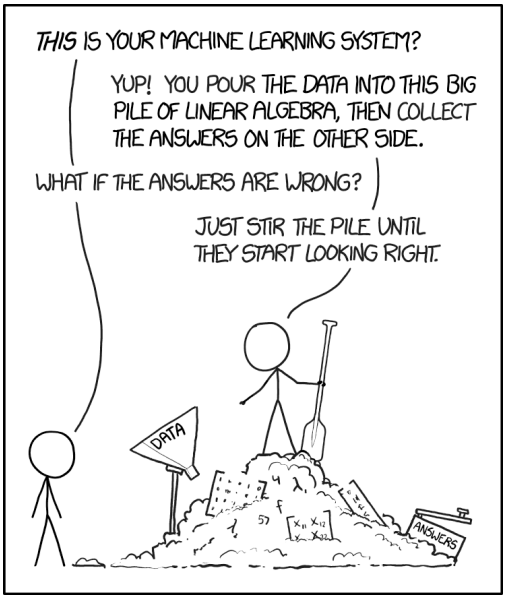
CHALMERS
UNIVERSITY OF TECHNOLOGY



Predicting Impact of Training Data for Pedestrian Detection in Autonomous Vehicles Using Influence Functions

Master's Thesis in Complex Adaptive Systems

BRITTA THÖRNBLOM

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis EX052/2018

Predicting Impact of Training Data for Pedestrian Detection in Autonomous Vehicles Using Influence Functions

BRITTA THÖRNBLOM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Mechatronics Group
Chalmers University of Technology
Gothenburg, Sweden 2018

Predicting Impact of Training Data for Pedestrian Detection in Autonomous Vehicles Using Influence Functions BRITTA THÖRNBLOM

Supervisor: Arian Ranjbar, Department of Electrical Engineering
Supervisor: Nasser Mohammadiha, Zenuity
Examiner: Jonas Fredriksson, Department of Electrical Engineering

Master's Thesis EX052/2018
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Web comic ridiculing the black-box property of machine learning algorithms.[15]

Typeset in LaTeX
Printed by [Name of printing company]
Gothenburg, Sweden 2018

Predicting Impact of Training Data
for Pedestrian Detection in Autonomous Vehicles
Using Influence Functions
BRITTA THÖRNBLOM
Department of Electrical Engineering
Chalmers University of Technology

Abstract

With the widespread use of neural networks comes a need to illuminate their inner workings. In applications such as autonomous vehicles where human safety is involved, accountability and transparency is crucial not only for the sake of user trust but can also be a tool for developers seeking to increase the robustness of the systems. To this end, influence functions from robust statistics has been presented as a tool of predicting what impact training data has on a given classification. Earlier works using influence functions have showed correlation between these predictions and the loss after retraining the model without the given training image. These experiments were done on data sets of images much smaller than those processed in the autonomous vehicle industry. The purpose of this thesis was to investigate if influence functions could be used in the same way on a binary pedestrian detection model more similar to those used in real-life applications. The results show no correlation like those found in earlier works, but still has an apparent internal coherency. In total, this points to the fact that the increased complexity of the model puts constraints on the algorithm that needs to be met in future research.

Keywords: autonomous vehicles, convolutional neural network, classification, predictions, influence functions.

Acknowledgements

I would like to thank my supervisors Arian Ranjbar and Nasser Mohammadiha and my examiner Jonas Fredriksson for all their help and support throughout the project.

Furthermore, a big thanks to Zenuity's Deep Learning group, and especially Christofer Cincinnati for helpful discussions regarding Tensorflow as well as general encouragement.

Britta Thörnblom, Gothenburg, June 2018

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
1.1 Motivation . . . 2
1.2 Goals . . . 2
1.3 Contributions . . . 2
1.4 Limitations . . . 3
1.5 Outline of the thesis . . . 3

2 Theory **5**
2.1 Convolutional Neural Networks . . . 5
2.1.1 Layer types . . . 5
2.1.1.1 Convolutional layer . . . 5
2.1.1.2 Max pooling layer . . . 6
2.1.1.3 Softmax layer and cross entropy . . . 7
2.1.2 Training - Gradient descent . . . 7
2.2 Influence Functions . . . 8
2.2.1 Necessary traits of the system . . . 10
2.2.1.1 Twice-differentiable loss function . . . 10
2.3 Dealing with the inverse hessian . . . 10
2.3.1 Conjugate Gradients . . . 11
2.3.1.1 Hessian assumptions . . . 12
2.3.2 LiSSA: Linear Stochastic Second-Order Algorithm . . . 12

3 Methods **13**
3.1 Implementation . . . 13
3.2 Leave-one-out-retraining . . . 13
3.3 Pedestrian detection . . . 16
3.3.1 Data sets . . . 16
3.3.2 Network . . . 17
3.3.3 Code Design . . . 17

4 Results **19**
4.1 Pedestrian detection problem . . . 19
4.1.1 Network performance . . . 19

4.1.2 Leave-one-out retraining results 20

5 Discussion **25**
5.1 Reasons for low correlation 25
5.2 Image interpretation 26
5.3 Retraining from $\hat{\theta}$ 28
5.4 Dataset 28
5.5 Further research 29
5.5.1 Road Segmentation 29
5.5.2 Earth mover's distance 30

6 Conclusion **33**

A Appendix 1 **I**

List of Figures

2.1 A schematic view of an artificial neural network. Data flows from left to right through layers where data processing takes place. 6

3.1 Schematic of the loss differences evaluated during the retraining experiment. 15
3.2 Work flow of the retraining experiment. $\hat{\theta}$, θ', $\tilde{\theta}_1$, $\tilde{\theta}_j$ and $\tilde{\theta}_k$ are parameter sets after the corresponding training sessions. LD stands for loss difference and is always taken in the point of the test image. . . . 16
3.4 The architecture of the studied network 17
3.3 The test image used throughout the experiments. 17

4.1 Label distribution in top 100 predictions. 20
4.2 In blue dots the retraining loss difference when retraining without the training image corresponding to the index of the x-axis. The orange dots are the predicted loss difference for the same training image. . . 21
4.3 Training images with top 5 highest predicted influence. 22
4.4 Training images with sixth to tenth highest predicted influence. . . . 23

5.1 Training images visually similar to the test image. 27
5.2 Training image with confusing annotation. 29

A.1 Training images with top 1 - 5 highest influence. II
A.2 Training images with top 6 - 10 highest influence. III
A.3 Training images with top 11 - 15 highest influence. IV
A.4 Training images with top 16 - 20 highest influence. V
A.5 Training images with top 21 - 25 highest influence. VI
A.6 Training images with top 26 - 30 highest influence. VII
A.7 Training images with top 31 - 35 highest influence. VIII
A.8 Training images with top 36 - 40 highest influence. IX
A.9 Training images with top 41 - 45 highest influence. X
A.10 Training images with top 46 - 50 highest influence. XI
A.11 Training images with top 51 - 55 highest influence. XII
A.12 Training images with top 56 - 60 highest influence. XIII
A.13 Training images with top 61 - 65 highest influence. XIV
A.14 Training images with top 66 - 70 highest influence. XV
A.15 Training images with top 71 - 75 highest influence. XVI
A.16 Training images with top 76 - 80 highest influence. XVII

A.17 Training images with top 81 - 85 highest influence. XVIII
A.18 Training images with top 86 - 90 highest influence. XIX
A.19 Training images with top 91 - 95 highest influence. XX
A.20 Training images with top 96 - 100 highest influence. XXI

List of Tables

4.1 Losses evaluated after the retraining with full data set. The first two columns contain the loss values taken before and after the extra training iterations, respectively. The third column is the difference between these values taken as $L(\hat{\theta}) - L(\theta')$ meaning that a negative sign is equivalent to an improvement in network performance. 20

1
Introduction

Artificial neural networks are one of the most popular machine learning models today. Out-of-the-box models from one of the large number of available frameworks make them accessible with little foreknowledge which has helped increase their popularity. However, with the spread of the use comes a growing concern of their lack of opacity. While they are powerful and adaptable to varying applications, their inner workings remain something of a mystery to humans. It might be annoying or cute when a face lock on a smart phone cannot differentiate between a parent and their child, but there are instances where the neural networks simply cannot be allowed to fail. In the growing field of autonomous vehicles, convolutional neural networks (CNNs) are widely used for various tasks such as road segmentation, [8], and feature detection, [22]. Today these models are pushed to their limits to accommodate for the rapid development speed as companies race to build the first fully autonomous vehicle. The pace of this development has brought to the public's attention the problems that arise when systems like these fail, [6].

Unlike a car accident where a driver has been controlling the vehicle, when an autonomous vehicle is involved it is not apparent what caused it or who is responsible. Such occurrences has shed light on the need for neural networks that can explain their decisions. This is important for several reasons. A deeper understanding of neural networks could help the development of new, more robust models. To this end, machine learning engineers need to open the black-box and understand the inner workings of the models. Research has been conducted seeking to pinpoint these mechanisms (see for example [2], [17] and [20]) and explain the behaviour of these models. A second reason is that the black-box property of neural networks decrease their trust from the end users, [3]. If these models provided explanations and justifications alongside their predictions this could increase the trust from the end user. Another reason why this is so important is that even accurate predictions can be difficult to leverage if the producing model is too opaque. If the model produces a motivation alongside the decision or classification it makes, these can be interpreted by a human to assess the correctness of the result. This is the case especially in medicine where convolutional neural networks (CNNs) have successfully been used in a large spectrum of applications such as identifying bone erosion from rheumatoid arthritis, [16], and diagnosing breast cancer, [19]. In these types of applications, the motivations of the neural network can be used by a medical professional to assess the quality of a classification, see [7].

Several attempts have already been made to produce explanations for CNN classifications. Because of the complexity of the models this can be done in many different ways. Some earlier works have studied individual pixels in the training

data, [21], and constructing attention maps to identify what parts of a training image the network considers most when classifying a test image, [18]. Given these different interpretation tools, attempts have also been made to unify them, [13]. These methods often seek to find a part of a single training image and how it impacts the behaviour of a network. In contrast, influence functions, [12], a concept originating from robust statistics, measure the impact of an entire training image on a given classification. Earlier works have applied them successfully to the MNIST and CIFAR-10 data sets with good results. Now the question is if they can also be applied to the feature detection systems of the autonomous vehicle industry.

1.1 Motivation

In the growing field of autonomous vehicles, neural networks are tasked with making precise and important decisions. Since the classification performed by a neural network is critical in this kind of application, their every aspect must be studied and understood. Network architecture is a common point of study, but the training data is another important factor in the training process. To this end, influence functions can be used to try to find a common characteristic in the data pinpointing valuable, harmful or insignificant data. This might help development of the classifications of the future since it can indicate what kind of training data the models use to learn different features.

Hopefully, this can lead to future systems used for pedestrian detection being even more robust than those existing today. Even if the problems studied in this project is down scaled both in terms of problem complexity and data set size, their outcome can provide an indication of whether influence functions are a viable option to further optimise the training set up for such systems.

