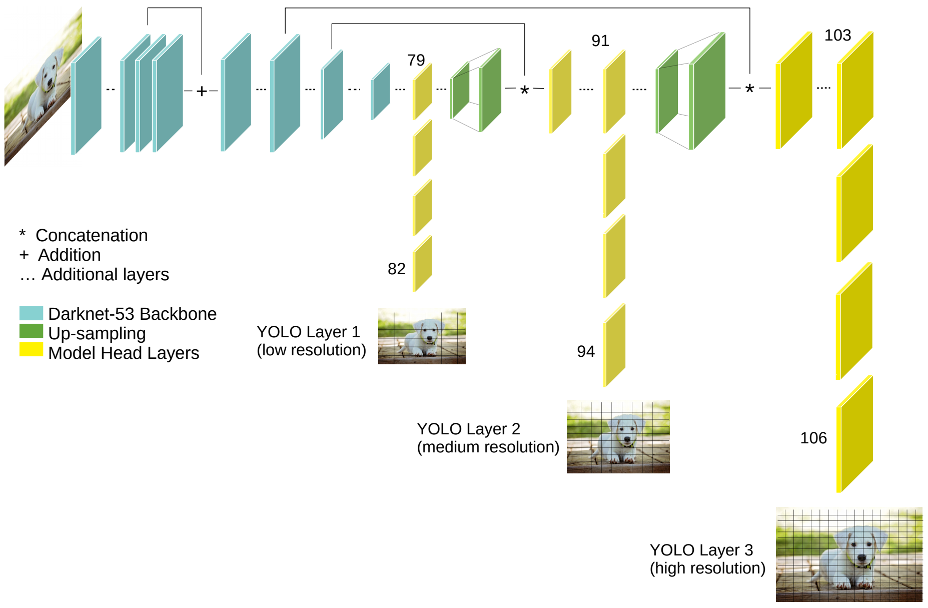


Figure 2.6: YOLOv3 architecture

2.2.2 YOLOv3

YOLOv3, *You Only Look Once version 3*, (Redmon and Farhadi, 2018) is a novel one-stage network architecture for a fast and accurate object detector building on previous detection models such as Faster-RCNN (Ren et al., 2015) and FPN (Lin et al., 2016). Compared to similar performing object detection models, YOLOv3 is significantly faster at inference due to its one-stage detection process. The model is able to predict 20 frames per second (FPS) compared to 6 FPS for Faster-RCNN, while still maintaining similar performance on the COCO dataset (Redmon and Farhadi, 2018). The high inference speed is especially attractive in a real-time detection application.

Figure 2.6 shows the network architecture. The idea is to extract features from an image using Darknet-53, a backbone built with residual blocks and convolutional layers, which down-samples along the network depth using the stride length instead of max pooling. This backbone architecture is shown in Figure 2.7. The backbone is divided into residual blocks which are marked as boxes in the figure i.e. leveraging shortcut connections similarly to ResNet backbones (He et al., 2015). The benefits of such skip connections is that they deal with vanishing gradients and at the same time encourage feature reuse, which makes the model more parameter-efficient.

The YOLOv3 network predicts bounding boxes at three resolution levels. These final prediction layers are referred to as YOLO layers and are seen in Figure 2.6 as layers 82, 94, and 106. Each prediction layer consists of a grid, where each cell contains the prediction of a bounding box, its objectness score and a classification score for each class. After the

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

Figure 2.7: Darknet-53 architecture

low resolution YOLO layer (layer 82), responsible for detecting high-level objects, the output is up-sampled and concatenated with an intermediate output from Darknet-53, which corresponds to the same up-sampled resolution. This concatenated tensor is passed through several convolutional layers and finally through the second YOLO layer. The same procedure is then repeated for the third and last prediction layer.

The YOLO layers are grids, where the cells are responsible for predicting the bounding boxes as well as containing the predicted object and class probability. In inference, bounding boxes are non-maximum suppressed according to their objectiveness score, filtering out instances which the network believes have low probability of containing objects. The remaining bounding box predictions are then used in the actual prediction of the model.

2.2.3 Training on Synthetic Data

Convolutional networks often have complex architectures with a lot of parameters to train. Therefore, these networks are often divided into two parts: a backbone responsible for feature extraction and a detection head or classifier. Since training a backbone can take several weeks, training of convolutional networks often uses pre-trained backbone weights at initialization to reduce the computations needed.

A convenient approach is to load pre-trained weights into the backbone, and only train the head of the network on the task-specific data. The idea is that the feature extraction part has already learnt how to produce generally meaningful features from images which is useful for any specific vision task. Arguments can be made that the feature extraction layers should be considered good enough and that it is actually beneficial to freeze the layers as a kind of regularization (Hinterstoisser et al., 2017). On the other hand, the feature extraction layer weights may still have room for actual improvement and further training could increase the overall performance. One obvious upside of freezing the weights is the decreased number of trainable parameters, increasing the training speed.

When training models on synthetic data for usage in the real domain, Hinterstoisser et al. (2017) showed that freezing the backbone weights (when they are initialized from a pretrained backbone) yields better performance. The authors noticed a significant performance drop if they let the backbone be trainable when training on synthetic images. The dataset consisted of pairs of images with an object in a real environment. The difference between the images in the pairs was that one image contained a real object while the other had a 3D model of the object superimposed onto the environment. They noted that the Euclidean distance between the feature vectors of these image pairs were significantly higher when using a trainable backbone compared to a frozen backbone for the model trained on the synthetic images.

It is worth noting that later work (Raghu et al., 2017; Morcos et al., 2018) has shown that the Euclidean distance is not an ideal measurement of similarity between hidden layer outputs, but it can still give some useful insights. However, Tremblay et al. (2018) show opposite results i.e. that freezing backbone layers yielded worse result. The authors argue that a possible explanation could be that the dataset used in Tremblay et al. (2018) was large and diverse enough to further improve the backbone.

2.2.4 Implementation

We built our code base on the open source implementation of YOLOv3 in PyTorch, developed by a team of engineers at Ultralytics (2019). It is a well used repository with over 1.4k forks as of this writing, backed by a well established corporation within the field. We have no reason to believe there are any major problems about this code that would severely impact the results and conclusions.

Cosine Learning Rate Decay

Adjusting the learning rate during training can greatly impact the outcome. Exponential decay of the learning rate is widely used by for example decreasing the learning rate by 0.5 every 2nd epoch. However, He et al. (2018) propose a so called cosine decay for scheduling the learning rate which has shown to give better results for training models. The learning rate η_t at epoch t is described by the following formula:

$$\eta_t = \frac{1}{2}\left(1 + \cos\frac{t\pi}{T}\right)\eta_0, \tag{2.5}$$

where η_0 is the initial learning rate and T is the total number of epochs. This learning rate decay was used in our implementation to achieve better training results. The learning rate η normalized to η_0 as a function of the epoch t normalized to T is shown in Figure 2.8. The cosine decay is compared with an example of the conventional exponential step decay.

Patience

We set a maximum of 100 epochs with a patience of 10 epochs when training our models. Patience is a way to implement early stopping, a way to counteract overtraining. The idea of patience is simple: if the validation mAP has not increased for 10 training epochs, the

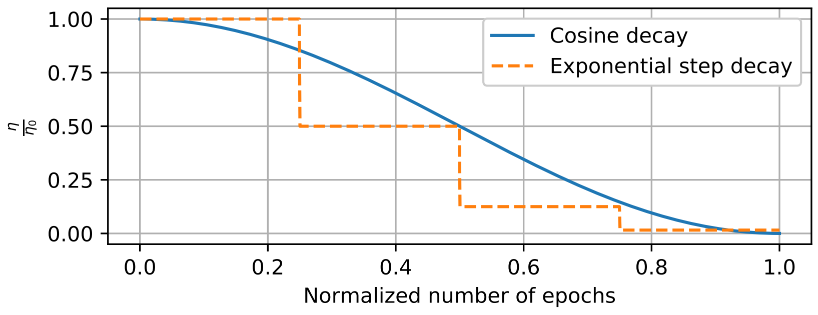


Figure 2.8: Cosine compared to exponential step decay.

training will stop early. Otherwise it will continue up to 100 epochs. The patience of 10 epochs and the total epochs of 100 is chosen based on empirical results of initial testing of the training setup.

Batch Normalization and its Implications

In the entire YOLOv3 architecture, we are applying a technique known as *batch normalization* (BN) (Ioffe and Szegedy, 2015), which is a way to severely reduce training time and increase performance of a network. The goal of BN is to counteract the problem known as *covariance shift* that occurs when training on a set of images which are not normalized to have zero mean and unit variance. While the images in the set may describe similar settings, the network has to learn not only how to produce meaningful features but also to account for the varying non-normalized input which causes an increased training time and performance loss. While it is widely known that input normalization improves accuracy in general computer vision tasks (Ioffe and Szegedy, 2015), the BN approach takes this a step further by normalizing between every layer in the feature extractor. This means that the BN layers have to learn a bit about the datasets to be able to generalize its normalization of each batch.

The BN parameters which normalize the input to each layer are updated according to a statistical approach and not through back propagation. Hinterstoisser et al. (2017) did not mention using BN in their frozen backbone, but we postulate that it is not a good idea to leave the statistical parameters of the BN layers untouched. We came to this conclusion after preliminary experiments when training on synthetic data using BN layers which were not updated after initialization and the result was a diverging loss.

According to Ioffe and Szegedy (2015), an important perk of using BN layers is that it acts as a countermeasure to the problem when a bad feature extraction in the first convolutional layer propagates and leads to worse features down the road when it is passed through the deeper layers. This is useful to us in the later experiments when we want to compare two networks trained on different datasets and maintain layers that are not dependent on the statistical distribution of the colors of the images but rather the textures and details.

