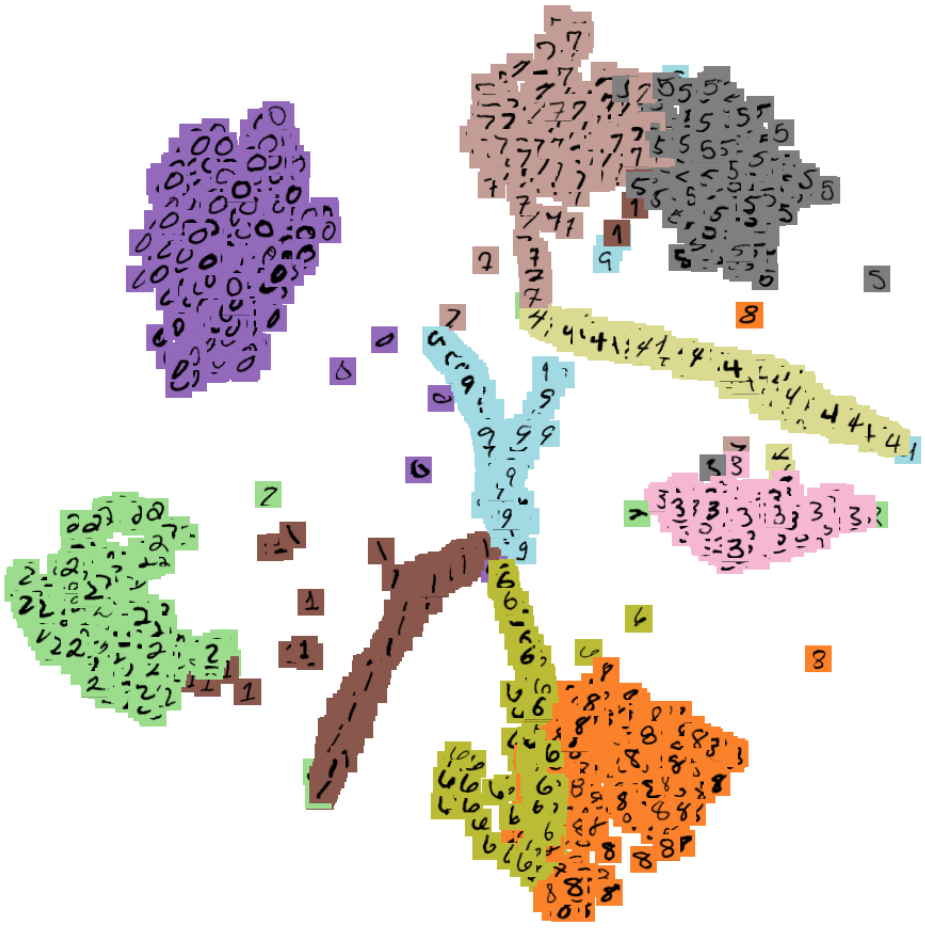
CHALMERS
UNIVERSITY OF TECHNOLOGY



Finding Influential Examples in Deep Learning Models

Master's thesis in Engineering Mathematics and Computational Science and Complex Adaptive Systems

ADAM BRINKMAN
JOHAN LARSSON HÖRKÉN

Departement of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018

Finding Influential Examples in Deep Learning Models

ADAM BRINKMAN
JOHAN LARSSON HÖRKÉN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2018

Finding Influential Examples in Deep Learning Models
ADAM BRINKMAN
JOHAN LARSSON HÖRKÉN

Supervisor: Daniel Langkilde, Annotell
Academic Advisor: Rebecka Jörnsten, Department of Mathematical Sciences
Examiner: Johan Jonasson, Department of Mathematical Sciences

Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: T-SNE visualization of MNIST images as represented in the embedding space.