1.2 Goals

The purpose of this thesis is to apply the theory of influence functions to a classifying CNN comparable to those used in autonomous vehicles today. The results could provide valuable insights on why these systems provide the classifications they do, which in turn is crucial for developers seeking to make future versions more robust. Moreover, user trust is dependent on human interpretation of the algorithms.

1.3 Contributions

To take state-of-the-art tools previously only used on academic applications and see if they make sense in the context of problems usually studied in the autonomous vehicle industry.

1.4 Limitations

A single binary pedestrian detection problem is studied to determine if influence functions can accurately predict how individual training images impact a given classification. The data used in the set up is a subset of the well studied KITTI object detection data set, [9]. Due to the computationally expensive operations needed by the experiment, the model was deliberately simple compared to those commonly used for similar data sets. Within the time constraints of the thesis a from-scratch implementation of the experiment could not been completed and the code run is heavily based on that used in earlier works, [12].

1.5 Outline of the thesis

The second chapter covers the mathematical theory alongside some approximation techniques. It also gives a very brief introduction to neural networks. Chapter three provides a detailed run through of the experimental procedure and the studied model and data set. The results of the experiment are presented in chapter four and discussed in chapter five. Finally, the sixth chapter contains the conclusions of the thesis.

2
Theory

Using influence functions on convolutional neural networks (CNNs) requires a fusion of a result of robust statistics with the classification models. This chapter starts with a brief run down of CNNs, which should not be seen as an exhaustive description of these systems but rather a reminder. For an introduction to CNNs please see one of the free sources online (for example the documentation of the machine learning framework DL4J, [5]). The focus will instead be put on the influence functions themselves, since understanding their notation is detrimental to interpreting the results of the experiments. Finally, the complexity of the CNNs pushes the equations to their limits and some approximations will be needed. Some of these methods will also be covered by this chapter.

2.1 Convolutional Neural Networks

The overall structure of all neural networks is a number of layers that given an input produces an output. The output of a preceding layer is the input of a following. This flow of data can be seen in figure 2.1. What kind of output that is produced and how it is calculated depends on the type of layer. This section will give a brief description of the layers of neural networks and how they are trained. It is by no means exhaustive. While neural networks is a large family of models, this project deals only with CNNs used for image classification. ("Is this an image of a dog or a cat" type questions). Apart from section 5.5.1 all neural networks mentioned in this report will be of this type.

2.1.1 Layer types

There are a number of different layers used in CNNs, especially state-of-the-art networks. This section will only cover the more traditionally used layer types, since the networks used in this project have been kept simple because of computational constraints. Every layer has parameters that are conditioned during training. Before applying the respective functions of a layer, the input is multiplied by some parameters (often called weights) and summed with others (called biases).

2.1.1.1 Convolutional layer

Like their name suggests, the convolutional layers perform convolutions on the data. CNNs are often used in image classifications so the data is often images. They are characterised by the size of their kernels which determines the line of sight of the

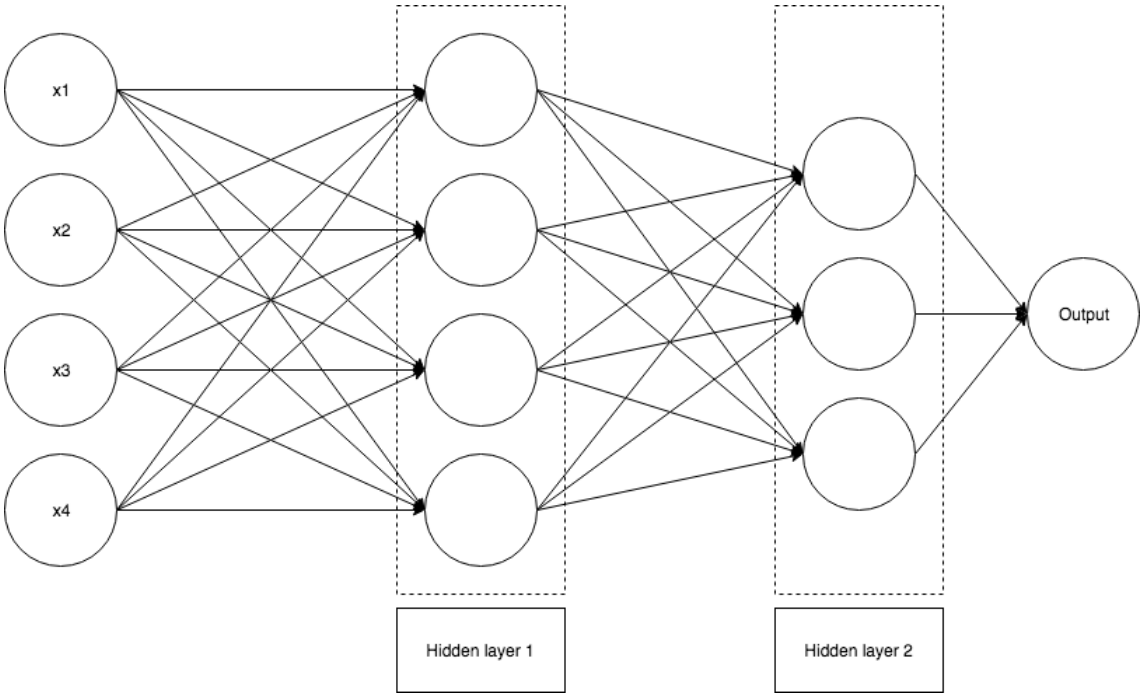


Figure 2.1: A schematic view of an artificial neural network. Data flows from left to right through layers where data processing takes place.

operation. The convolution can be interpreted as a filter swiping over the data, and by its nature, where the overlap is great the result will be higher. (So one convolutional layer in a network meant to detect images of dogs might have a 'snoot' filter).

After the convolutional operation, an activation function is applied to the result to add a measure of non-linearity to the layers. Two common choices of activation function are $tanh(x)$ and the rectifier function $R(x)$. The definition of $tanh$ is:

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.1}$$

while the definition of the rectifier function is:

$$R(x) = max(0, x) \tag{2.2}$$

this means that it is equal to 0 if its input is 0 and linear otherwise.

2.1.1.2 Max pooling layer

A pooling layer is often put after a convolutional layer to squash the size of the data by condensing the information. It swipes an $n \times n$ max filter over the image. The stride is often chosen to ensure that the regions to which it is applied are non-overlapping. This means the operation gives a reduction in data size as well as a sort of binning operation of the regions.

2.1.1.3 Softmax layer and cross entropy

The Softmax layer often serves as the final output from the network. It squashes a K-dimensional vector of arbitrary real values to a K-dimensional vector that can be interpreted as a probability distribution. Its values sum to 1 and the larger the number the higher the probability. When the number of target classes is K it is defined as:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{i=1}^{K} e^{z_i}} \tag{2.3}$$

where z is the input vector. Say a network is trying to determine if an image contains a cat or a dog. The image it is processing contains a dog which has label 0. The ground truth for this image is then a vector $[1, 0]$. If the result from the softmax layer is $[0.8, 0.2]$ this means the network believes to a degree of 80% that the image is of a dog and 20% that it is of a cat. In upcoming sections we need a metric of how well the network is performing. To determine the error in a classification output, there needs to be a comparison between the distribution resulting from the softmax function and the target feature vector for the input. To this end, cross entropy is often used. If the network input consists of N examples, the cross entropy between the distributions q (softmax output) and p (target vector) is defined as:

$$H(q, p) = -\sum_{x} p(x) \log(q(x)) \tag{2.4}$$

Cross entropy (equation 2.4) provides a metric that when minimised should give better network classifications. It is chosen as loss function in all models used in the project.

2.1.2 Training - Gradient descent

Say we have a neural network with given architecture and a set of parameters θ (the weights and biases of the network). Given an input x, the network will produce some output y. The goal is to perform an iterative fitting of the parameters θ (training) over known pairs of input-output (training data) until the network can correctly classify previously unseen examples (test data).

In each training step, the output of the network is compared to the (known) true classification of the input. The error of the classification is quantified as the loss $L(z_{\text{input}})$ (often the cross entropy from equation 2.4) of the network for the input z_{input}. To use this error to update the parameters, it's gradient it taken with respect to the parameters of the network:

$$\nabla_\theta L(z_{\text{input}}, \theta) = \nabla_\theta H(q(z_{\text{input}}, \theta), p) \tag{2.5}$$

where the right side holds if we use cross entropy as loss function. Now we will take a moment to remember what we are actually doing. Since the loss is a function of all the parameters of the system we are dealing with a scalar field with a well defined gradient. (It is well defined thanks to equations 2.4 and 2.3 and the chain rule of derivatives). By definition, the gradient of a scalar field always points in the

direction of steepest increase. (In three dimensions it is easily visualised as always pointing in the direction of the steepest climb from the point where it is evaluated). This means that taking a step in parameter space in the opposite direction should take us to a point where the value of the scalar field (loss) is lower. Based on this, the values of the parameters are updated by taking a small step in the opposite direction of the gradient of the loss:

$$\theta_{\text{new}} = \theta - \eta \nabla_\theta L(z_{\text{batch}}, \theta) \tag{2.6}$$

where η is the learning rate of the model. In some applications, η is adapted (reduced) throughout training which makes the earlier updates of the parameters more significant and then as the model is assumed to start reaching an optimum it decreases. Performing this kind of steps over a large number of iterations is the principle of gradient descent and the idea is to reach a minimum in the loss space. There are other optimisation options for neural networks (for example the ADAM optimiser[11]) that take into account additional metrics, but the idea of gradient descent is still present at their core.

There are a number of things to consider when putting gradient descent into practice. In addition to the learning rate decay already mentioned, there are other hyper parameters that can be used. Weight decay puts a penalty on large parameters and is implemented by adding a term to the loss function which means the loss will increase with larger parameters. It is one of the tools for preventing the network to model the noise of the training data in addition to the relevant features (overfitting).

Another risk factor for overfitting is the number of training iterations the networks goes through. The session should not be so long that the network memorises the training data, but needs to be long enough that it learns the characterising features of the target classes, (i.e "what is a dog"). An overfitted model will excel at classifying training data, but the memorised noise (some features of the training dog images that are not, in fact, characteristic of dogs) will make it worse at classifying test images. (Kind of like memorising compared to learning when studying for an exam).

2.2 Influence Functions

To understand what influence functions are it can be helpful to look at what they seek to find; "The measure of the dependence of the estimator on the value of one of the points in the sample", [4], or in other words: "Which training data is responsible for a given prediction?". We can break the question down into two parts:

What is the effect from a training point on the parameters of the model? (2.7)

and

What is the impact of the parameters on the loss for the test point? (2.8)

It should be evident that in mathematical terms these are connected through the chain rule of derivatives.