2.2.5 The Base Models

To properly analyze the differently trained networks, we have conceived four base models to choose from when training or transfer learning. The goal here is to present each model so that the reader can clearly see how the various base models were created when reading the rest of the thesis.

Model A – Randomly initialized according to Kai-Ming uniform distribution in all of its convolutional layers;

Model B – Initialized with the Darknet-53 backbone which populates layers 0 to 74. Layers 75 up to layer 106 was populated randomly according to Kai-Ming uniform distribution;

Model C – Initialized as model B. Then trained on the GTAV dataset with all layers trainable for 100 epochs and a learning rate of 10^{-3};

Model D – Initialized as model B. Then trained on the GTAV dataset with only detection head trainable i.e. layer 75-106 and thus leaving the backbone untrainable. The learning rate was 10^{-3} and it was trained for 100 epochs.

2.3 Layer-wise Model Comparison

2.3.1 Similarity Metrics

Comparing the similarity between two neural networks can be done in many different ways. One approach is to look at the output for each individual layer and compare the outputs between networks. The problem can be described in the following way (Kornblith et al., 2019):

> Let $X_i \in \mathbb{R}^{p \times n}$ and $Y_i \in \mathbb{R}^{p \times n}$ be the output of layer i in form of matrices from two networks with p neurons each, fed with the same n inputs. We want to introduce a metric function $s(X_i, Y_i)$ that can be used to compare the similarity between two output matrices, to give insight of the behaviour and similarities between the hidden layers inside the models.

Several measures of similarity complying with this definition have been suggested. SVCCA (Raghu et al., 2017) and PWCCA (Morcos et al., 2018) are two examples of measuring representational similarity. Both metrics are invariant to invertible linear transforms i.e.

$$s(X, Y) = s(AX, BY)$$

for any invertible matrices A and B. This is argued to be an important property for comparing layer outputs. However, according to Kornblith et al. (2019), a metric with invariance to invertible linear transformations has the limitation of yielding the same similarity for all outputs with a greater width than the number of datapoints i.e. $p \geq n$.

The authors further argue that the scale of layer outputs also is important for representations. Therefore, similarity indices that preserve scale information, such as Euclidean distance, can be helpful on giving insights of the activations. For a metric that is invariant to invertible transforms, the magnitude of the vectors in the activation space is irrelevant and therefore ignores this important information.

Instead of requiring the similarity index to be invariant to invertible linear transform, a weaker invariance condition can be considered: invariance to orthogonal transformations. Invariance to orthogonal transformations means that $s(X, Y) = s(UX, VY)$ for any orthogonal matrices U and V. A property is that invariance to orthogonal transformations also means invariance to permutations which is important since the convolutional layer outputs should have the same representations independent of channel-wise permutations.

One such similarity index is *central kernel alignment* (CKA) (Kornblith et al., 2019). CKA is not only invariant to orthogonal transforms but also invariant to isotropic scaling i.e. $s(X, Y) = s(\alpha X, \beta Y)$ for any $\alpha, \beta \in \mathbb{R}^+$. For the matrices X and Y, the CKA with a linear kernel is defined as:

$$CKA(X, Y) = \frac{||Y^T X||_F^2}{||X^T X||_F ||Y^T Y||_F}, \tag{2.6}$$

where $|| \cdot ||_F$ is the Frobenius norm and n is the number of data points i.e. columns in X and Y. With this index definition, Kornblith et al. (2019) have shown that the CKA captures intuitive similarity ideas such as models trained in the same way with different initialization should be similar.

Convolutional Layers

While the CKA analysis requires matrices, the convolutional layers in the network output tensors. To solve this problem, we follow the line of Kornblith et al. (2019) and treat the output tensors of shape (N, X, Y, C) as a collection of vectors of the shape $(N \cdot X \cdot Y, C)$ where N is the number of images fed through the network, X and Y are the dimension of the image, and C is the number of output channels i.e. the number of convolutional kernels for the specific layer.

2.3.2 Representational Similarity

We start by training a YOLOv3 model using all of the available BDD data from our training set on top of model B as detailed in Section 2.2.5. This gives us a baseline for the performance we can obtain by the naive approach of only collecting a lot of real data.

We work under the assumption that the best possible outputs between each consecutive layer in the network is given by an image passed through this specific model. This is an important assumption which gives us a solid reference when doing layer-wise comparisons with models trained on other compositions of real/synthetic data. This approach of analysing layers from different models may also be too specific since two convolutional layers of the same layer index may have different roles between two networks trained on different data. Moreover, arguments can be made that the output of individual layers is not as important as the resulting output after a block of layers. For the sake of our analysis, we still interpret the single layer outputs as significant and leave these other interpretations to the discussion.

Our first step in the experiments is to use the CKA method described by Kornblith et al. (2019). Our aim is to analyze the similarity between convolutional layers of several models to see if the result of such an analysis can be leveraged to more intelligently use the synthetic and real data of our use case. The layer-wise similarity analysis is done by feeding 200 random images from the validation set through both of the trained networks and performing CKA of the layer outputs to find which layers seem to be similar and which are not. Since residual layers i.e. shortcut layers essentially just sum outputs from two layers without any weights, we leave these layers out from the analysis.

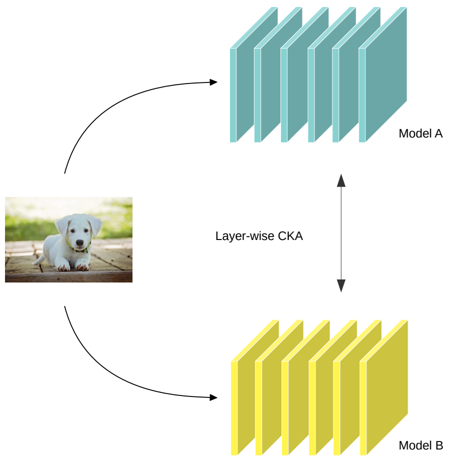


Figure 2.9: Visualization of how CKA is performed for an image. In the actual analysis, images from the BDD dataset were passed through the networks.

2.3.3 Layer Re-randomization and Swapping

In addition to studying how similar layers from different networks are, we also investigate how important individual layers are to the overall performance. One approach to studying layer importance is using layer re-initialization and re-randomization. Re-initialization looks at the performance when a layer is rewound to the initial weights while re-randomization re-samples the weights from the initialization distribution.

By re-initializing each layer and measuring performance drop, Zhang et al. (2019) suggest that layers can be categorized as either critical or ambient: critical being important and ambient being unimportant to the network's performance. Building upon this idea, Chatterji et al. (2019) propose the criticality metric by measuring robustness to re-initialization at different training epochs. This metric is somewhat complicated and requires training and saving a lot of different models. Therefore, for simplicity and to save time, we propose a simpler measurement called *re-randomization sensitivity* (RR-sensitivity) ξ_i defined as:

$$\xi_i = 1 - \frac{\text{mAP}(m_i)}{\text{mAP}(m)} \tag{2.7}$$

where m denotes the original model, m_i is the original model with re-randomized weights at layer i and mAP(m) is the mean average precision on some validation set for model m. Thus the layer importance ξ_i becomes a number between 0 and 1 where $\xi_i = 1$ corresponds to a 100% performance drop and $\xi_i = 0$ corresponds to no effect on the performance at all.

The RR-sensitivity was evaluated for various compositions of synthetic/real data. This investigation could only be done when re-randomizing convolutional layers with actual weights, leaving out shortcut layers and YOLO layers which do not contain any trainable parameters. By comparing the layer-wise importance of these models, we can hopefully see similarities and differences in the hidden layers between synthetically trained models and models trained on real data.

Another way of comparing two models is swapping the weights from the same layer between the models. Assuming that the hidden layers have similar functions in each model, the performance drop can give an indication of the similarity between the layers of the networks as well as the importance of that layer. Using the same definition of the RR-sensitivity (2.7), but subject to swapping layers instead of re-randomization, this similarity measurement could potentially complement the CKA.

Chapter 3

Evaluation

3.1 Results and Discussion

Model and Training Notation. We use the model notation described in Section 2.2.5 to simplify and avoid confusion amongst the many models tested. Additionally, to describe further training on these initial models, we use the notation *M+DataN%*, where *M* is the initial model, *Data* describes the dataset used for further training and *N* is the percentage of the dataset used. For example, model D+BDD5% denotes the initial model D trained on 5% of the BDD training set. Moreover, the performances presented as mAP in the experiments hence is more specifically mAP@0.5 i.e. mAP at 0.5 IoU as discussed in Section 2.2.1.

3.1.1 Base Models Trained on Synthetic Data

Training YOLOv3 model on the GTAV dataset with trainable and frozen backbone with learning rate of 10^{-3} and batch size 8, yielded our base models trained on synthetic data i.e. model *C* and *D*. Since the GTAV dataset is very large, training these networks takes a very long time. Therefore, we did not conduct any extensive parameter-tuning when training our base models. Furthermore, we believe that parameter-search is more important for the fine-tuning part of the training. This is because the training and evaluation images are now from the same domain and this final training is likely to affect the performance on BDD the most. The mAP of these models evaluated on the BDD and GTAV validation sets is presented in Table 3.1.

We see that model *D* has a higher mAP on the BDD validation set than model *C*. This means that freezing the backbone during training on synthetic images yields better performance on the real dataset. Figure 3.1 shows two sample images with the predictions made from these models. We can see that model *D* seems to be slightly better at predicting the correct bounding boxes. This result is in line with results from Hinterstoisser et al. (2017) but differing from results presented in Tremblay et al. (2018). As the dataset used in Hinterstoisser et al. (2017), the GTAV dataset seems to not be large and diverse enough to improve the Darknet-53 feature extractor. Whereas the inherent diversity of the domain randomized

Table 3.1: Performance of base models trained on synthetic data: model *C* and *D* on the BDD and GTAV validation sets.