Typeset in LaTeX
Gothenburg, Sweden 2018

Finding Influential Examples in Deep Learning Models
ADAM BRINKMAN
JOHAN LARSSON HÖRKÉN
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Machine learning models are powerful, but not without errors and the complexity of large models makes it hard for a human to intuitively understand the cause of the error. This thesis approaches the task of explaining predictions made by deep learning models by studying the importance of specific examples in the training data, referred to as *influence*. In practice, the embedding representation of the training data, defined as the output from an arbitrary layer in the model, is compared to the influence on a prediction. Two models are investigated; a Logistic Regression model and a Convolutional Neural Network. The aim of this thesis is thus to identify influential examples in deep learning models in a computationally efficient way, by studying the relation between the representation of the data in a network and its influence. The main results include comparisons between various metrics of distance in the embedding representation of the images to their influence. Similar examples are shown to be clustered close together, training examples close to a test example exhibited high influence for a correctly classified test example. Training examples far away from its class centroid in the embedding space also show high influence.

Keywords: influence, convolutional, network, embedding, features, similarity

Acknowledgements

We would like to thank our supervisor from Annotell, Daniel Langkilde, who has provided continuous support in this thesis work, as well as being an inspiration and sounding board in any situation. We would also like to thank our academic advisor Rebecka Jörnsten and examiner Johan Jonasson from the Department of Mathematical Sciences at Chalmers for great guidance throughout the process. Finally, we would like to thank Chalmers Centre for Computational Science and Engineering for the opportunity to use their high performance computing cluster, enabling our results to reach further.

Adam Brinkman and Johan Larsson Hörkén, Gothenburg, June 2018

Contents

List of Figures

List of Tables

1
Introduction

Modern supervised machine learning models have increased in complexity and performance, allowing them to solve complicated problems related to e.g. computer vision and natural language processing. The model complexity can be a drawback when these models inevitably fail, since it is hard find an intuitive way to explain the cause of the failure and diagnose the model to remedy the problem. The errors might be caused by e.g. insufficient training data, incorrectly annotated data or a poor model. It is therefore important to understand black-box models, not only stir its hyperparameters to minimize test loss, but rather explain its behaviour in an easily interpretable way. Explaining predictions made by a model does not only allow efficient trouble shooting, but also aid human-machine interaction by an increased understanding of the model. Since supervised learning models are defined both by their parameters and training data, it is relevant to understand the influence of training data on a prediction.

There are several approaches to empirically approximate influence of examples from the training data. These approaches are referred to as *perturbation schemes*, since influence analysis is conducted by perturbing the input to a model and measure the effect of that perturbation on the output. An example of such a perturbation scheme is *leave-one-out*, which in practice is equivalent to removing one or several examples from the dataset. The perturbed model is compared to the original model and their differences evaluated. This is a feasible approach, but can be computationally inefficient for models consisting of millions of parameters. For such models, it is of great interest to find ways to calculate the influence in an efficient way. The paper *Understanding Black-box Predictions via Influence Functions* by Pang Wei Koh and Percy Liang [18] presents an efficient implementation of approximating influence by studying parameter change using gradients and Hessian-vector products.

The characteristics of a model are defined by its weights. However, the weights also define the representation of the data inside the network. The representation of data amongst the final layers in a *Convolutional Neural Network* (CNN) can be referred to as *embedding*, which exhibit interesting properties related to similarity between data [11, 15, 38]. This thesis investigates if similarity in embedding relate to influence.

In models that can be understood intuitively, such as a logistic regression model, it is sometimes enough to inspect the data in order to get a sense of which examples have a high influence on a prediction. However, for complex models and data of high dimension, distinguishing influential examples become harder. This thesis focuses on finding influential examples in deep learning models, by reasoning from the results of studying the influences in a logistic regression model. In particular, a CNN was used

to classify handwritten digits from the MNIST dataset. Since the theory applies to any CNN, the results should generalize to other models than the one used to produce the results presented in this thesis.