For neural networks, the answer to the first question is the impact of one training image on the parameters θ. We cannot directly answer this, so we ask a similar question "How are the parameters changed if we up weight a training point by some small measure ϵ?". (In practice this would correspond to increasing the frequency of training on the given data point.) In mathematical terms, this means:

$$\begin{aligned}\mathcal{I}_{\text{up, params}}(z) &= \left.\frac{\mathrm{d}\hat{\theta}_{\epsilon,z}}{\mathrm{d}\epsilon}\right|_{\epsilon=0} \\ &= -H_{\hat{\theta}}^{-1}\nabla_\theta L(z,\hat{\theta})\end{aligned} \tag{2.9}$$

where

$$H_{\hat{\theta}} = \frac{1}{n}\sum_{i=1}^{n}\nabla_\theta^2 L(z_i,\hat{\theta}) \tag{2.10}$$

with $z_i \in$ training data set. This is a result from robust statistics[4]. The reason ϵ is set to 0 is that it corresponds to considering all training points equally, which is the case we are interested in. This means that equation 2.9 answers the question in equation 2.7.

Now we move on to the question in equation 2.8. Again we will consider an up weighting ϵ when deriving the equations, but set it to 0 to ensure the results hold for the base case where no up weighting is present. The impact on $L(z_{\text{test}},\hat{\theta})$ by this up weighting is equal to its derivative:

$$\begin{aligned}\mathcal{I}_{\text{up, loss}}(z,z_{\text{test}}) &= \left.\frac{\partial L(z_{\text{test}},\hat{\theta})}{\partial\epsilon}\right|_{\epsilon=0} \\ &\quad \{\text{Deriving according to the chain rule}\} \\ &= \nabla_\theta L(z_{\text{test}},\hat{\theta})\left.\frac{\partial\hat{\theta}_{\epsilon,z}}{\partial\epsilon}\right|_{\epsilon=0} \\ &\quad \{\text{Leveraging equation 2.9}\} \\ &= -\nabla_\theta L(z_{\text{test}},\hat{\theta})H_{\hat{\theta}}^{-1}\nabla_\theta L(z,\hat{\theta})\end{aligned} \tag{2.11}$$

This ties together the answers to the two questions of equations 2.7 and 2.8. We now define the product between $H_{\hat{\theta}}^{-1}$ and the gradient of the test loss as

$$s_{\text{test}} = H_{\hat{\theta}}^{-1}\nabla_\theta L(z_{\text{test}},\theta) \tag{2.12}$$

which means that the influence of a single training image z_i on a test image z_{test} is:

$$\mathcal{I}_{\text{up, loss}}(z_i,z_{\text{test}}) = -\nabla_\theta L(z_i,\hat{\theta})s_{\text{test}} \tag{2.13}$$

The negative sign means that a positive predicted influence means that the algorithm believes that the training image is helpful when classifying s_{test}. (Removing the

training image would cause a positive change in the loss function, and a higher loss is worse). After comparing $\mathcal{I}_{\text{up, loss}}(z_i, z_{\text{test}})$ across all i that the most influential training data can be found.

The equations are deceivingly concise but hides some computational traps. First of all, remember from equation 2.10 that $H_{\hat{\theta}}$ is a sum of the twice differentiated loss function in all training points with respect to every parameter of the model. To top that up, it is also inverted. Even just applying the ∇_θ operator once is a cumbersome operation since it means differentiating with respect to all parameters of the network, and for models such as neural networks the number of parameters are usually very large.

To allow for a practical implementation there are several numerical approximation approaches. Two possible approaches are to get the inverse hessian vector product (s_{test}) through the conjugate gradients method, or; approximating the inverted hessian by leveraging Taylor expansion (LiSSA method) and then inserting this value into equation 2.12 to get s_{test}. These approaches will be further discussed in section 2.3.1 and 2.3.2. (The focus on these two methods comes from their proven success in earlier experiments, [12]).

2.2.1 Necessary traits of the system

The equations described have neat forms, but there are requirements hidden in their formulations. It is evident from equation 2.12 that the loss of the system needs to be twice differentiable. Moreover, if we want to use the conjugate gradients method, the hessian of the system has to be positive definite and symmetrical. The former is true if we have reached a local minimum of the parameter space during training (i.e. if we have trained the network enough) and the latter holds if (again) the second derivatives are continuous in an environment surrounding $\hat{\theta}$.

2.2.1.1 Twice-differentiable loss function

From the definition of the influence functions, equation 2.9, it is evident that if we want to calculate the influence for a model the associated loss function must be twice-differentiable. (Or the weaker condition; it must be possible to approximate it with good enough precision). In reality this is only a slight limitation on the possible choices of loss definition, and since the cross entropy used in this project passes this criterion it will not be investigated further. (Some exploration of this topic this is done in earlier works [12] with good experimental results for approximated gradients). For the applications of this project, the gradient of the loss (cross entropy) with respect to the parameters is well defined. The second order derivatives used in the LiSSA approximation method were evaluated using an adaption of the Tensorflow hessian method.

2.3 Dealing with the inverse hessian

The worst bottleneck when applying influence functions to neural networks is the inverse hessian. Neural networks can have a large number of layers which each in

turn have a set of weights and biases. Since the hessian takes into account the second derivative with respect to each parameter it is important to find a sufficiently, but not overly, complex model architecture since this will make the evaluation of the hessian too cumbersome.

Explicitly evaluating and inverting the hessian part of equation 2.12 is basically impossible when studying a neural network of reasonable size. To avoid doing this, we can use an approximation instead. This section introduces algorithms for this.

2.3.1 Conjugate Gradients

To use the conjugate gradients method, the problem must be reinterpreted and certain assumptions made on the hessian. Instead of calculating the hessian, inverting it and multiplying it by a vector, which is what we want in accordance with equation 2.12, we instead look at this as an implicit hessian vector product (IHVP). In those terms the vector we want to multiply the inverse hessian by is the gradient $\nabla_\theta L(z_{\text{test}}, \theta)$. For simplicity in the derivations we will set some new variables. We are looking for:

$$s_{\text{test}} = H_{\hat{\theta}}^{-1} L(z_{\text{test}}, \hat{\theta}) = H_{\hat{\theta}}^{-1} v = t \tag{2.14}$$

with $v = L(z_{\text{test}}, \hat{\theta})$ and t as the unknown, sought-after quantity. s_{test} now coincides with the solution t to the matrix equation:

$$H_{\hat{\theta}} t = v \Leftrightarrow t = H_{\hat{\theta}}^{-1} v \tag{2.15}$$

We form the quadratic form $f(t) = \frac{1}{2} t^T H_{\hat{\theta}} t - v^T t$. Given that the hessian is positive definite (PD) and symmetrical (explored below in section 2.3.1.1), the t that minimises $f(t)$ is also the solution to equation $H_{\hat{\theta}} t = v$. We will use the fact that the gradient of the quadratic form

$$f'(t) = \frac{1}{2} H_{\hat{\theta}}^T + \frac{1}{2} H_{\hat{\theta}} t - v \tag{2.16}$$

will reduce to

$$f'(t) = H_{\hat{\theta}}^T - v \tag{2.17}$$

if the hessian is symmetric. Setting $f'(t) = H_{\hat{\theta}} t - v = 0$ we see that we are back to equation 2.15. This means that if we find the v that correspond to this stationary point (which is a minimum given the assumption that the hessian is PD) we have found s_{test}. Given this, we must only find the value v that minimises the quadratic form to find s_{test}.

A more compact summary of the results is: if $H_{\hat{\theta}}$ is positive definite and symmetric, the solution t to the equation $H_{\hat{\theta}} t = v$ coincides with $\arg\min f(t) = \arg\min \left(\frac{1}{2} t^T H_{\hat{\theta}} t - v^T t \right)$. So the inversion problem has been transformed into a minimisation problem. To solve this equation through conjugate gradients method we leverage the first and second derivative of the quadratic form: $f'(t) = H_{\hat{\theta}} t - v$ and

$f''(t) = H_{\hat{\theta}}$. This problem can then be solved with out-of-the-box tools such as fmin_ncg (Newton-CG method) from the scipy Python library.

While the conjugate gradients method is a viable option for obtaining the inverse hessian product, the size of the CNNs will still affect the efficiency of the approximations. In each iteration, we need to take the first and second derivative of the quadratic form. While they both have nice closed forms, they will still scale with the square of the number of parameters of the network. Once again the equations are deceptively concise while reality reveals the true complexity of the method.

2.3.1.1 Hessian assumptions

As a reminder, this method requires two things of the hessian; that it is symmetrical and positive definite in the point $\hat{\theta}$. The former is true if the second derivatives are continuous in a neighbourhood surrounding $\hat{\theta}$ while the latter holds when $\hat{\theta}$ corresponds to a minimum in parameter space. For cross entropy the second derivatives will be continuous. The convexity of the hessian in $\hat{\theta}$ depends on the model architecture and the training. In practice this means training will be done longer than what is usual in classifying problems since it is crucial that a minimum has been reached.

2.3.2 LiSSA: Linear Stochastic Second-Order Algorithm

Another approach to dealing with the hessian computation problems is the Linear Stochastic Second-Order Algorithm, or LiSSA, presented by Agarwal et al in 2016[1]. Instead of approximating a hessian vector product like conjugate gradients does, it seeks the value of the inverse hessian itself. It is is based on the Taylor expansion of the inverse Hessian matrix:

$$H^{-1} = \sum_{i=0}^{\infty} (I - H)^i \tag{2.18}$$

where I denotes the identity matrix. Note that equation 2.18 assumes that the norm of H is ≤ 1, $||H|| \leq 1$ and that H is positive definite ($H \succ 0$). Furthermore, we denote the first j terms of the Taylor expansion H_j^{-1}:

$$H_j^{-1} = \sum_{i=0}^{j} (I - H)^i \tag{2.19}$$

for which it holds that $H_j^{-1} \to H^{-1}$ as $j \to \infty$. In practice, this approximation is computed recursively for a given (large) number of steps. The result is then multiplied with the test loss gradient according to equation 2.12 to get s_{test}.

3
Methods

The main experiment follows the set up of earlier works applying influence functions to neural networks (see [10] and [12]). These steps will be described in section 3.2. To make the implementation easier, a machine learning framework was used for the base operations. There are a number of choices that supports sophisticated operations for neural networks and one had to be chosen early on in the project. The choice of Tensorflow was motivated by knowledge among the group where the thesis was done (available support) and the fact that it would make it easier to directly leverage open-source code from previous studies[12].

As previously mentioned, the methods of analysis are equivalent to those done before (see [10] and [12]). While the steps of the experiments are the same as in those previous works, the data used is more true to actual data used in the autonomous vehicle industry. This means that network-building is not a major part of the project, and that the results will be more applicable to the development of systems for autonomous vehicles. While it would have been useful if the networks studied had also followed the architecture of those used in these applications, hardware constraints made this impossible.

3.1 Implementation

The experiments run in the project leveraged code used in earlier research[12]. However, alterations have been made to allow for larger and RGB (colour) images. Generalisations has been introduced to the code base which allows it to run experiments on different types of data.

3.2 Leave-one-out-retraining

Recall from section 2.2 that what influence functions claims is "This is the effect the training point had on the prediction for the test point". The straight-forward way of validating this claim is to simply retrain the model without the given training point. This has been done before with successful results in [10] and [12]. The schematic of the full retraining experiment in figure 3.2 explains the work flow of the method while the plot in figure 3.1 shows an exaggerated example of the evaluated loss differences.