Model	mAP on BDD validation set	mAP on GTAV validation set
C	0.089	0.831
D	0.114	0.823

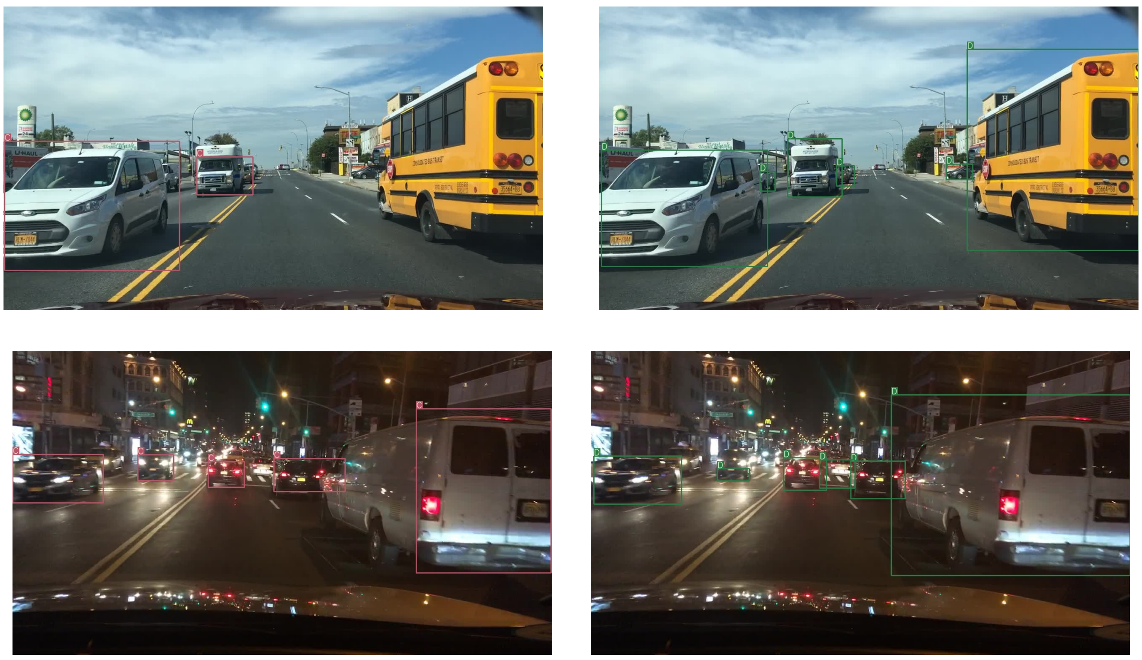


Figure 3.1: Predicted bounding boxes of two scenes from the BDD dataset. The left predictions are made by model *C* while the right predictions are made by model *D*.

dataset in Tremblay et al. (2018) could be the explanation to why they find differing results. Since model *D* performed better on the real validation set than model *C*, we use this model as our base model when fine-tuning on real data.

3.1.2 Transfer Learning on Synthetic Model

Hyper-parameter Search

Model training results are highly influenced by the chosen hyper-parameters. For example, the choice of optimizer type, number of training epochs, batch size and learning rate can yield different performances for the same model definition and training set. Therefore, we carried out a parameter search to be able to conduct a fair comparison between model performances.

We did the parameter search most extensively using 5% of the real training data due to time constraints. A subset of the parameter space was then searched for the remaining fractions of real data (10%, 25%, 50% and 100% respectively). The search was performed for training of model *B* and *D*, i.e. training a network with pre-trained backbone and transfer learning from a base model trained on synthetic data respectively. To limit the search space dimensions, we only look at optimizing for the batch size, optimizer types and initial learning rate.

The values and types tested are presented in Table 3.2. As seen in the table, this yields 24 models for training of model *B* and 24 models for transfer learning of model *D*. For all the proportions of the real training data used, when training both model *B* and *D*, the whole backbone is set to trainable i.e. all layers in the YOLOv3 model are trainable.

Table 3.2: Hyper-parameter values tested for 5% real training data.

Hyper-parameter	**Tested values**
batch size	4, 8, 16
optimizer type	ADAM, SGD
initial learning rate	10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}

The performances on the BDD validation set for model *B* trained on 5% of the real training data is shown in Figure 3.2. Here we can see that the initial learning rate of 10^{-4} using an ADAM optimizer and a batch size of 8 yielded the best performance. In Figure 3.3, the results are presented for transfer learning from model *D* trained on 5% of the real data. These results show that the best performance was reached using an initial learning rate of 10^{-4} using a SGD optimizer and a batch size of 8. These optimal values are shown in Table 3.3.

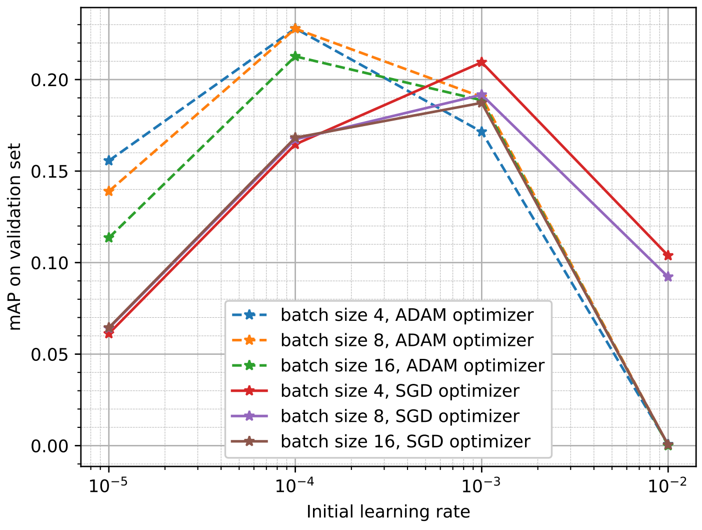


Figure 3.2: Performance on the BDD validation set as a function of the initial learning rate for different batch sizes and optimizers. Models are trained from model *B* on 5% of the real data.

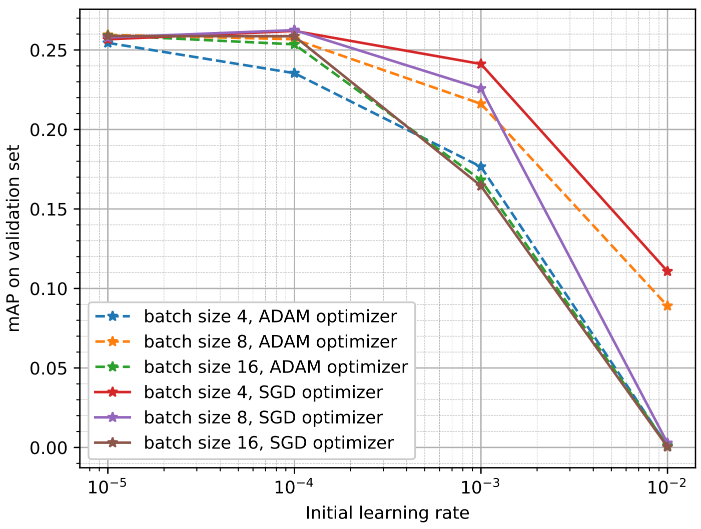


Figure 3.3: Performance on the full validation set as a function of the initial learning rate for different batch sizes and optimizers. Models are transfer learned from model D on 5% of the real data.

Table 3.3: Optimal hyper-parameter values for 5% real data training.

Hyper-parameter	**Optimal value training on model *B***	**Optimal value transfer learning on model *D***
batch size	8	8
optimizer type	ADAM	SGD
initial learning rate	10^{-4}	10^{-4}

Since the optimal values of hyper-parameters are dependent on the size of the training dataset, one would expect different optimal parameters for different proportions of real training data used. However, since training on larger training sets requires drastically more computations, the same parameter tuning could not be carried out for all proportions of real data within our time scope. Therefore parameter tuning was only carried out in the initial learning rate dimension for the proportions 10%, 25%, 50% and 100% using the optimal optimizer and batch size found in the results of the 5% training.

The performance on the validation set as a function of the initial learning rate is shown in Figures 3.4, 3.5, 3.6 and 3.7 for the proportions 10%, 25%, 50% and 100% of the real training data respectively. Performance as a function of initial learning rate is shown for both the training on model B as well as transfer learning on model D. From these figures, we determined the optimal initial learning rates for each percentage and for both training cases. These learning rates are presented in Table 3.4.

An important observation when studying Figures 3.2 to 3.7 is that the curves depicting the parameter search for Model D is consistently flatter. The flatter curves in the mAP-learning rate plot indicate that the parameter search for the fine-tuning of model D is more stable to different initial learning rates compared to training directly on model B. However, this robustness is mostly visible when lower percentages of training data is used. For example,

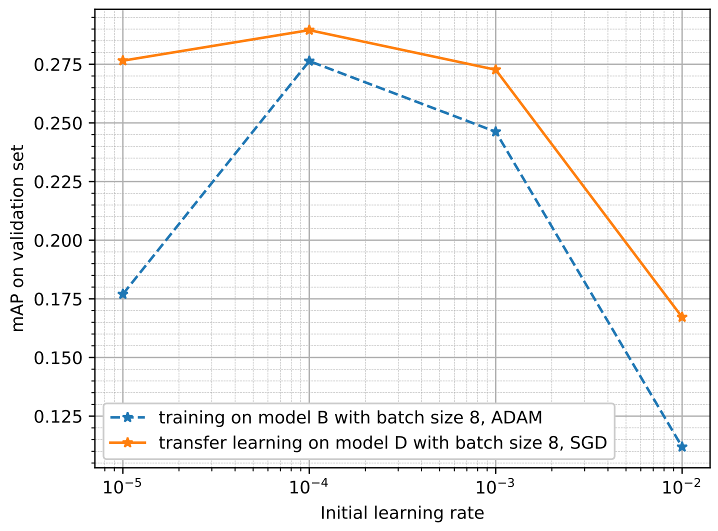


Figure 3.4: Performance on the validation set as a function of initial learning rate for different batch sizes and optimizers. Models are trained on 10% of the real data.