1.1 Related Work

This thesis has been largely inspired by the paper by Pang Wei Koh and Percy Liang where they ask the question: "*How can we explain the predictions of a black-box model?*" [18]. They propose to trace the model behaviour to its training data trough leave-one-out retraining (explained in Section 3.1). Since this is a computationally expensive procedure not feasible in large scale machine learning settings, they suggest a method of approximating the effect of leave-one-out retraining through *influence functions* (explained in Section 3.2). This thesis replicate and validate their results, then try to find a metric that can approximate the effect of leave-one-out retraining in deep neural networks more effectively than influence functions.
Other studies attempt to identify influential examples of classifiers. Wojnowicz et al. [36] investigated influential examples in large-scale regression models by embedding random projections into the construction of a generalized Cook's distance score [6], their implementation did not extend to CNNs. Kabra et al. [14] explained model behaviour of classifiers by training two models with two training examples labelled as +1 and -1 respectively, then looking at which model most affected a test example. They did this by looking at the distance between examples in *version space* [30], especially which examples were close to a decision boundary, inspiring this thesis to look at distance between examples in the *embedding space* (explained in Section 2.4).
A different approach related to understanding similarity between examples is the study of *Siamese networks*. Siamese networks train two sub-networks to extract features from an input into a feature vector, then optimize the pairwise distance between the feature vectors of examples so that similar examples are close and dissimilar examples are far away. This was originally proposed by Bromley et al. [5] and has since shown great results applied to various image recognition and verification tasks [1, 16, 34]. This method has been extended to use three networks, simultaneously optimizing the distance between one positive example and one negative example to a target example [21]. The approach of optimizing learned metrics between similar and dissimilar examples using a learned metric is commonly referred to as *metric learning* [2]. Metrics learned through feature extraction in final layers of deep neural networks are sometimes referred to as *deep metric learning* [28, 32]. Metric learning approaches are used to improve performance in object recognition tasks rather than used to differentiate which training examples are important to a prediction.
Similarity between examples cannot be faithfully characterized by their input representation in a complex visual feature space such as images. It has been shown that feature extraction in the final layers of deep neural networks contain information about similarity [12, 40]. This representation is sometimes referred to as *deep feature embedding* and has shown great performance in unsupervised and transfer

learning applications by clustering similar objects close together, even if the object type was not included in the original training dataset [11,15,38]. Since the deep feature embedding representation successfully show similarity between data, they may also contain information about the influence between data. This thesis explores the deep feature embedding space to find a relation to influence.

2
Background

Machine learning is often used to solve problems that cannot easily be formalized in an algorithmic way, such as detecting objects in images or interpreting speech. Problems like these are not necessarily hard for humans to solve, but require learning from the environment. For computers to be able to solve problems of this kind they need to learn from examples, similar to humans.

There are different approaches that can make computers learn to some extent. One application of machine learning is classification algorithms, which take an input and outputs a guess of which one of a finite set of classes the input belongs to. A common implementation of this is logistic regression, which will be introduced more in depth in Section 2.1. Because of their simplicity, such models will not always reliably classify complex data such as images. It can however help to give an intuition about results from more complex models.

A different approach of solving the same task, often resulting in a higher prediction accuracy, is *Artificial Neural Networks* (ANNs). This thesis focuses on ANNs that are *deep* in the sense that they are constructed by many layers of trainable parameters. In particular, this chapter aims at explaining the concept of CNNs. Before introducing the theory of CNNs, a short introduction of ANNs is given.

2.1 Classification Using Logistic Regression

Logistic regression models are frequently used to model problems where a dependent categorical variable y can be described using a set of independent variables $\mathbf{x}$. Such models have the advantage that the result is relatively easy to interpret and they achieve great performance for many real life problems. With logistic regression, one seeks to model the probability of an event happening depending on the independent variables.

As in the case with any other regression model, like linear regression, the goal of logistic regression can be described as finding coefficients, $\beta_0, \ldots, \beta_n$, that defines the best fitting model. However, unlike linear regression the output from the model is discrete and not continuous. Despite logistic regression being different in this way, many of the principles from linear regression can be transferred and used to motivate the method. In this thesis, the independent variable is dichotomous, meaning it can take only two values, in this case 0 and 1.