A test point z_{test} is chosen as the point of study. We must also choose a number k for how many predicted top impact training points we want to study. This number severely affects the time complexity of the experiment, but needs to

be large enough to ensure that the statistical metrics can be trusted. The number of training iterations n_{training} as well as the number of retraining iterations $n_{\text{retraining}}$ must also be chosen.

The model goes through an initial training session after which the parameter set of the network (denoted $\hat{\theta}$) is saved. The loss $L(z_{\text{test}}, \hat{\theta})$ is calculated and stored for future reference. Then the influence functions are used according to section 2.2 to make predictions on what the impact from each of the data points of the training set is on the loss for z_{test}. These values are used to evaluate which training points are the most important for the loss in the point z_{test}. These top impact training points are denoted z_j, and the predictions for their influence on the test image z_{test} is called $\mathcal{I}_{\text{up, loss}}(z_j, z_{\text{test}})$ in accordance with equation (2.13).

As a pre-step to the retraining part of the experiment, the model is trained for $n_{\text{retraining}}$ steps using the full data set, resulting in the parameter set θ'. This is done for two reasons; to ensure that the changes in loss

$$LD_{\text{full}} = L(z_{\text{test}}, \theta') - L(z_{\text{test}}, \hat{\theta}) \tag{3.1}$$

and

$$LD_{\text{full}}^{\text{train}} = \sum_{i=1} L(z_i, \theta') - L(z_i, \hat{\theta}) \tag{3.2}$$

with $z_i \in$ training data set are both close to 0. This is important since it indicates that the initial training has reached a somewhat stable point.

In the second part of the experiment, the retraining steps begin. This is where the predicted loss differences should be validated by measured loss differences. The model is reset to the parameter set $\hat{theta}$ and trained for an additional $n_{\text{retraining}}$ steps while withholding one of z_j at a time. (That is, the set of examples from where the batches are drawn consists of all original training images except z_j). We will call the parameter set after one such retraining session $\tilde{\theta}_j$. At the end each retraining session, the loss in the test point $L(z_{\text{test}}, \tilde{\theta}_j)$ is evaluated and the loss difference (LD_j), considered ground truth for the influence from training point is z_j, is taken according to:

$$LD_j = L(z_{\text{test}}, \tilde{\theta}_j) - L(z_{\text{test}}, \hat{\theta}) \tag{3.3}$$

The plot in figure 3.1 might be helpful to get a more intuitive understanding of this measure. Before retraining while withholding the next z_j, the parameters are reset to $\hat{\theta}$ so that every session starts with the parameter set the network had after the initial training. As a reminder, the schematics in figure 3.2 can be helpful to visualise this. This produces a set of n values describing the actual loss difference for the test point when the training point is withheld. These values should correlate with the loss differences predicted using influence functions. One suitable metric to determine this correlation is through the Pearson correlation coefficient r:

$$r = \frac{\sum_{j=1}^{k} \left(LD_j - \overline{LD}\right)\left(\mathcal{I}_{\text{up, loss}}(z_j, z_{\text{test}}) - \overline{\mathcal{I}}\right)}{\sqrt{\sum_{j=1}^{k} \left(LD_j - \overline{LD}\right)^2} \sqrt{\sum_{j=1}^{k} \left(\mathcal{I}_{\text{up, loss}}(z_j, z_{\text{test}}) - \overline{\mathcal{I}}\right)^2}} \tag{3.4}$$

where

$$\overline{\mathcal{I}} = \frac{1}{k} \sum_{j=1}^{k} \mathcal{I}_{\text{up, loss}}(z_j, z_{\text{test}}) \tag{3.5}$$

$$\overline{LD} = \frac{1}{k} LD_j \tag{3.6}$$

r assumes a value between -1 and +1 (inclusively) where +1 corresponds to complete linear and -1 perfect inverse linear correlation. If $r = 0$ the data sets have no linear correlation.

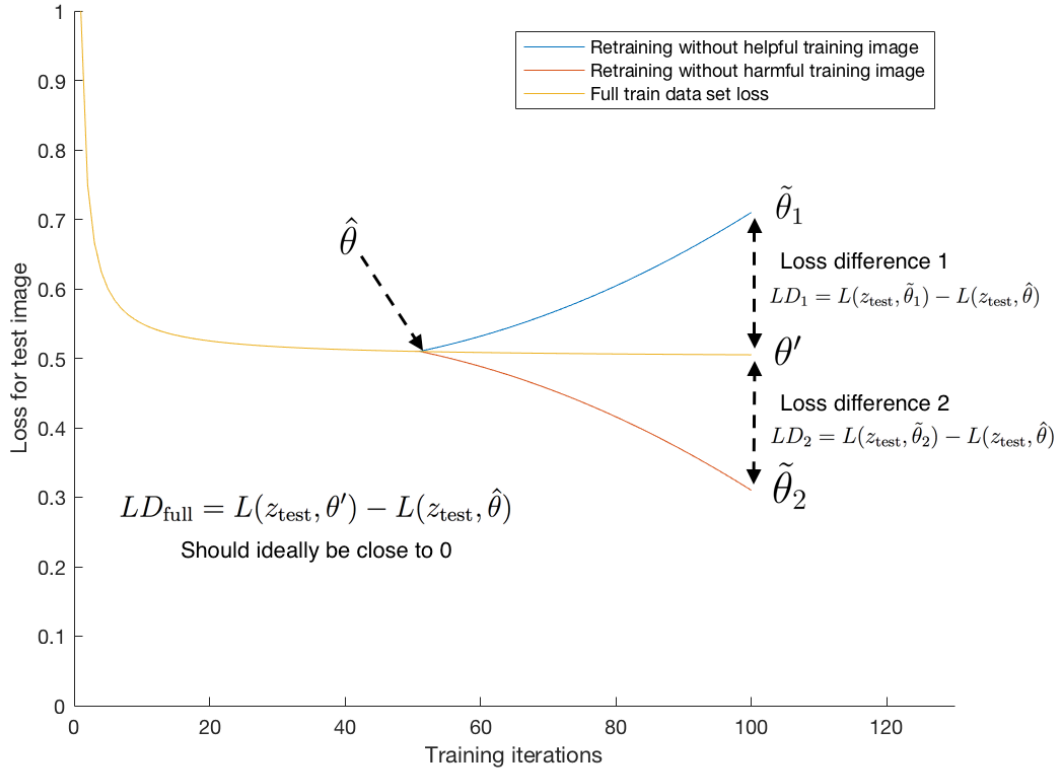


Figure 3.1: Schematic of the loss differences evaluated during the retraining experiment.

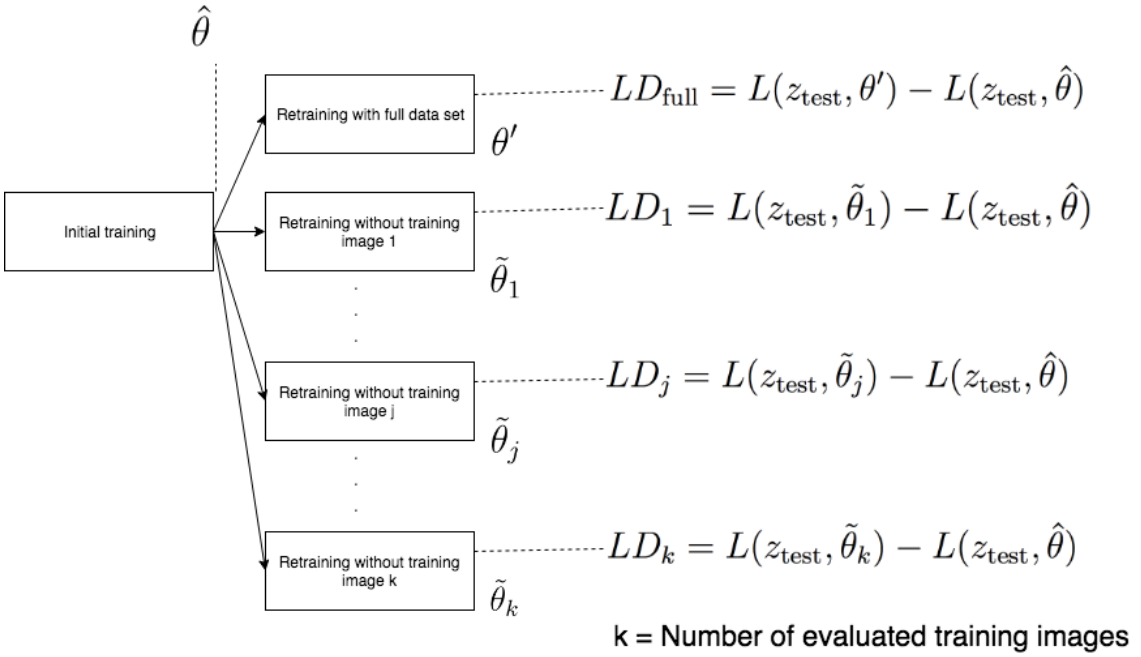


Figure 3.2: Work flow of the retraining experiment. $\hat{\theta}$, θ', $\tilde{\theta}_1$, $\tilde{\theta}_j$ and $\tilde{\theta}_k$ are parameter sets after the corresponding training sessions. LD stands for loss difference and is always taken in the point of the test image.

3.3 Pedestrian detection

To evaluate how useful influence functions are for applications in autonomous vehicles they have been applied on a subset of the KITTI dataset, [9], an object detection data set. The problem setup is a simple binary classifier tasked with determining whether a given image contained a pedestrian or not. The principal idea of this experiment is in line with earlier published works, [10] and [12], but the nature of the data set used is very different.

3.3.1 Data sets

Compared to the academic data sets studied before where the image sizes are 28x28, [12], and 30x30, [10], pixels, the KITTI data set image size is 370x1224 pixels. In order to use this data set, the images are down sampled by a factor 0.5, which mean that the resulting image size is 185x612 pixels. To ensure that the feature in an image did not shrink too much from the down sampling, a threshold of 35 pixels was set for the resulting height of a pedestrian. If its height fell below this after down sampling, the image would be labelled 0 (meaning it contained no pedestrian). From the KITTI object data set 683 images meet these requirements. To balance the set, 683 images containing no pedestrians were added. To increase the size of this set the images where then mirrored which meant the total training set contained 2732 images. 32 of these became a validation data set which meant the final size of the training data set was 2700.