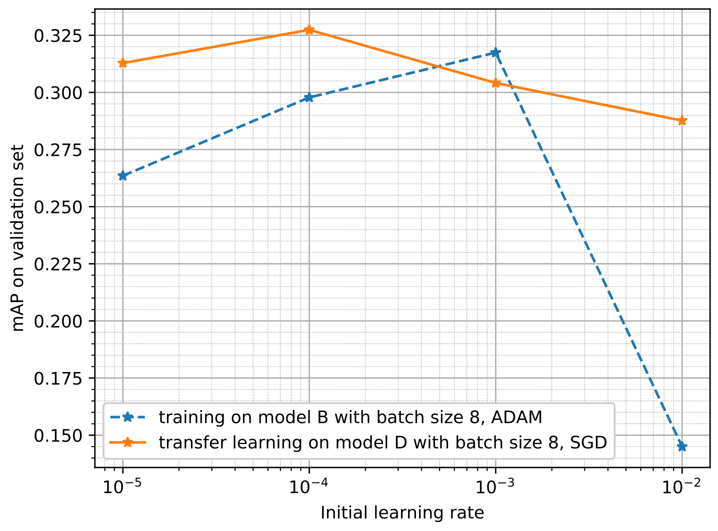


Figure 3.5: Performance on the validation set as a function of initial learning rate for different batch sizes and optimizers. Models are trained on 25% of the real data.

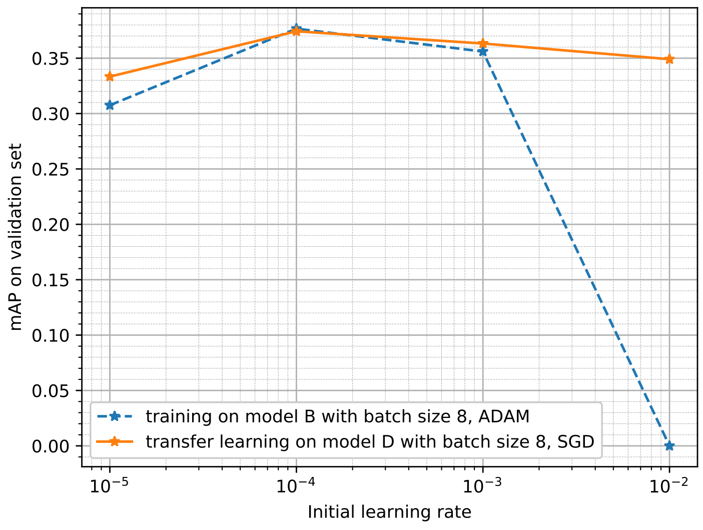


Figure 3.6: Performance on the validation set as a function of initial learning rate for different batch sizes and optimizers. Models are trained on 50% of the real data.

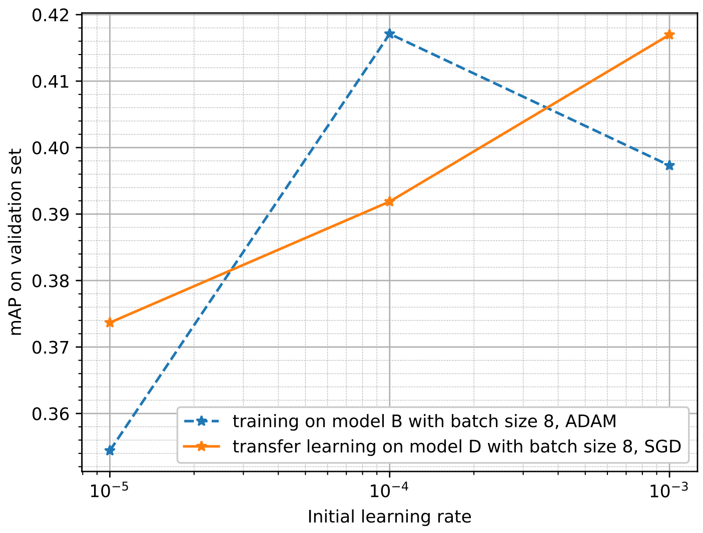


Figure 3.7: Performance on the validation set as a function of initial learning rate for different batch sizes and optimizers. Models are trained on 100% of the real data.

Table 3.4: Optimal initial learning rates for different proportions of real training data used when training on model *B* and transfer learning on model *D*.

Proportion of real training data	Optimal initial learning rate training on model *B*	Optimal initial learning rate transfer learning on model *D*
10%	10^{-4}	10^{-4}
25%	10^{-3}	10^{-4}
50%	10^{-4}	10^{-4}
100%	10^{-4}	10^{-3}

the flatness of the transfer learning curve in Figure 3.7 is not as evident as in Figure 3.2 to 3.4. Therefore, the higher robustness could be explained by the more data model *D* has trained on compared to *B*. Furthermore, model *B*'s detection head has completely random weights while the detection head of *D* has trained on the synthetic data. Nevertheless, this could be useful in a setting where a hyper-parameter search should be carried out for a model and where synthetic data is available.

Performance Comparison

Using the optimal hyper-parameters found, we compared the performance between model *B* and model *D* trained on different proportions of the real training set. The performance on the BDD test set is shown for these models as a function of percentage of the real training data used in Figure 3.8. The mAPs and relative change in mAP for the different percentages of real training data used are also presented in Table 3.5.

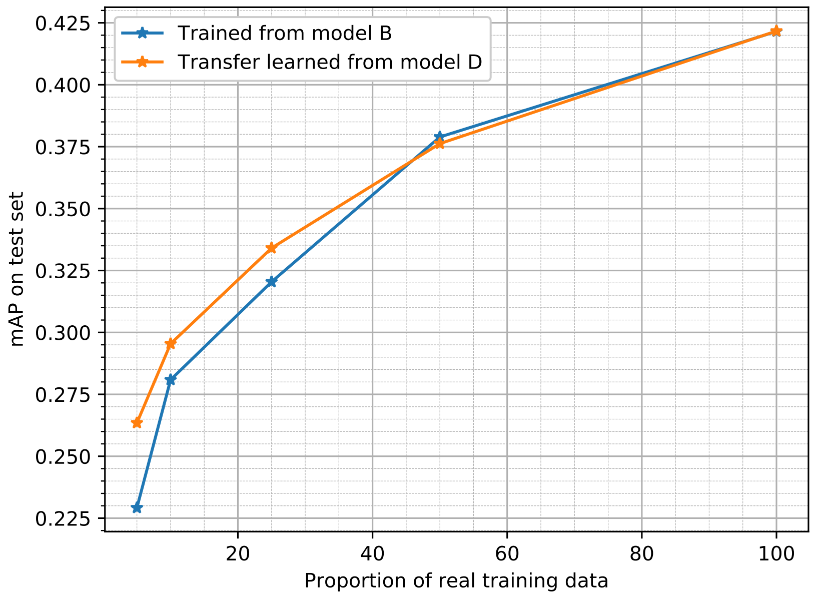


Figure 3.8: mAP on the test set as a function of percentage of real data when training model B and transfer learning model D.

Table 3.5: Optimal mAP for model B and D trained on different proportions of the real training set and the relative mAP change for each percentage level.

Proportion of real training data	**# Training images**	**Optimal mAP training on *B***	**Optimal mAP training on *D***	**Relative mAP Change** $(\text{mAP}_D - \text{mAP}_B)/\text{mAP}_B$
5%	2400	0.229	0.263	0.150
10%	4800	0.281	0.295	0.052
25%	12000	0.320	0.334	0.043
50%	24000	0.379	0.376	-0.007
100%	48000	0.422	0.422	0.000

This result shows a performance increase when using lower percentages of the real training data when fine-tuned on model *D* compared to training on model *B*. When transfer learning on 5% of the data we got an increase of 15% while 10% and 25% results in an increase of 5.2% and 4.3% respectively. For 50% we see a small decrease in performance. However this decrease of 0.7% is probably due to noise or variance in training rather than an actual decrease. Therefore, for 50% and 100%, the performance can be considered as unchanged when trained on model *B* compared to fine-tuned on model *D*.

We can conclude that using a base model trained on synthetic data is most useful when a low number of real images are available. When the real training set is sufficiently large and diverse, fine-tuning on the base model is not really useful. In our case, this region is between 25% and 50% of the real training set i.e. between 12,000 and 24,000 training images.

These results differ from Tremblay et al. (2018) and Harrysson (2019) where both show increased performance when utilizing synthetic data and the whole real dataset compared to only training on real data. One explanation could be that the BDD dataset is several times larger than the KITTI dataset used in Tremblay et al. (2018). The diversity of the data in BDD could be sufficient to achieve good performance and therefore the training on GTAV did not yield any improvements.

Another major difference in our investigation is that we use multiple classes in the dataset whereas Tremblay et al. (2018) only conducted experiments on cars and Harrysson (2019) only on license plates. Additionally, maybe other approaches of leveraging the synthetic data could be more beneficial for training on 100% of the real data. Since we train model *D* on *BDD100%*, it could be argued that the synthetic data *D* has seen could be overwritten by the introduction of real data. Maybe a method of mixing the datasets could be a better method for the higher percentages.

3.1.3 Layer-wise Analysis

CKA Similarity Analysis

From Table 3.5, we know that model *B+BDD100%* has the highest performance amongst all models tested. Therefore, the CKA was evaluated for each layer between different models and *B+BDD100%*. Using *B+BDD100%* as the reference or master model we could expect that high similarity is somewhat correlated to high performance. Furthermore, we want to observe differences in models trained on real and synthetic data. Model *B+BDD100%* is therefore used as the reference real model with which we compare other models against. In Figures 3.9 and

3.10, the results of the CKA analysis are shown when 200 images from the BDD validation set are fed through the models and compared against the output of model *B+BDD100%*.