Let y denote the outcome variable and $\mathbf{x}$ denote the independent variables. The assumption made in linear regression is that the expected value of y given $\mathbf{x}$ can be

written as a linear combination of the independent variables

$$E(y|\mathbf{x}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n. \tag{2.1}$$

In logistic regression, the right side of the equation is instead a model of the log-odds of the response being a 1, meaning

$$\log\left(\frac{\pi(\mathbf{x})}{1-\pi(\mathbf{x})}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n, \tag{2.2}$$

where $\pi(\mathbf{x})$ is the probability of $\mathbf{x}$ belonging to a specific class. Rewriting this in terms of the probability, the probability can thereby be written on the form

$$\pi(\mathbf{x}) = \frac{e^{\beta_0+\beta_1 x_1+\cdots+\beta_n x_n}}{1+e^{\beta_0+\beta_1 x_1+\cdots+\beta_n x_n}}. \tag{2.3}$$

2.2 Artificial Neural Networks

This section intend to introduce notation and give a brief introduction to how ANNs were used in this thesis. For an input $\mathbf{x} \in \mathbb{X}$ with corresponding label $y \in \mathbb{Y}$, the goal is to find a function f and a set of optimal parameters $\hat{\theta}$ such that for $\tilde{y} = f(\mathbf{x}; \hat{\theta})$ then $\tilde{y} = y$ [10]. For regression tasks, the label is $y \in \mathbb{R}$. For classification tasks the label $\mathbf{y}$ can be one hot encoded, meaning for an n-class classification problem $\mathbf{y}$ is a vector $\mathbf{y} \in \mathbb{R}^n$. The position in the vector correspond to the class, e.g. the label 4 correspond to the one hot encoding $[0, 0, 0, 0, 1, 0]$ in a 6 class classification problem. In these cases the output $\tilde{\mathbf{y}}$ of a network is usually a probability distribution over the n potential classes $\tilde{\mathbf{y}} \in \mathbb{R}^n$. Unless the network is perfect, there will be an error between $\mathbf{y}$ and $\tilde{\mathbf{y}}$ referred to as the *loss*. The loss can be calculated in various ways, where a commonly used metric is *cross entropy* [27], defined as

$$L(\mathbf{y}, \tilde{\mathbf{y}}) = -\sum_i y_i \log(\tilde{y}_i). \tag{2.4}$$

The parameters θ are usually initialized such that their values are normally distributed. The goal of the ANN is to update the parameters θ to minimize the loss for an arbitrary input $\mathbf{x}$. This can be done by differentiating the loss with respect to the parameters of the network, and updating the parameters using any gradient descent optimization algorithm, such as the *Adam optimization algorithm* [17] or *stochastic gradient descent* [29]. This is referred to as *backpropagation*.
In order to learn a generalized way to connect inputs to labels, the parameters need to be updated for a large amount of examples, this process is often referred to as *training*. In order to efficiently train a model, a dataset of inputs and labels $\mathbf{z}_i = (\mathbf{x}_i, y_i)$ is often divided into non-overlapping sets for training, validation and testing. The training dataset is used in the training where the loss on the training dataset can be calculated continuously and its loss should decrease as the network learn. The validation dataset is not used to train the network, but rather as a reference during training, i.e. to see that the network does not *overfit*. Overfitting occurs when the model learn the specific features of the training dataset too well,

which does not generalize to other data. It is therefore common practice to stop training before overfitting occur. There are various ways to counteract overfitting, referred to as *regularization* where the most common approaches are *dropout*, L_2 and *batch normalization* [27]. When training is completed, the test dataset is used to asses how well the model performs.