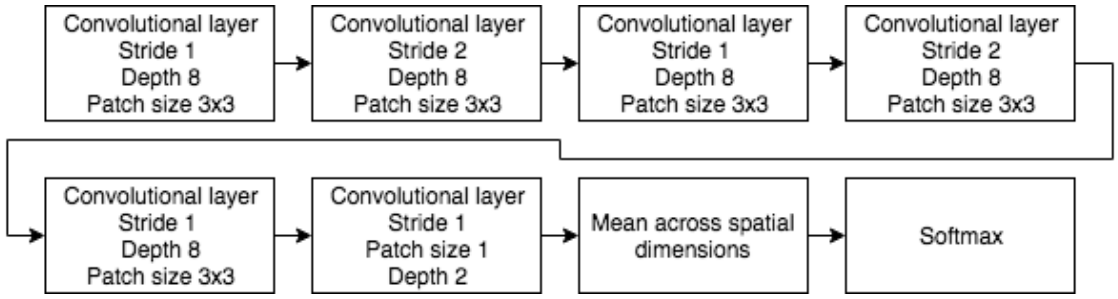


Figure 3.4: The architecture of the studied network

The testing data was hand annotated based on the requirements above since KITTI contains no test labels. A test image (figure 3.3) was hand picked for the experiments based on feature visibility and the fact that there were a number of images from the same scene that also contained pedestrians.

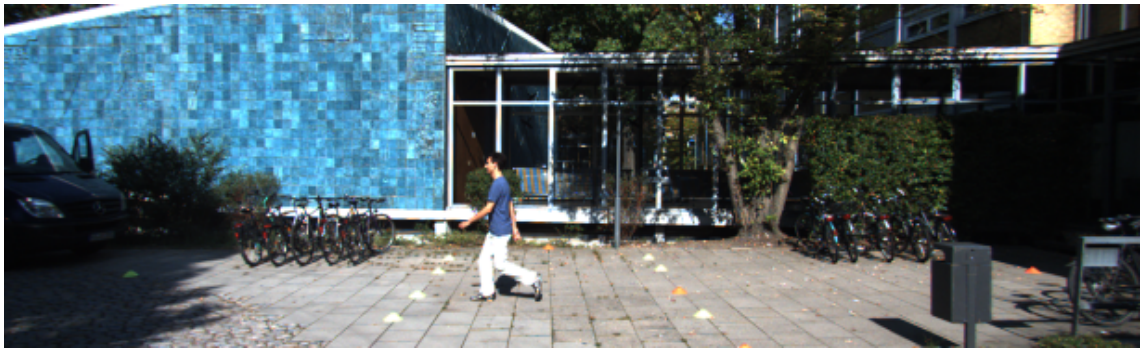


Figure 3.3: The test image used throughout the experiments.

3.3.2 Network

The network architecture was the same as that used to run the influence experiment in previous works [12]. It consisted of 6 consequent layers of convolutions with tanh activations, finalised by a softmax layer. The convolutional layers had a patch size of 3x3 pixels. It utilised weight decay on its parameters. It contained no maxpool layers which meant that all layers had easily interpreted gradients. All six layers contained eight hidden units. The schematics of the network can be seen in figure 3.4.

3.3.3 Code Design

Some of the design choices of the code base are not in accordance with the current Tensorflow guidelines. While this puts a dent in the efficiency of the code, it was necessary if Tensorflow was to be used at all. In particular, during the spring of 2018 when the project took place, queue runners were the recommended data feeding practice for projects in Tensorflow. While these provide an efficient way of injecting data into the model, they offer no option of tailoring the batches. During the retraining step where all but one example is used for training would have required

n separate instances of queue runner set ups. (Where n is the number of images withheld, one at a time). While this is certainly possible to do, it would have required scripts constructing each individual local data set to be read from, since the queue runners consume blindly from their given source.

4
Results

There are two important metrics to keep in mind when looking at the results. First, we have the predicted influence on the loss for the test image from training image z_i, denoted $\mathcal{I}_{\text{up, loss}}(z_i, z_{\text{test}})$. Then there is the measured loss difference when retraining without the training image, LD_j. This is only evaluated for the training images with the 100 highest predicted influences (z_j), i.e z_j is a subset of z_i.

The results were acquired by training for an initial 50 000 iterations and then using influence functions to get the predicted loss differences $\mathcal{I}_{\text{up, loss}}(z_i, z_{\text{test}})$. After this, an additional 10 000 iterations long training session was completed. The last step was the retraining sessions when one training image at a time was withheld from the network, and after each of them the respective measured loss differences LD_j was evaluated.

A negative LD_j means the network got better when training image z_j was withheld and that it had previously impeded the classification of the test image. If LD_j is positive it meant the loss for the test image went up after retraining without z_j and that it had been helpful to the model.

4.1 Pedestrian detection problem

This section presents the results from running the retraining experiment using the network configuration described in section 3.3.2. The initial training consisted of 50 000 iterations and retraining for 10 000 iterations. Batch size was 100 for both training sessions. The conjugate gradients method described in section 2.3.1 was used to approximate the IHVP (implicit hessian vector product), and it was done over 100 iterations. The IHVP approximation was terminated when the maximum number of iterations was reached and a minimum had not been reached.

4.1.1 Network performance

The final accuracy across the train data was 0.85 and across the test data 0.72. Considering the problem at hand was a binary classifier these numbers do not inspire confidence. However, the network gave the softmax output [0.0300, 0.9699] for the test image with label [0, 1].

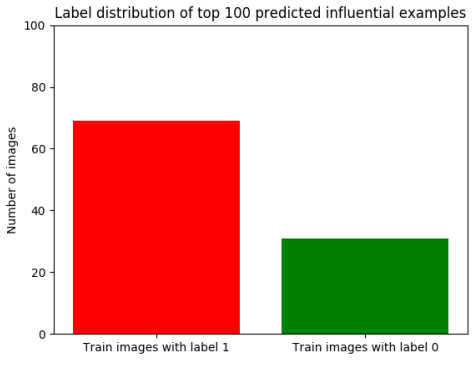
The control retraining loss differences LD_{full} and $LD_{\text{full}}^{\text{train}}$ was evaluated after the initial training. Their values can be seen in table 4.1.

	Original model ($\hat{\theta}$)	Retrained model (θ')	Difference
z_{test}	0.030558474	0.019499514	LD_{full} = -0.01105896
All z_{train}	0.38356976	0.36736464	$LD_{\text{full}}^{\text{train}}$ = -0.01620513

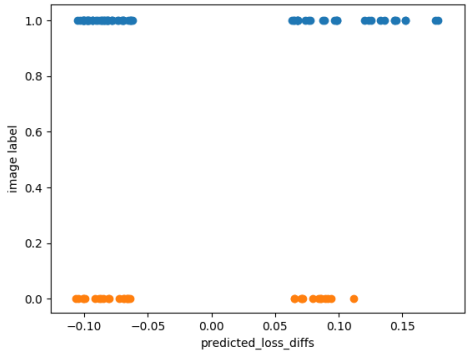
Table 4.1: Losses evaluated after the retraining with full data set. The first two columns contain the loss values taken before and after the extra training iterations, respectively. The third column is the difference between these values taken as $L(\hat{\theta}) - L(\theta')$ meaning that a negative sign is equivalent to an improvement in network performance.

4.1.2 Leave-one-out retraining results

The measured and predicted loss differences (LD_j, $\mathcal{I}_{\text{up, loss}}(z_j, z_{\text{test}})$) can be seen in figure 4.2. The magnitude of the predictions grow with increasing index since that is the selection metric for the set z_j. However, there is a cluster of predictions in the top right corner that seem to break the somewhat linear characteristics of the other prediction values. The images corresponding to these predictions can be seen in figures 4.3 and 4.4. Figure 4.1a indicates that according to the influence functions, a larger impact on the loss of the test image will come from training images also containing pedestrians. The plot in figure 4.1b show the predictions for images with and without pedestrians (blue and orange, respectively). The breaking of symmetry from figure 4.2 is evident in the top right corner here as well. The Pearson correlation coefficient (equation (3.4)) between the predictions and the actual loss was evaluated as $r = -0.0177$.



(a) Comparison of the number of label 1 and label 0 among the predicted top 100 most influential training images.



(b) Labels of training images compared to the value of their predictions. Blue dots are training images with label 1 (pedestrian) and orange label 0 (no pedestrian).

Figure 4.1: Label distribution in top 100 predictions.

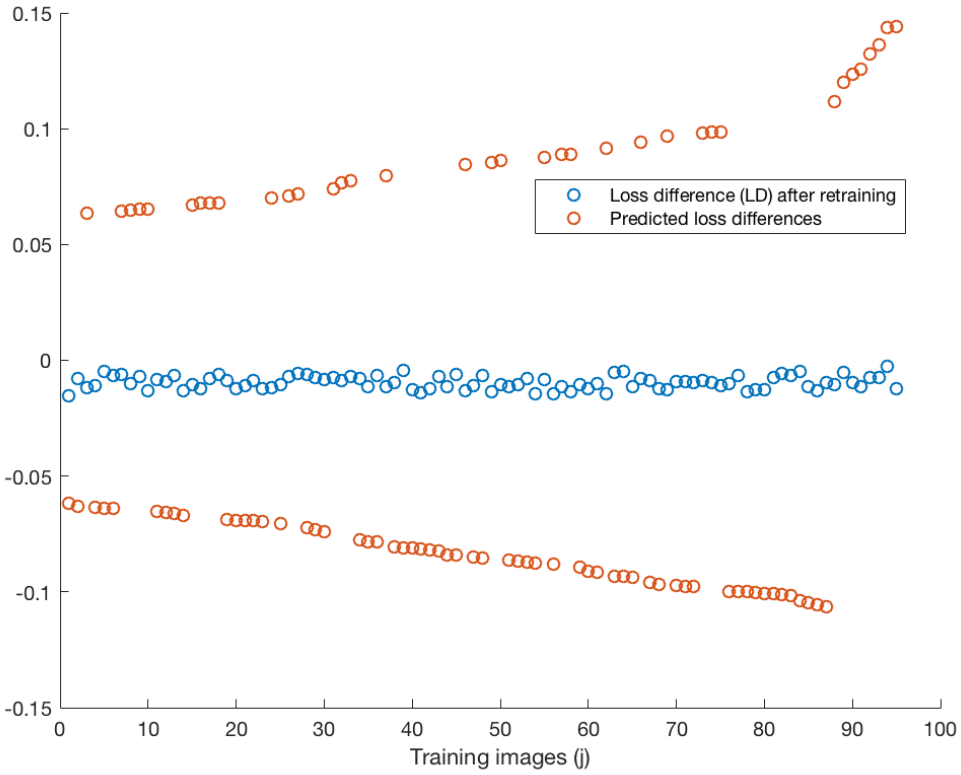


Figure 4.2: In blue dots the retraining loss difference when retraining without the training image corresponding to the index of the x-axis. The orange dots are the predicted loss difference for the same training image.

(a) Training image with index 1713. Highest predicted influence with $\mathcal{I}_{\text{up, loss}}(z_{100}, z_{\text{test}}) \approx 0.1762$.

(b) Training image with index 1661. Second highest predicted influence with $\mathcal{I}_{\text{up, loss}}(z_{99}, z_{\text{test}}) \approx 0.1524$.

(c) Training image with index 2497. Third highest predicted influence with $\mathcal{I}_{\text{up, loss}}(z_{98}, z_{\text{test}}) \approx 0.1523$.

(d) Training image with index 1343. Forth highest predicted influence with $\mathcal{I}_{\text{up, loss}}(z_{97}, z_{\text{test}}) \approx 0.1454$.