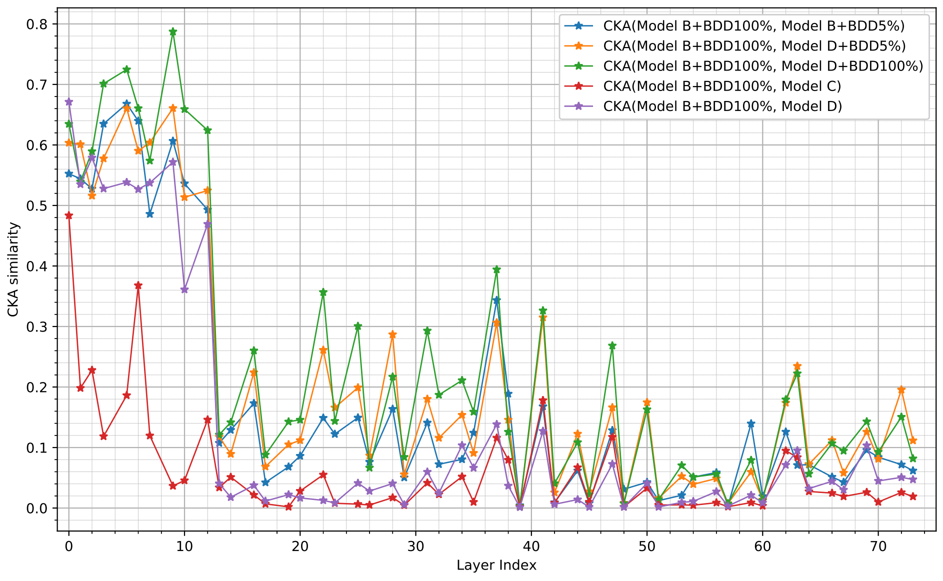


Figure 3.9: The CKA between model *B+BDD100%* and *D+BDD5%*, *D+BDD100%*, *C* and *D* in the backbone layers. Higher value means higher similarity to *B+BDD100%*.

An interesting thing to note is that the model *D+BDD5%* is more similar to *B+BDD100%* than model *B+BDD5%* in the majority of the backbone layers in Figure 3.9. Both of these models have seen equal amounts of real data during training and have similar backbones before the final training. Therefore, *B+BDD5%* and *D+BDD5%* could be expected to have equal similarity score when compared to the master model. We believe that the base model trained on synthetic data *D* has a better working detection head which makes it easier for the optimizer to focus on the backbone when training on real data. Another result from Figure 3.9 is that the similarity is low between model *B+BDD100%* and model *C* compared to model *D*. This is expected since only model C had the backbone trained with synthetic data and it is promising that the analysis can capture this.

Since model *B+BDD5%* is initialized identically to model *B+BDD100%* and only trained on less real data, we expected it to have a higher similarity throughout the network compared to the other models which are based on model *C* or *D*. This is not true for the backbone layers as *D* based models have higher similarity, but is true for the detection head where the similarity is higher than any other model, see Figure 3.10.

Looking at the CKA in the backbone layers in Figure 3.9 and at the performances achieved in Figure 3.8, we see that a high similarity to the model *B+BDD100%* seems to correlate with a high performance on the BDD test set. In the same way, low similarity with *B+BDD100%* in the backbone layers seems to correlate with lower performance.

Similarly, one could assume that a higher similarity score to *B+BDD100%* in the detection head layers would be correlated with a higher performance. However, this is not true ac-

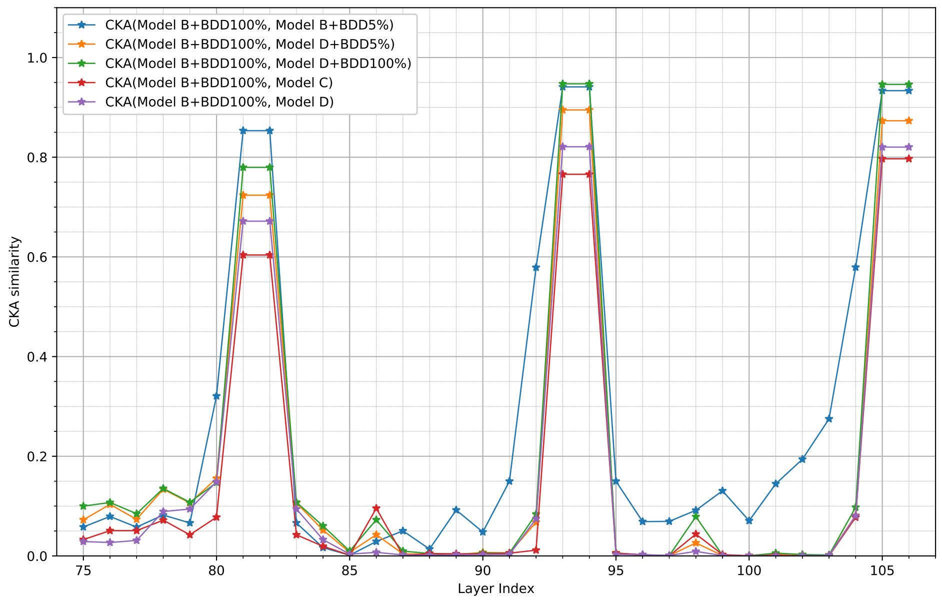


Figure 3.10: The CKA between model *B+BDD100%* and *D+BDD5%*, *D+BDD100%*, *C* and *D* in the detection head layers. Higher value means higher similarity to *B+BDD100%*.

cording to our results as we can see from Figure 3.10. The highest performing model we have using 5% of real training data, model *D+BDD5%* has a lower similarity in the detection head compared to *B+BDD5%* even though *B+BDD5%* has a lower performance on the test set (Section 3.1.2). A big difference between the detection heads of model *B* and *D* is that model *B* has completely random weights while model *D* has trained on synthetic data. This difference before the final training on BDD could be an explanation to why model *B+BDD5%* has consistently higher similarity with *B+BDD100%* in the detection head compared to the *D* based models.

We base our analysis on the idea that *B+BDD100%* is the best possible model that we can create, but we know that there are still many objects that are not found by this model. Our assumption that high similarity should imply better performance only holds in the case, where the model is predicting a subsection of the detections made by *B+BDD100%*. There may very well exist models that are not similar to this model, but have a similar or even better performance if they are able to predict labels missed by *B+BDD100%*. We argue that the transfer learned models *D+BDD5%* and *D+BDD100%* are examples of such models, where the head of the network is not converging to *B+BDD100%* but is still able to achieve high performance.

Re-randomization Sensitivity

The re-randomization sensitivity (RR-sensitivity) defined in Eq. (2.7) for model *B+BDD5%*, *D+BDD5%*, *D+BDD100%*, *C*, *D* and *B+BDD100%* was measured by looking at the performance drop on BDD validation set, where the weights in each layer are reset randomly from the same initialization distribution. This is shown for the backbone layers in Figure 3.11 and in the detection head layers in Figure 3.12 as a function of the re-randomized layer.

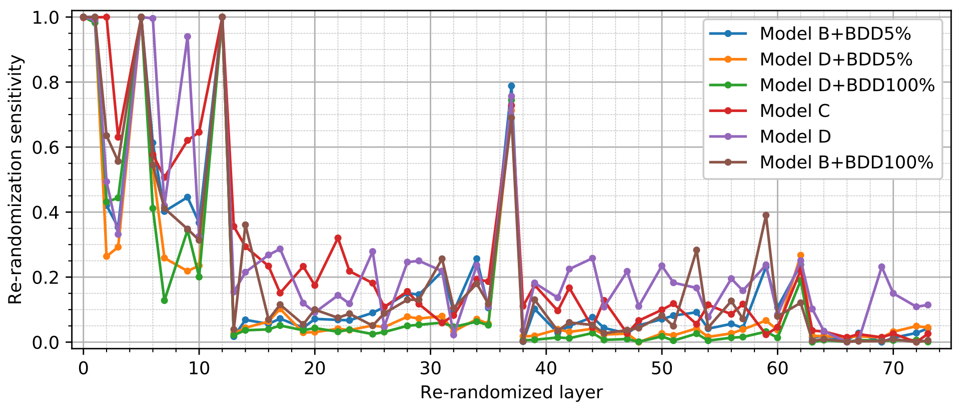


Figure 3.11: Re-randomization sensitivity of the backbone layers as a function of the re-randomized layer for model *B+BDD5%*, *D+BDD5%*, *D+BDD100%*, *C*, *D* and *B+BDD100%*. Higher value means higher sensitivity to re-randomization i.e. larger performance drop.

Looking at Figure 3.11, we can see multiple peaks in the RR-sensitivity at layers 0, 1, 5, 12 and 37 which can be explained by the residual block architecture of YOLOv3. In YOLOv3, the shortcut layers of the residual blocks are connected so that parallel routes through the network are possible. However, layers 0, 1 ,5, 12, 37, 62 are *bottleneck* layers and there are no parallel routes. This principle is shown in Figure 3.13. The left sequence represents a path where a parallel route is available. The right sequence shows a section containing a bottleneck layer.

If one of these bottleneck layers is re-randomized, the information in the layers after the bottleneck in the network will be ruined. Moreover, the layers 0, 1, 5 and 12 have a RR-sensitivity of 1 meaning that the performance drops to 0 mAP if they are reset while layers 37 and 62 have RR-sensitivities less than one. This is because layers 36 and 61 tap out into the detection head of the network. Thus, some predictions can still be made using the previous layers, even though the later layers are useless.