(e) Training image with index 2493. Fifth highest predicted influence with $\mathcal{I}_{\text{up, loss}}(z_{96}, z_{\text{test}}) \approx 0.1439$.

Figure 4.3: Training images with top 5 highest predicted influence.

(a) Training image with index 1712. Sixth highest predicted influence with $\mathcal{I}_{\text{up, loss}}(z_{95}, z_{\text{test}}) \approx 0.1438$.

(b) Training image with index 455. Seventh highest predicted influence with $\mathcal{I}_{\text{up, loss}}(z_{94}, z_{\text{test}}) \approx 0.1362$.

(c) Training image with index 1577. Eighth highest predicted influence with $\mathcal{I}_{\text{up, loss}}(z_{93}, z_{\text{test}}) \approx 0.1324$.

(d) Training image with index 1660. Ninth highest predicted influence with $\mathcal{I}_{\text{up, loss}}(z_{92}, z_{\text{test}}) \approx 0.1257$.

(e) Training image with index 2421. Tenth highest predicted influence with $\mathcal{I}_{\text{up, loss}}(z_{91}, z_{\text{test}}) \approx 0.1235$.

Figure 4.4: Training images with sixth to tenth highest predicted influence.

5
Discussion

While influence functions have been applied to classifying CNNs before, it has only been done in academic settings with data sets primed for the task. The experiments performed in this project seems to be one of the first tries with applying them to data closer to that processed by autonomous vehicles. During the project, the available hardware put considerable constraints on the architecture of the networks to be studied. It is possible that when taking the step from academic data sets such as MNIST to the more complex KITTI object data set, these networks simply could not keep up. While a ramp up in the network architecture might have been positive for the results, the models used were already saturating the available memory in the GPU. The time constraint from it being a thesis project also meant that experiments simply could not take too long.

The results after using the first network architecture had a correlation coefficient of $r = -0.0177$, which means that there was basically no correlation between the predictions and measure retraining loss difference. In comparison, earlier works, [12], have produced values of $r > 0.9$ after equivalent experiments on the MNIST data set. The major mystery of the results from the KITTI experiment is the obvious theme of the top 10 predicted influential images of figures 4.3 and 4.4. It seems like they share some feature that the equation 2.13 picks up on. Since the correlation between the measured loss differences is basically nonexistent, this is the only indication in the results that the influence functions manage to find some intrinsic characteristic in the training data. However, it is not evident why the algorithm considers these types of images to be so important to z_{test}.

5.1 Reasons for low correlation

One important sign that the algorithm was not given full freedom to make correct predictions is the control loss differences taken after the retraining using the full training set. Recall from table 4.1 that there was a significant decrease in loss for both the full training data set and for the test image after these extra 10 000 iterations. This strongly indicates that the initial 50 000 training iterations were not sufficient for the model to reach a stable point. It also rules out the explanation that the model got stuck in a local minimum. Further proof that the model should have gone through a longer initial training is that all of the 100 retraining loss differences have a negative sign. This means that regardless of which training image that was withheld during retraining the model still improved. Available resources made it impossible to run probing experiments to find these warning signs in time,

or run additional experiments with a higher number of training iterations. It is possible that changing only these hyper parameter would be enough to improve the correlation between the predictions and the measures loss differences.

The fact that time constraints meant that influence functions could only be tested on a smaller network was probably one of the biggest reasons for the disappointing results. The studied network simply does not have the architecture needed to classify images as complex as those in the KITTI object detection data set, even after they were down sampled. It was built to run on the MNIST data set which is barely comparable to those used in the pedestrian detection problem where the images sizes are magnitudes larger and the images are RGB instead of grey scale. It is probable that a more sophisticated network tailored to the data used would have performed better.

The methods used for the approximation of the hessian has never been completed without warning from the used method, either numerical precision loss or exceeding the maximum allowed number of iterations without reaching a satisfactory approximation. When the method was run on the MNIST data set, it ran without problems. The complexities of both the problem and the network will increase the difficulty of approximating the inverse hessian product, but without further study of the problem it is not obvious how much. It is possible that a different method is needed for the hessian approximation when studying data sets such as KITTI compared to those used when running the experiment on MNIST.

To summarise, the experiment contained many arbitrary hyper parameters such as the number of iterations for initial and the retraining sessions, the number of iterations used for hessian or IHVP approximation and which test image is used. One of the most important, the network architecture, proved to be very difficult to explore given the available time and hardware resources available.

5.2 Image interpretation

The 10 top predicted influential images (figures 4.3 and 4.4) were all assumed to be helpful to the test image (figure 3.3). They are all from the same scene even though they are scattered across the data set. This indicates that the prediction algorithm finds something in these types of images helpful to the model when classifying the test image, but it is not evident what that is. There are images in the training set that are taken at the same scene as the test image, but they did not get as high predicted influence as these images. (In fact, they were not even present among the predicted top 100 influential training images). One thing to note is that figures 4.3d, 4.3e, 4.4b and 4.4c all contain a blue car in the shadows of the left side. The test image also has a brightly blue segment in the shape of a building wall. However, since the blue shadows are not present in all the top 10 predicted influential images, it cannot be the whole explanation to their high prediction values. The overall light/shade composition of the images might be another explanation, but the images from the same scene as z_{test} are more similar in that regard.

Interestingly, the pedestrian giving many of these images their label is almost invisible in the shadows. The fact that the network still finds them useful might be explained by the presence of the bicycle riders that resembles pedestrians to a high

enough degree. This points to the fact that perhaps the model could not differentiate properly between pedestrians and bicyclists, further impeding its performance.

The complete set of the 100 training images with highest predicted influence can be seen in A. As mentioned earlier, none of them are from the same scene as the test image, even though the training data contains a number of such images with pedestrians in them. (Some examples of images like these can be seen in figure 5.1). Since the test image has a characteristic blue wall it is not naive to expect that at least some of these images would be present among the top 100 predicted influential training images. It is not apparent why they are not.

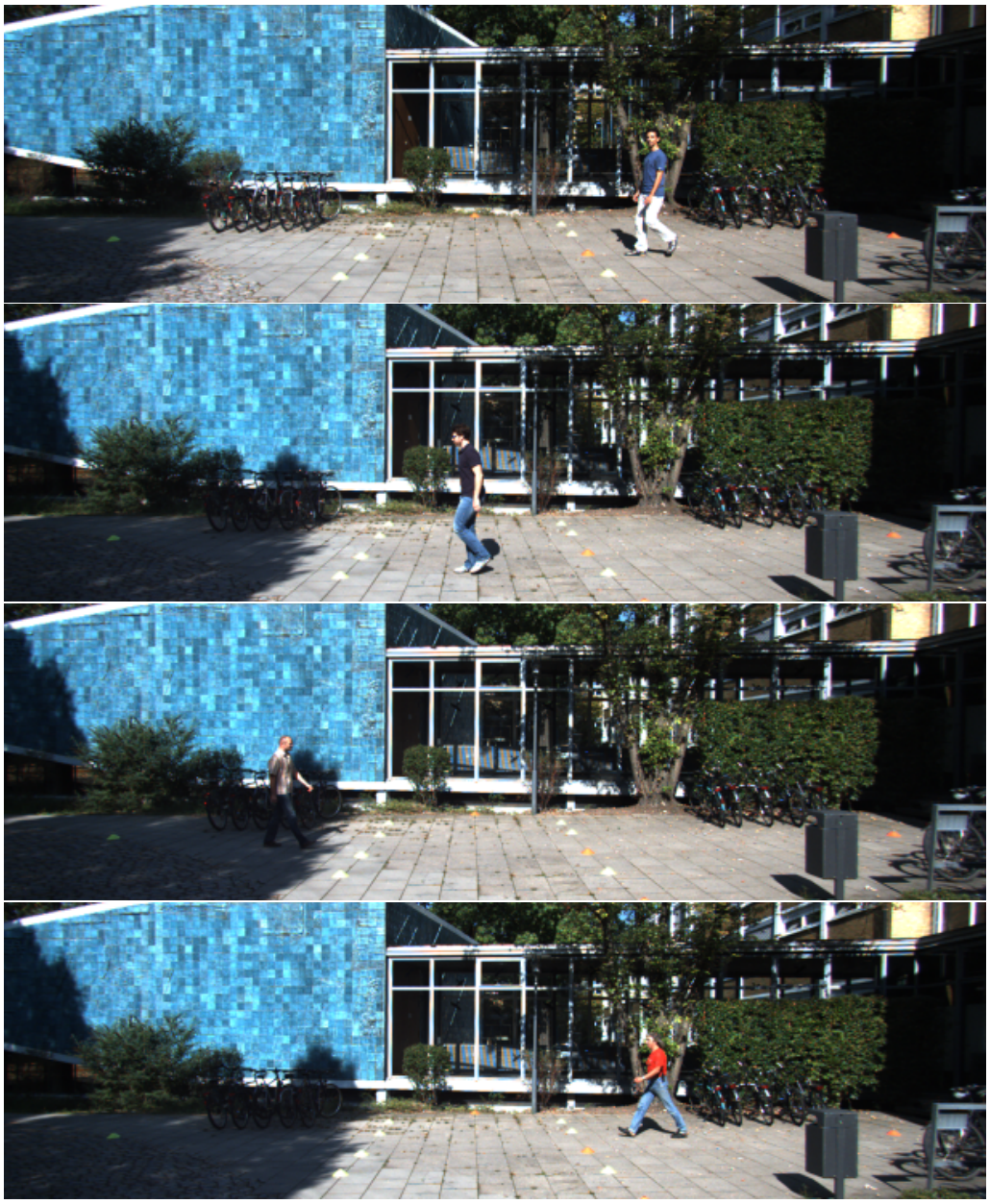


Figure 5.1: Training images visually similar to the test image.

5.3 Retraining from $\hat{\theta}$

The design of the retraining experiment was the same that was used in earlier works, [12] and [10]. As a reminder, it starts each retraining session at a point $\hat{\theta}$ that the model has reached by already going through training. With enough time and computational power, another evaluation of the predictions could be done where the model is retrained from scratch with randomly initialised parameters. One training image at a time would then be withheld as before, but each retraining session would start with an untrained model. Since the equations of influence functions is defined in the point $\hat{\theta}$ these trials would show if the predictions are actually a true representation of the characteristics of the data or not. If the predictions only hold for some given initial training their merit as evaluation tool is not especially useful. If, on the other hand, they can actually pinpoint the underlying relations between a test image and the training images an experiment with this set up could show that.

One choice of set up for this experiment could be training the model for a given number of iterations n and evaluating the loss in the test image. ($\hat{\theta}$ would be the resulting parameter set, like before). At that point influence functions would be used to make predictions in the impact on that loss form each training image. A number of them would be picked for retraining based on the magnitude of their predictions. The difference would then come in the retraining sessions. Instead of reverting the model back to the parameter set $\hat{\theta}$ like in the retraining experiment described in this report, the model would be reinitialised with random parameter values and then retrained for the same number of steps n as the initial training. The loss difference LD_j could then be taken as $LD_j = L(s_{\text{test}}, \tilde{\theta}_j) - L(s_{\text{test}}, \hat{\theta})$, and the rest of the analysis would be equivalent to the one done in this report. If correlation could be found between the predictions and a loss difference it would indicate that the influence is intrinsic to the data and that the hessian, parameters and other model defining metrics after initial training are simply tools of finding it. If it would not, it would mean that influence functions are able to adequately predict the loss difference only in an environment surrounding $\hat{\theta}$, in that case it could be explained by the training point that is later removed has been detrimental in some path choices taken by the optimisation during initial training.

5.4 Dataset

The big step taken from MNIST to KITTI might have been too much, especially when not scaling up the networks used sufficiently. When approximating the hessian it is not apparent that the desired accuracy has been reached, especially for the more complex network 2.

When annotated according to the rules described in section 3.3.1 some images would lose their features since they would be too small (according to the threshold) after re-sizing. However, the fact that the KITTI object data set contains humans in different configurations resulted in annotations that could be confusing at times. The image in figure 5.2 is one such image where the bicyclist in the foreground is ignored (since as a feature it is a Bicyclist and not a Pedestrian) and the pedestrians

in the background became a few pixels too small to still count as features after down sampling. In reality this will be a false negative for the network since the woman in the foreground has very similar geometrical shape to a person taking a step. It is difficult to say how many false negatives like this example the training data contains, but it is likely that it caused some problems for the classifier. One possible solution to this could be to also allow for bicyclist to be considered pedestrians when generating the data set.

Figure 5.2: Training image with confusing annotation.

5.5 Further research

Given that one error source of the experiments might have been the hessian approximations, it would have been interesting to try using the LiSSA method for this step and see if it would give different results.

Earlier works [12] have shown that the choice of test image can have great impact of the results of the influence experiment. While time limitations puts this outside of the scope of this project, an interesting experiment would have been to run the experiment on several different test images and see if there is something characterising what makes it possible to predict influence from that point of view.

While influence functions have an accessible definition that makes it fairly straightforward to think of interesting applications, many can be difficult to implement in reality. Especially for a Master's Thesis project which is limited both in terms of work force and time scope. Some interesting ideas that have come up during the project still deserve an honourable mention, even if time and other limiting factors made it impossible to fit them under the scope of the project. This section will describe some of these ideas.

5.5.1 Road Segmentation

The road segmentation problem is simply the segmentation problem applied to a topic relevant to the autonomous vehicle industry. The system needs to know what parts of an image is road. Since each pixel of an image is to be classified, this means that each pixel in an image will have an associated loss. Each pixel has a label and the cross entropy is defined in each one. The accuracy of a run is the fraction of correctly classified pixels in the test image. Let's assume that the input data is of

shape nxn and index the pixels by $0 \leq i, j, \leq n-1$. Since the loss is defined in each pixel we will have to (and want to!) see how each of them individually influences the parameters (and then, by continuation, the test loss). The expression from equation 2.9 becomes:

$$\mathcal{I}_{\text{up, params}}(z) = -H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta}) \tag{5.1}$$

$$= -H_{\hat{\theta}}^{-1} \nabla_\theta L(z_{i,j}, \hat{\theta}) \tag{5.2}$$

$$= \left. \frac{\mathrm{d}\hat{\theta}_{\epsilon,z}}{\mathrm{d}\epsilon} \right|_{\epsilon=0} \tag{5.3}$$

This might not seem like a major change in closed form, but in practice $\mathcal{I}_{\text{up, params}}(z)$ just went from having the same shape as the set of the parameters, lets call it $shape(param)$ (which is dictated by network architecture), to having shape $shape(param) * n * n$. Similarly, the gradient of the test loss with respect to the up weighting (equation 2.13) will expand in dimension to:

$$\mathcal{I}_{\text{up, loss}}(z, z_{\text{test}}) = \nabla_\theta L(z_{\text{test}_{i,j}}, \theta) \left. \frac{\mathrm{d}\hat{\theta}_{\epsilon,z}}{\mathrm{d}\epsilon} \right|_{\epsilon=0} \tag{5.4}$$

$$\{\text{Plugging in the value from equation 5.2}\} \tag{5.5}$$

$$= \nabla_\theta L(z_{\text{test}_{i,j}}, \theta)(-1) H_{\hat{\theta}}^{-1} \nabla_\theta L(z_{i',j'}, \hat{\theta}) \tag{5.6}$$

This describes the influence on each pixel i, j in $z_{\text{test}_{i,j}}$ from each pixel in the training image $z_{i',j'}$ (index subscript over training id dropped for simplicity). Since i, j, i', j' all go from 1 to n, the complexity of this will skyrocket. The influence over all pixels from each training image could then be summed up to find the top influential images, and from there, top impact pixels could be identified. Hopefully this would yield clusters of pixels so that the results would make sense to a human. Another option would be to study a single pixel (perhaps wrongly classified) to find out what caused the model to erroneously classify it. This is very similar to earlier works where the influence of individual pixels of the training set was studied, [21].

5.5.2 Earth mover's distance

In earlier works, [12], a logistic regression mode was studied using influence functions. Since this kind of model is a lot simpler compared to a neural network, the definition of influence functions on it will also be simpler. It also has a very useful characteristic; the dot product is part of the closed form expression. Leveraging this, factors of the influence function can be removed to pinpoint which one is responsible for catching which correlation. In particular, the influence functions can be stripped down until only the dot product remains. All this yields nice comparisons which visualises what parts of the influence functions carries which information. Specifically, it gives a measure of how much better influence functions are at predicting the impact of a training point compared to simply making that assumption based on image likeness computed through the dot product.

This is an interesting experiment, but it has two drawbacks when working with neural networks. First of all, the influence functions cannot be peeled down to a dot product. And more importantly, the dot product is a bad measure of image-image likeness. A better one is the Wasserstein, or earth mover's distance. It can be shortly described as the cost of transforming one distribution into another. This is readily interpreted in the context of images if one views them as two dimensional distributions. Has been used in comparison to Euclidean distance, Pearson's Correlation and one more in a k-NN application where it outperformed the other distance measures [14].

A better measure is the earth mover's distance or Wasserstein distance. An interesting experiment could be to compare the influence function predictions to the Wasserstein measure between the training images and the test image. One problem is that evaluating the Wasserstein distance itself is also a complex operation and if it should be taken between the test image and each training image it would be a big experiment in itself.

On a smaller scale, the retraining experiment described earlier in this report could be performed and the Wasserstein distance taken between the chosen test image and the top predicted influencers. If correlation is found between these predictions and the Wasserstein distance it would indicate that image likeness is important for influence functions. If no correlation is found it means that the parameters of the network take into consideration features that are more abstract than the image likeness captured by the measure of earth mover's distance. This would be quite similar to earlier works done with attention maps [18].

6
Conclusion

The results of this project give some indication that influence functions may pick up on characteristics of the training data that affect the classifications made by the model. However, when running the experiment on larger images the clear correlation that could be found when studying MNIST and CIFAR-10 was lost. There are a number of possible explanations for this.

There were clear problems when approximating the IHVP (inverse hessian vector product) when working with larger images. The fact that the network evidently did not go through enough initial training meant both that its classifications were not as good as they might have been, but also that all the assumptions made when using the conjugate gradients methods were not met by the system. The absolute first thing to change with the experiment would be to increase the number of training iterations.

The surprising result where the top 10 predicted influential training images were strongly correlated visually still points to the fact that there is something to be gained by further trying to apply influence functions to applications within the autonomous vehicle industry. Even if the correlation between prediction and loss differences were close to 0, the algorithm managed to find some sort of subset of the training data that it for some reason found interesting. Considering the size of the training data set this can not be accounted to chance.

Finally, it must be said again that the models studied needs to be more complex than those used on data sets such as MNIST and with that comes a computational cost. The experiment described in this report is cumbersome and needs sufficient resources to tackle that.

Bibliography

[1] N. Agarwal, B. Bullins, and E. Hazan. "Second-Order Stochastic Optimization for Machine Learning in Linear Time". In: *ArXiv e-prints* (Feb. 2016). arXiv: 1602.03943 [stat.ML].

[2] J. M. Benitez, J. L. Castro, and I. Requena. "Are artificial neural networks black boxes?" In: *IEEE Transactions on Neural Networks* 8.5 (Sept. 1997), pp. 1156–1164. ISSN: 1045-9227. DOI: 10.1109/72.623216.

[3] Jong Kyu Choi and Yong Gu Ji. "Investigating the Importance of Trust on Adopting an Autonomous Vehicle". In: *International Journal of Human–Computer Interaction* 31.10 (2015), pp. 692–702. DOI: 10.1080/10447318.2015.1070549. eprint: https://doi.org/10.1080/10447318.2015.1070549. URL: https://doi.org/10.1080/10447318.2015.1070549.

[4] R. Dennis Cook and Sanford Weisberg. ""Characterizations of an Empirical Influence Function for Detecting Influential Cases in Regression."" In: *Technometrics* 22.4 (1980), pp. 495–508. DOI: 10.2307/1268187.

[5] Deep Learning 4 Java Documentation. *A Beginner's Guide to Deep Convolutional Neural Networks (CNNs)*. June 2018. URL: https://deeplearning4j.org/convolutionalnetwork.

[6] Saeed Elnaj. *The Uber Accident, Waymo Technology And The Future Of Self-Driving Cars*. June 2018. URL: https://www.forbes.com/sites/forbestechcouncil/2018/05/24/the-uber-accident-waymo-technology-and-the-future-of-self-driving-cars/#6e09c7647814.

[7] Cabitza F, Rasoini R, and Gensini G. "Unintended consequences of machine learning in medicine". In: *JAMA* 318.6 (2017), pp. 517–518. DOI: 10.1001/jama.2017.7797. eprint: /data/journals/jama/936418/jama_cabitza_2017_vp_170094.pdf. URL: +%20http://dx.doi.org/10.1001/jama.2017.7797.

[8] M. Foedisch and A. Takeuchi. "Adaptive real-time road detection using neural networks". In: *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No.04TH8749)*. Oct. 2004, pp. 167–172. DOI: 10.1109/ITSC.2004.1398891.

[9] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[10] A. Ghorbani, A. Abid, and J. Zou. “Interpretation of Neural Networks is Fragile”. In: *ArXiv e-prints* (Oct. 2017). arXiv: 1710.10547 [stat.ML].

[11] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *ArXiv e-prints* (Dec. 2014). arXiv: 1412.6980 [cs.LG].

[12] P. W. Koh and P. Liang. “Understanding Black-box Predictions via Influence Functions”. In: *ArXiv e-prints* (Mar. 2017). arXiv: 1703.04730 [stat.ML].

[13] Scott Lundberg and Su-In Lee. “A unified approach to interpreting model predictions”. In: *CoRR* abs/1705.07874 (2017). arXiv: 1705.07874. URL: http://arxiv.org/abs/1705.07874.

[14] Michael Miller and Jan Van Lent. “Monge’s Optimal Transport Distance with Applications for Nearest Neighbour Image Classification”. In: *CoRR* abs/1612.00181 (2016). arXiv: 1612.00181. URL: http://arxiv.org/abs/1612.00181.