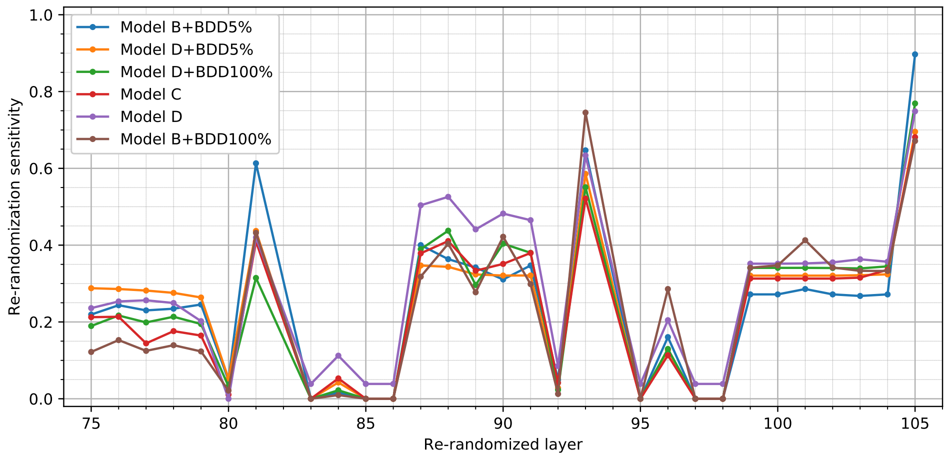


Figure 3.12: Re-randomization sensitivity of the detection head layers as a function of the re-randomized layer for model *B+BDD5%*, *D+BDD5%*, *D+BDD100%*, *C*, *D* and *B+BDD100%*. Higher value means higher sensitivity to re-randomization i.e. larger performance drop.

In the detection head shown in Figure 3.12, we can see three peaks at layers 81, 93 and 105. These layers are the layers directly before the YOLO detection layers which we refer to as pre-prediction layers. We can also see that the sensitivity of these pre-prediction layers seems to be increasing as we go deeper in the network. This suggests that high resolution predictions deeper in the network are more important for the overall performance compared to the earlier low-level predictions. Re-randomizing these pre-prediction layers (81, 93 and 105) leads to the network making random predictions at the specific detection resolution. This can be shown in Figure 3.14b for a sample BDD image where the weights in the pre-prediction layer 81 have been re-randomized.

However, at layer 80 and 92, i.e. layers before the pre-prediction layers, we see that the RR-sensitivity is very low compared to the other layers. This can be explained by the fact that layers 80 and 92 only influence the detections made at their specific resolutions whereas the layers before also affect the predictions at higher resolutions by affecting the inputs to the upsampling layers. This result indicates that resetting layer 80 or 92 i.e. disabling one of the detection heads does not really affect the performance and the predictions are still made by the remaining two detection layers. Qualitatively, this can be seen by comparing model *B+BDD100%* with no layers re-randomized in Figure 3.14a with the same model but layers 80 and 92 re-randomized in Figure 3.14c and 3.14d respectively.

A common factor for the different models is that the RR-sensitivity seems to be high at early layers in the networks around layer 0 to 13 even when disregarding bottleneck layers. This indicates that the early layers are important across all models tested. Moreover, the RR-sensitivity for D+BDD100% and D+BDD5% seems overall lower than the sensitivity of the other networks. This can be verified by calculating the mean RR-sensitivity for the networks which is shown in Table 3.6. According to Chatterji et al. (2019), a lower mean criticality corresponds to high generalization of a network. Since our RR-sensitivity measure is highly related to their criticality metric, a lower RR-sensitivity could also suggest a higher network generalization. This means that the lower mean sensitivity of the transfer learned

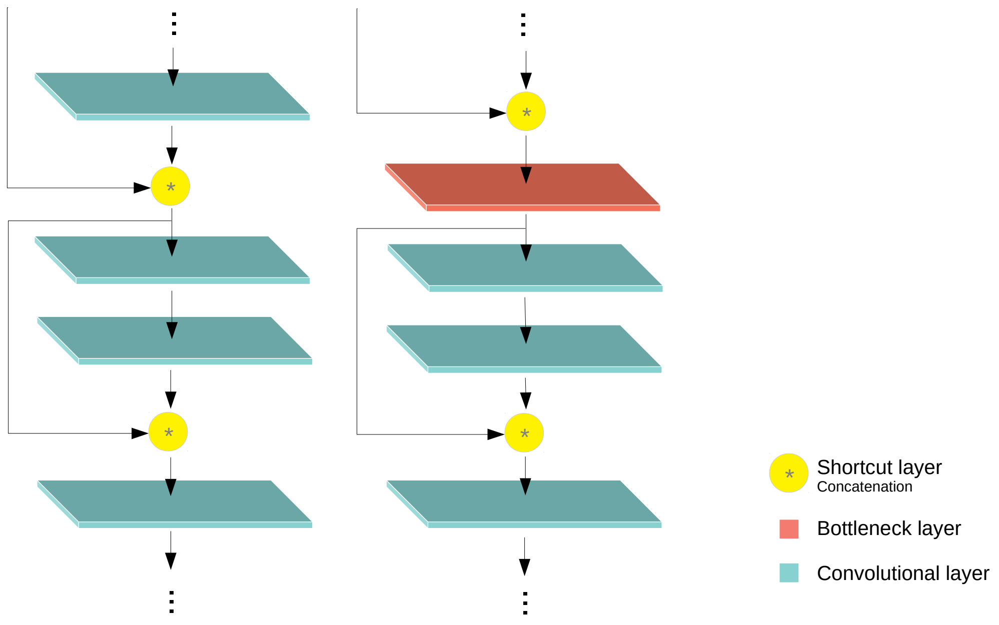
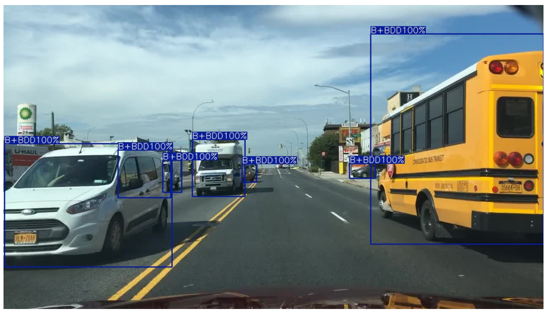


Figure 3.13: Principle of a *bottleneck* layer in the residual blocks of YOLOv3.

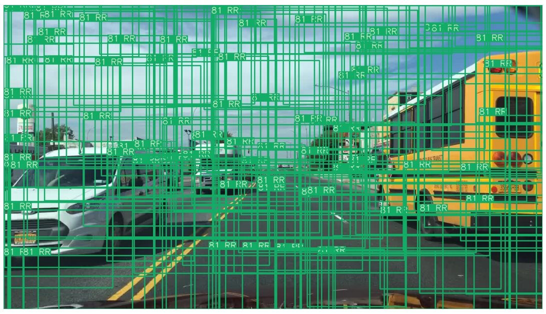
models suggests that transfer learning on a base model trained on synthetic data gives better generalization than directly training on real data.

Table 3.6: Mean RR-sensitivity for the different models.

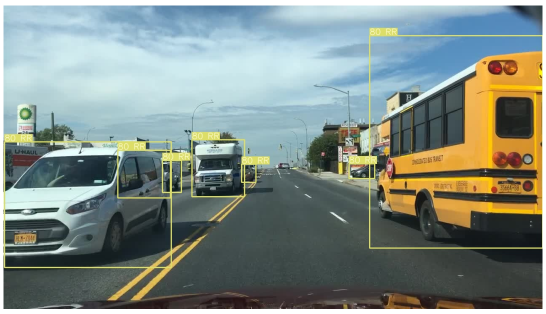
Model	**Mean RR-sensitivity**
B+BDD5%	0.202
D+BDD5%	0.160
D+BDD100%	0.150
C	0.262
D	0.282
B+BDD100%	0.214



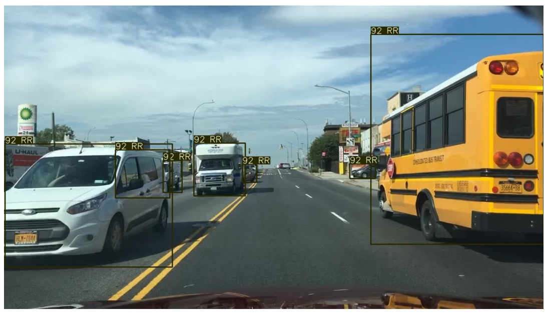
(a) Prediction made by model B+BDD100% without any re-randomized layer.



(b) Prediction made by model B+BDD100% with the pre-prediction layer 81 re-randomized.



(c) Prediction made by model B+BDD100% with layer 80 i.e. layer before the pre-prediction layer re-randomized.



(d) Prediction made by model B+BDD100% with layer 92 i.e. layer before the pre-prediction layer re-randomized.

Figure 3.14: Predicted bounding-boxes by the networks with different layers re-randomized.

Swap Sensitivity

The swap sensitivity of model B+BDD100% with weights swapped with model B+BDD5%, D+BDD5%, D+BDD100%, C, D is shown in Figures 3.15 and 3.16 by evaluating the model after swapping layers on the BDD validation set.

Figure 3.15 shows that swapping the first layers of model *B+BDD100%* with layers from model *C* results in a high sensitivity i.e. large performance drop. For the rest of the backbone we see a generally low sensitivity which indicates that swapping layers has low impact on the performance of the models. There are some peaks in sensitivity that are interesting to study, especially together with the previous RR-sensitivity analysis. Looking at Figure 3.11, we can see that there are peaks present at 0, 5, 11 and 37 where all models experience an equal drop in performance when these layers are re-randomized. Peaks at 11 and 37 are also present in the swap sensitivity in Figure 3.15, but with different magnitudes in performance drop for the different models. Swapping *B+BDD100%* with layers from model *D+BDD100%* yielded the smallest performance drop at these peak layers while swapping with model *C* resulted in the largest sensitivity and performance drop.

For the detection head layers shown in Figure 3.16, we see a pattern indicating that swapping layers for the low resolution predictions in the range from 75-82 has little impact on the model performance. This coincides with the results from the RR-sensitivity analysis that the higher resolution detection outputs are more important than the low resolution outputs.

Comparing Figures 3.16 and 3.12, swapping layers from *D* based models around layer 90 gives worse performance than re-randomizing the layer. This implies that these layers are fundamentally different and do not encode the predictions in the same representational way. This result is coherent with the CKA similarity analysis which also suggests that the models except *B+BDD5%* are fundamentally different from *B+BDD100%* in the detection head.

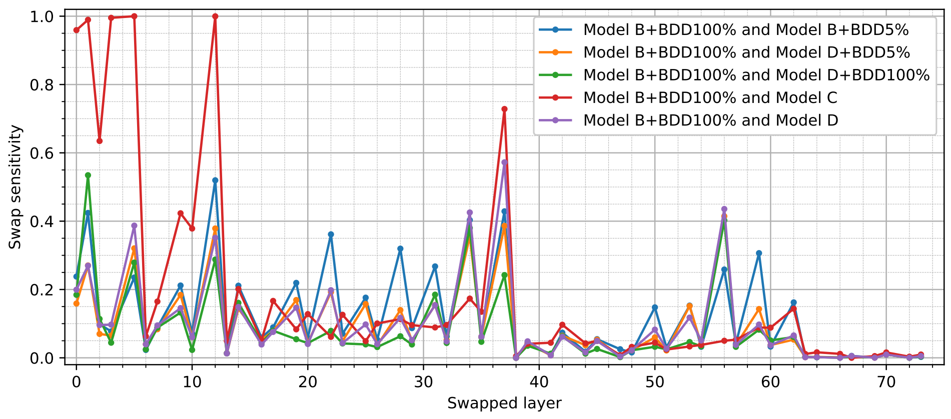


Figure 3.15: Swap sensitivity of the backbone layers as a function of the re-randomized layer for model B+BDD5%, D+BDD5%, D+BDD100%, C, D and B+BDD100%. Higher value means higher swap sensitivity i.e. larger performance drop.

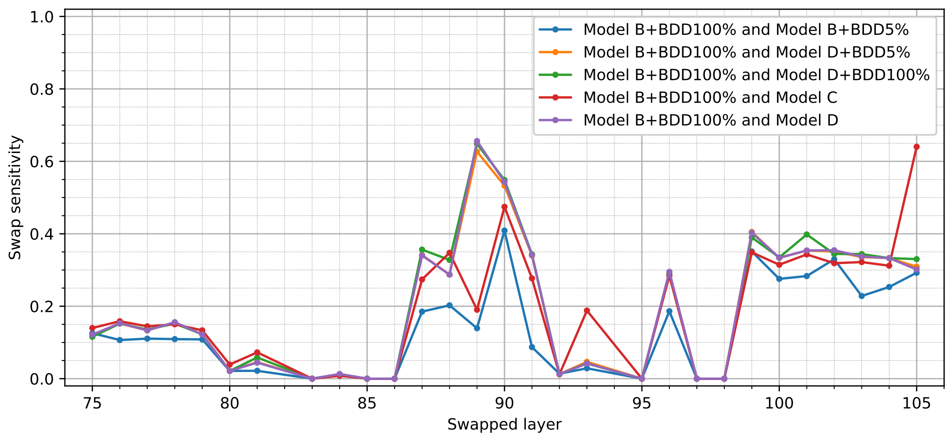


Figure 3.16: Swap sensitivity of the detection head layers as a function of the re-randomized layer for model B+BDD5%, D+BDD5%, D+BDD100%, C, D and B+BDD100%. Higher value means higher swap sensitivity i.e. larger performance drop.

Chapter 4
Conclusions

4.1 Transfer Learning on Synthetic Model

From the extensive parameter search and our experiments with transfer learning, we have been able to establish a solid baseline for utilizing synthetic data together with real data to achieve good performance. We show that there are indeed benefits of introducing synthetic data when limited real data is available. However, the effect of using synthetic data seems to render diminishing returns as more real data is available. Combining both the synthetic data and real data did not result in a model that could outperform a model trained on only real data. This means that it may be worth considering a synthetic dataset when annotated real data is hard to acquire. More sophisticated approaches to leverage synthetic data would be required to surpass state-of-the-art performance of a model trained on a highly diverse and extensive dataset such as BDD.

Another positive effect of transfer learning on real data from a base model trained on synthetic data is the increased robustness for hyper-parameters. A base training using synthetic data before hyper-parameter tuning and training on real data could significantly reduce the search time by being able to have a greater step length in the parameter space. Our experiments indicate that exposure to synthetic data may lead to a reduction of the complexity of the optimization landscape and a model with higher generalization.

4.2 Layer-wise Analysis

In the second part of the thesis, we tried to introduce new ways to think about and experiment with the internal convolutional layers of the models to shed light on the differences between models trained differently on the synthetic and real datasets. We had some success in finding promising patterns when using the CKA analysis to study the difference between the layers of the networks. We quickly realized that a complex model such as YOLOv3 introduces several other sources of difference in the layers such as skip connections which occlude some of the conclusions that can be drawn from this type of analysis.

Nonetheless, we conclude that there seems to be several different weight configurations that the models can converge to that all result in good performance. This result is most apparent in the detection head of the network. The backbone seemed to be more hesitant to find a completely different weight configuration in its layers, even if the head of the network was completely different between the models.

In the final experiments, we tried two approaches to study the behaviour of the models when we re-randomize and swap the layers of the models and measure the resulting performance drop. While these sensitivity ideas are quite simple, the results were surprisingly hard to analyze because of the complexity of the YOLOv3 model architecture.

Nonetheless, one conclusion is that training on real data on a base model trained on synthetic data seemed to yield a lower mean sensitivity to re-randomization compared to a model only trained on real data. This indicates that these models are better generalized i.e. have less overfitted parameters compared to models trained on only real data. The swapping sensitivity experiments also verified the CKA analysis conclusions regarding the layers in the head of the real and synthetic models.

When comparing models trained on synthetic and real data and models trained on only real data, layers in the head were fundamentally different. The output of each layer was representationally different but the predictions were similar. Therefore, the networks use different weight configurations taking different routes to make the same predictions. Meanwhile, the backbone layers remain relatively similar representationally.

In the end, we had hoped that the differences between a model trained on synthetic and real data would be more visible in the various layer-wise experiments that we conducted. Instead, we found that the analysis captures more of the intrinsic architectural design of YOLOv3 rather than the major differences between the training data we used.

We also had indications that in our use case, the predictions of the lower resolution output is not that important. The important predictions seem to come from the more detailed and high resolution outputs. This is a valuable observation for a system, where we want to introduce synthetic data. It implies that detail in the image is indeed important and worth spending time on when creating the synthetic dataset.

4.3 Future Work

In this thesis, we have investigated the effect of synthetic data on the YOLOv3 architecture from a performance perspective. While the parameters and approach that we applied for this specific architecture and datasets have rendered good results, it would be interesting to see what can be found when applying this new knowledge on another architecture of similar or lower complexity.

The understanding of the mathematical properties of the internal workings and hidden layers of complex neural network architectures that we have developed in this thesis is indeed promising. While some conclusions can be drawn from the analysis presented, more extensive experiments with other architectures and datasets would be interesting to further reinforce and generalize our findings.

Another interesting study would be parameter tuning on the synthetic model *D*, similarly to what we conducted for the fine-tuning. This could also impact the overall result since in this thesis, we are not using the optimal base models, model *C* and *D*.

References

Alain, G. and Bengio, Y. (2016). Understanding intermediate layers using linear classifier probes. In *ICLR 2017 workshop*.

Amazon (2005). Amazon mechanical turk.

Astermark, J. (2018). Synthesizing training data for object detection using generative adversarial networks. Master's Thesis.

Cabon, Y., Murray, N., and Humenberger, M. (2020). Virtual kitti 2.

Chatterji, N. S., Neyshabur, B., and Sedghi, H. (2019). The intriguing role of module criticality in the generalization of deep networks.

Chen, X., Kundu, K., Zhang, Z., Ma, H., Fidler, S., and Urtasun, R. (2016). Monocular 3d object detection for autonomous driving. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*.

Dosovitskiy, A., Ros, G., Codevilla, F., López, A., and Koltun, V. (2017). CARLA: an open urban driving simulator. *CoRR*, abs/1711.03938.

European Commission (2018). 2018 reform of eu data protection rules.

Fong, R. C. and Vedaldi, A. (2017). Interpretable explanations of black boxes by meaningful perturbation. In *The IEEE International Conference on Computer Vision (ICCV)*.

Gaidon, A., Wang, Q., Cabon, Y., and Vig, E. (2016). Virtual worlds as proxy for multi-object tracking analysis. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 4340–4349.

Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. (2013). Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*.

Girshick, R. B., Donahue, J., Darrell, T., and Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524.

Golub, G. H. and Reinsch, C. (1971). *Singular Value Decomposition and Least Squares Solutions*, pages 134–151. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hardoon, D. R., Szedmak, S., and Shawe-Taylor, J. (2004). Canonical correlation analysis: An overview with application to learning methods. *Neural Computation*, 16(12):2639–2664.

Harrysson, O. (2019). License plate detection utilizing synthetic data from superimposition. Master's Thesis.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. B. (2017). Mask R-CNN. *CoRR*, abs/1703.06870.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., and Li, M. (2018). Bag of tricks for image classification with convolutional neural networks. *CoRR*, abs/1812.01187.

Hinterstoisser, S., Lepetit, V., Wohlhart, P., and Konolige, K. (2017). On pre-trained image features and synthetic images for deep learning. *CoRR*, abs/1710.10710.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Jaeger, P. F., Kohl, S. A. A., Bickelhaupt, S., Isensee, F., Kuder, T. A., Schlemmer, H., and Maier-Hein, K. H. (2018). Retina u-net: Embarrassingly simple exploitation of segmentation supervision for medical object detection. *CoRR*, abs/1811.08661.