[15] Randall Munroe. *Machine learning*. May 2018. URL: https://xkcd.com/1838/.

[16] Seiichi Murakami et al. “Automatic identification of bone erosions in rheumatoid arthritis from hand radiographs based on deep convolutional neural network”. In: *Multimedia Tools and Applications* (Dec. 2017). ISSN: 1573-7721. DOI: 10.1007/s11042-017-5449-4. URL: https://doi.org/10.1007/s11042-017-5449-4.

[17] Julian D Olden and Donald A Jackson. “Illuminating the “black box”: a randomization approach for understanding variable contributions in artificial neural networks”. In: *Ecological Modelling* 154.1 (2002), pp. 135–150. ISSN: 0304-3800. DOI: https://doi.org/10.1016/S0304-3800(02)00064-9. URL: http://www.sciencedirect.com/science/article/pii/S0304380002000649.

[18] Dong Huk Park et al. “Attentive Explanations: Justifying Decisions and Pointing to the Evidence”. In: *CoRR* abs/1612.04757 (2016). arXiv: 1612.04757. URL: http://arxiv.org/abs/1612.04757.

[19] Reza Rasti, Mohammad Teshnehlab, and Son Lam Phung. “Breast cancer diagnosis in DCE-MRI using mixture ensemble of convolutional neural networks”. In: *Pattern Recognition* 72 (2017), pp. 381–390. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.08.004. URL: http://www.sciencedirect.com/science/article/pii/S0031320317303084.

[20] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “"Why Should I Trust You?": Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: ACM, 2016, pp. 1135–1144. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939778. URL: http://doi.acm.org/10.1145/2939672.2939778.

[21] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”. In: *CoRR* abs/1312.6034 (2013). arXiv: 1312.6034. URL: http://arxiv.org/abs/1312.6034.

[22] M. Szarvas et al. “Pedestrian detection with convolutional neural networks”. In: *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.* June 2005, pp. 224–229. DOI: 10.1109/IVS.2005.1505106.

A

Appendix 1

Top 100 from CG method experiment.

Figure A.1: Training images with top 1 - 5 highest influence.

Figure A.2: Training images with top 6 - 10 highest influence.

Figure A.3: Training images with top 11 - 15 highest influence.

Figure A.4: Training images with top 16 - 20 highest influence.

Figure A.5: Training images with top 21 - 25 highest influence.

Figure A.6: Training images with top 26 - 30 highest influence.

Figure A.7: Training images with top 31 - 35 highest influence.

Figure A.8: Training images with top 36 - 40 highest influence.

Figure A.9: Training images with top 41 - 45 highest influence.

Figure A.10: Training images with top 46 - 50 highest influence.

Figure A.11: Training images with top 51 - 55 highest influence.

Figure A.12: Training images with top 56 - 60 highest influence.

Figure A.13: Training images with top 61 - 65 highest influence.

Figure A.14: Training images with top 66 - 70 highest influence.

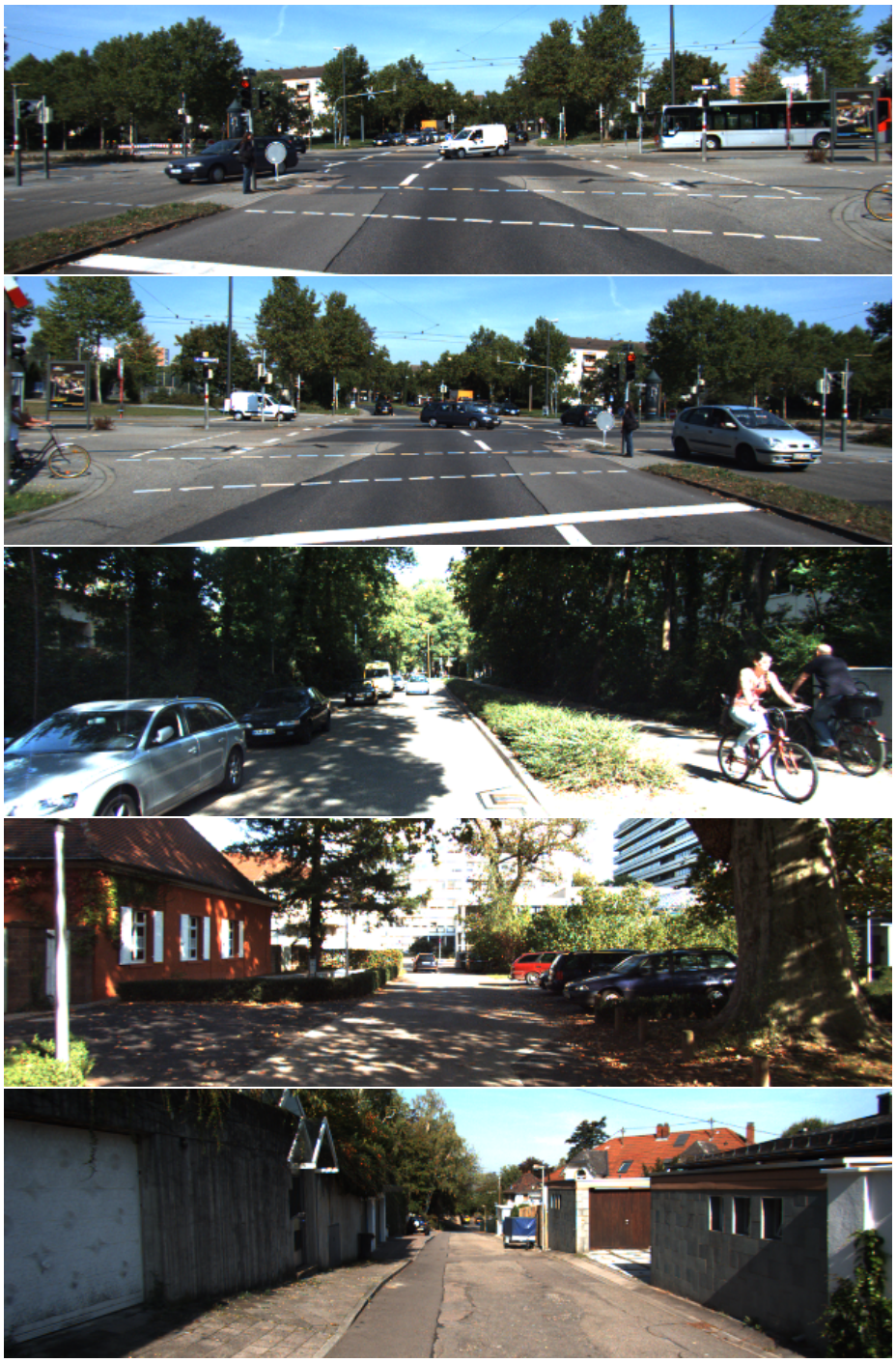


Figure A.15: Training images with top 71 - 75 highest influence.

Figure A.16: Training images with top 76 - 80 highest influence.

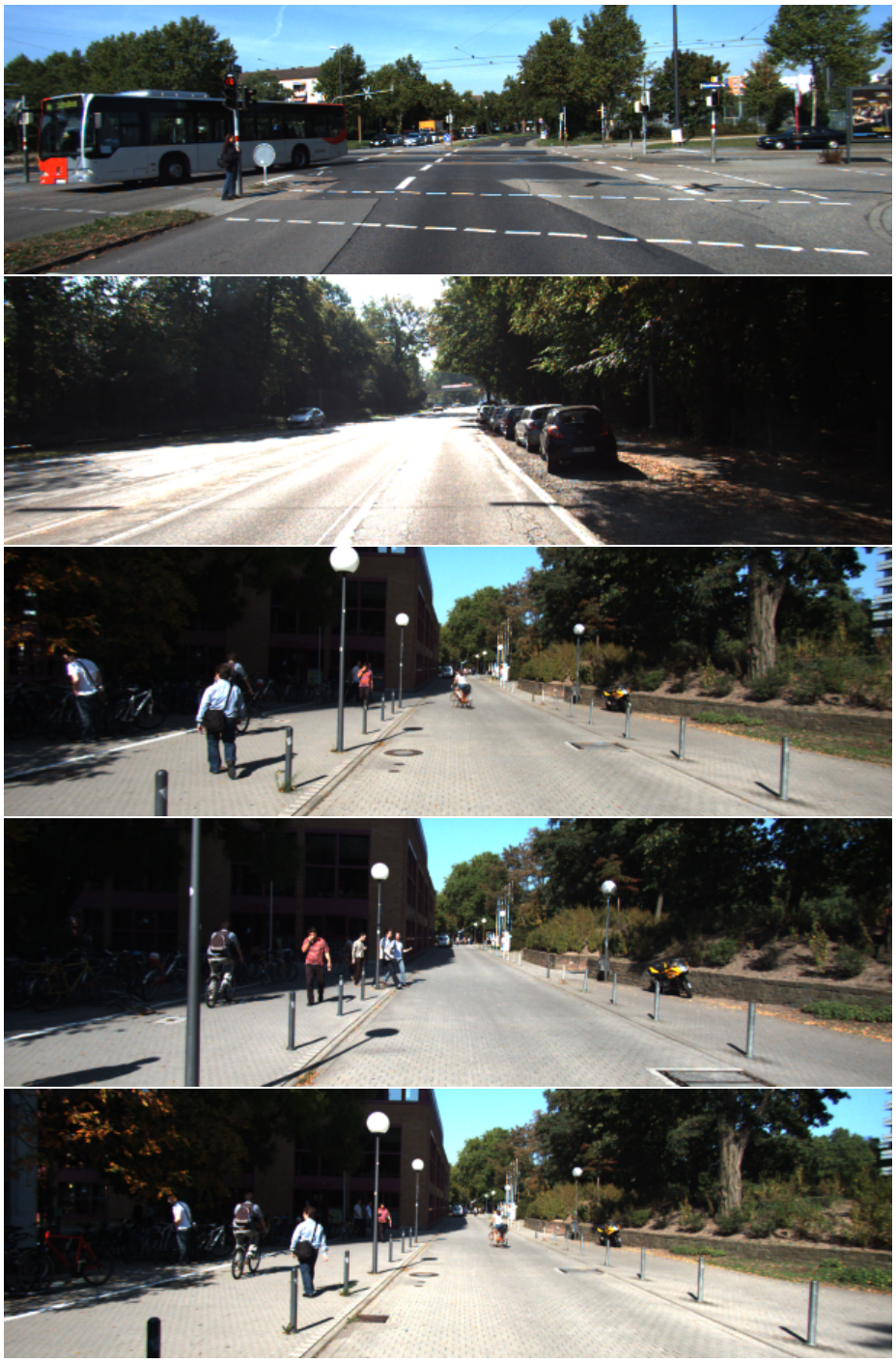


Figure A.17: Training images with top 81 - 85 highest influence.

Figure A.18: Training images with top 86 - 90 highest influence.

Figure A.19: Training images with top 91 - 95 highest influence.

Figure A.20: Training images with top 96 - 100 highest influence.