Jiang, H. and Learned-Miller, E. G. (2016). Face detection with the faster R-CNN. *CoRR*, abs/1606.03473.

Johnson-Roberson, M., Barto, C., Mehta, R., Sridhar, S. N., and Vasudevan, R. (2016). Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? *CoRR*, abs/1610.01983.

Kornblith, S., Norouzi, M., Lee, H., and Hinton, G. E. (2019). Similarity of neural network representations revisited. *CoRR*, abs/1905.00414.

Lee, N., Weng, X., Boddeti, V. N., Zhang, Y., Beainy, F., Kitani, K. M., and Kanade, T. (2016). Visual compiler: Synthesizing a scene-specific pedestrian detector and pose estimator. *CoRR*, abs/1612.05234.

Lin, T., Dollár, P., Girshick, R. B., He, K., Hariharan, B., and Belongie, S. J. (2016). Feature pyramid networks for object detection. *CoRR*, abs/1612.03144.

Lin, T., Goyal, P., Girshick, R. B., He, K., and Dollár, P. (2017). Focal loss for dense object detection. *CoRR*, abs/1708.02002.

Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C., and Berg, A. C. (2015). SSD: single shot multibox detector. *CoRR*, abs/1512.02325.

Long, J., Shelhamer, E., and Darrell, T. (2014). Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*.

Morcos, A., Raghu, M., and Bengio, S. (2018). Insights on representational similarity in neural networks with canonical correlation. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 5727–5736. Curran Associates, Inc.

Nowruzi, F. E., Kapoor, P., Kolhatkar, D., Hassanat, F. A., Laganière, R., and Rebut, J. (2019). How much real data do we actually need: Analyzing object detection performance using synthetic and real data. *CoRR*, abs/1907.07061.

Odena, A. (2016). Semi-supervised learning with generative adversarial networks. In *Proceedings of the Data Efficient Machine Learning workshop at ICML*.

Raghu, M., Gilmer, J., Yosinski, J., and Sohl-Dickstein, J. (2017). Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability.

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

Redmon, J. and Farhadi, A. (2016). YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242.

Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv*.

Ren, S., He, K., Girshick, R. B., and Sun, J. (2015). Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497.

Richter, S. R., Hayder, Z., and Koltun, V. (2017). Playing for benchmarks. *CoRR*, abs/1709.07322.

Ros, G., Sellart, L., Materzynska, J., Vázquez, D., and López, A. (2016). The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Porceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition*, pages 3234–3243.

Tremblay, J., Prakash, A., Acuna, D., Brophy, M., Jampani, V., Anil, C., To, T., Cameracci, E., Boochoon, S., and Birchfield, S. (2018). Training deep networks with synthetic data: Bridging the reality gap by domain randomization.

Ultralytics (2019). Ultralytics implementation of yolov3. `https://github.com/ultralytics/yolov3`.

Wrenninge, M. and Unger, J. (2018). Synscapes: A photorealistic synthetic dataset for street scene parsing. *CoRR*, abs/1810.08705.

Yeh, C., Lin, C., Muchtar, K., Lai, H., and Sun, M. (2017). Three-pronged compensation and hysteresis thresholding for moving object detection in real-time video surveillance. *IEEE Transactions on Industrial Electronics*, 64(6):4945–4955.

Yu, F., Xian, W., Chen, Y., Liu, F., Liao, M., Madhavan, V., and Darrell, T. (2018). Bdd100k: A diverse driving video database with scalable annotation tooling.

Zhang, C., Bengio, S., and Singer, Y. (2019). Are all layers created equal?

Appendices

Appendix A
Effect of the Label Mapping

As mentioned in Section 2.1.3 we made a slight mistake when creating the common labels for the BDD dataset. As a result, the synthetic base models could recognize motorcycles but the labels for motorcycles was not present when evaluating on BDD. Another problem was that we mapped the label *rider* to *cycle*, while in GTAV the riders are labeled as *person*. To study whether these choices have any significant effect on our results, we evaluated the models C and D when we did use the label *motor* mapped to *cycle* and changed mapped *rider* to *person* instead of *cycle*. The result is presented in the following table:

Table A.1: Performance of base models trained on synthetic data: model *C* and *D* on the BDD validation sets when the labels was corrected.

Model	**mAP on BDD valid. set with adjusted labels**	**mAP on BDD valid. set with original labels**	**Relative mAP change**
C	0.092	0.089	+3.4%
D	0.120	0.114	+5.3%

As we can see, the performance of both the models increased slightly. However, the conclusion from Section 3.1.1 that model D was the best performing is not changed. This slight increase in performance is expected since the base models trained on synthetic data have already learned to detect motorcycles as well as the riders as *person*. Before the label mapping adjustment, these motorcycle predictions became FP when they actually were correct and the riders were predicted with wrong labels.

To investigate the impact on the fine-tuning experiments, we trained model *B* and *D* on the dataset with adjusted labels with 5 and 10% of the real training data. The resulting performance on the BDD test set is shown in Table A.2. The models were trained with the optimal hyper-parameters found in Section 3.1.2

We see a small drop in performance for both B-based and D-based models when using the adjusted labels compared to using the original label mapping. The performance drop could be explained by the new label *motor* being relatively difficult to predict. Another effect is that

Table A.2: Performance of model *B+BDD5%* and *D+BDD5%* trained on BDD with adjusted labels.

Model	**mAP on BDD test set with adjusted labels**	**mAP on BDD test set with original labels**	**Relative mAP change**
B+BDD5%	0.228	0.229	-0.47%
D+BDD5%	0.250	0.263	-2.06%
B+BDD10%	0.278	0.281	-1.07%
D+BDD10%	0.294	0.295	-0.34%

remapping the *rider* bounding boxes to *person* means that each the *riders* are learned as *person* instead. Since a *person* riding a bike looks different to a walking person, this could also be a factor for the performance decrease. Nonetheless, The performance change is really small and could be insignificant. Therefore, we can conclude that this change in label mapping does not really affect our original results.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2020-06-11

EXAMENSARBETE The Effect of Synthetic Data in Training of Deep Neural Networks for Object Detection in Cityscapes
STUDENTER Arvid Mildner, Tony Liu
HANDLEDARE Pierre Nugues (LTH); Martin Ljungqvist, Otto Nordander (Axis Communications)
EXAMINATOR Jörn Janneck (LTH)

Träning av objektdetektor med hjälp av syntetiska bilder i stadsmiljö

POPULÄRVETENSKAPLIG SAMMANFATTNING **Arvid Mildner, Tony Liu**

AI-modeller som används för objektdetektion är i behov av stora mängder exempelbilder på objekt såsom bilar och personer. Det är dyrt att både samla in och annotera dessa bilder. Därför skulle ett system som kan tränas på datorgenererade bilder kraftigt minska kostnaden för att träna modeller i nya miljöer.

Man skulle kunna tänka sig att det inte är så stor skillnad på en riktig bild och en detaljerad syntetisk bild. För det mänskliga ögat verkar det i alla fall inte svårare att urskilja en bil i ett datorspel än vad det är om man tittar på en riktig bild. Den typ av modell som vi har experimenterat med är ett så kallat neuralt nätverk som på en konceptuell nivå fungerar ungefär som man tror att vår egen hjärna behandlar synintryck.

(a) Riktig bild från Berkeley Deep Drive.

(b) Syntetisk bild från datorspelet GTA V.

Tyvärr visar det sig att det inte är så enkelt - en modell som enbart har sett syntetiska bilder har svårt att få god precision när den tillämpas på riktiga bilder. Vår idé var att försöka förstå detta och se om vi istället kunde göra något för att använda så få riktiga bilder som möjligt tillsammans med de syntetiska bilderna. Först experimenterade vi med något som kallas för *transfer learning* för att nå en så god precision som möjligt given en viss mängd riktig data. Grundidén med den här metoden är att ett nätverk som enbart har sett syntetisk data behöver en liten skjuts i rätt riktning genom att få se riktig data mot slutet av träningen. Då ska den ha lärt sig i stora drag vad den ska söka efter av den första träningen med syntetisk data. Här visade vi att syntetisk data kan vara mycket värdefull om man har få riktiga bilder men har avtagande effekt på precisionen när en större mängd riktig data introduceras.

Därefter försökte vi dyka in i nätverkets inre struktur för att hitta vad som egentligen skiljer en modell som tränats med hjälp av syntetisk data och en som tränats med riktig data. Här presenterar vi olika matematiska metoder för att studera likheten mellan de interna representationerna i nätverket som leder fram till de prediktioner som görs. En allmän uppfattning inom forskningsområdet är att de olika interna lagrena i nätverket får olika uppgifter som sträcker sig från att detektera konturer till att detektera faktiska saker som hjul och bilfönster. Detta gör att nätverket slutligen kan hitta hela bilen allt eftersom den ursprungliga bilden filtreras genom hela nätverkskroppen. Idén var att vi skulle kunna isolera delar av nätverket som verkar göra ungefär samma sak oberoende av om de enbart sett syntetisk data eller riktig data.

Det visade sig vara svårare än vi trodde att tyda något anmärkningsvärt från den här analysen. Mycket av de slutsatser vi kan dra hamnar lite i skymundan bakom detaljer som har att göra med modellens arkitektur snarare än det faktum att modellerna sett olika typer av data. Trots detta ger vår analys indikationer på att en modell som tränats på både syntetisk och riktig data kan hitta nya objektdetektioner och generaliserar bättre till nya miljöer jämfört med en modell tränad på enbart riktig data. Resultatet av detta är en sorts granulär lupp som skulle kunna användas vid vidare forskning på hur ett nätverk kan bli mer effektivt med hjälp av syntetisk data.