2.3 Image Classification Using Convolutional Neural Networks

CNNs have recently gained a lot of publicity showing great performance across a wide variety of tasks [20], even though the technology has existed for quite some time [22].
In its basic form CNNs, as used for image classification, apply multiple combinations of *convolution*, *activation*, *pooling* and *fully connected* operations to an input image in order to extract certain features of the image and classify its content, where one operation is referred to as a layer. The amount and order of these layers depend on the task and varies between different network architectures [23]. The input to a CNN is referred to as *input layer* and the output of the final layer as *output layer*, outputs from intermediate layers are sometimes referred to as *feature maps* [10].
Intuitively, the convolution operation, can be compared to applying various filters to the input image to extract certain features. The convolution operation is commonly followed by an activation function, to introduce nonlinearity in the system in order to extract more information from the image. After a few layers of convolution and activation, it is common to include the *pooling* operation to reduce the dimensionality of the feature map. After multiple layers of convolution, activation and pooling, fully connected layers are applied to classify the extracted features to an output layer. The output layer is usually a class vector with the same length as the number of classes the image can belong to, and the values in the vector is a prediction of the correct class. [23]

2.3.1 Images as Tensors

A vector can be seen as an array of ordered elements, where each element in the array can be identified by its index in the order of appearance. Similarly a matrix can be seen as an array of arrays, or a 2D array, where each element is identified by two indices instead of one. Expanding the same concept to more dimensions, a *tensor* is an array with arbitrary amount of dimensions, or *orders*. A third order tensor with height H, width W and depth D is denoted: $\mathbf{A} \in \mathbb{R}^{H \times W \times D}$. An element (i, j, k) in a third order tensor would therefore be referenced by $\mathbf{A}_{i,j,k}$ with $0 \leq i \leq H$, $0 \leq j \leq W$ and $0 \leq k \leq D$ [10, 37].
The tensor notation can be used to represent images. A black and white image with width W and height H can be seen as a matrix where each position contains the intensity for a pixel, which can be represented as a second order tensor $\mathbf{A} \in \mathbb{R}^{H \times W}$. A color image with channels for red, green and blue (RGB) can be represented by

a third order tensor $\boldsymbol{A} \in \mathbb{R}^{H \times W \times 3}$, where the third order represent the channels for the different colors. [10, 37]
Furthermore, a set of color images can be represented as a fourth order tensor, where the fourth order represent the amount of images in the set. A set of N RGB images would therefore be represented as $\boldsymbol{A} \in \mathbb{R}^{H \times W \times 3 \times N}$. [10, 37]
It may sometimes be useful to convert a higher order tensor into a lower order tensor. This operation is following a predefined order, for instance the second order tensor A is converted to a first order tensor as follows

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad vec(\boldsymbol{A}) = (1, 3, 2, 4)^T = \begin{bmatrix} 1 \\ 3 \\ 2 \\ 4 \end{bmatrix} \tag{2.5}$$

using the same order as the `ravel`[1] and `flatten`[2] operators in Numpy[3] which is a Python package commonly used for scientific computing. The vectorization notation *vec* is commonly used in mathematics to convert a higher order tensor to a vector (order 1 tensor). For vectorization of an order 3 tensor, the first channel is vectorized first in the same way as the above example, then the second channel etc., till all channels are vectorized. The vectorization of the order 3 tensor is the concatenation of the channels in this order. A similar approach can be taken for even higher order tensors. [37]

2.3.2 Convolution

Convolution is a mathematical operation on two functions, formally defined as

$$s(t) = \int x(\tau) w(t - \tau) d\tau \tag{2.6}$$

for a continuous signal, commonly denoted by an asterisk

$$s(t) = (x * w)(t). \tag{2.7}$$

The first function x is usually referred to as the *input*, the second function w as the *kernel* and the output $s(t)$ is sometimes referred to as a *feature map*. It is often practical to work with discrete time, where the time index t can then only take integer values. The discretized version of the convolution operation is defined as

$$s(t) = (x * w)(t) = \sum_{\tau=-\infty}^{\infty} x(\tau) w(t - \tau), \tag{2.8}$$

which is useful in machine learning applications since the inputs commonly are discrete multidimensional arrays of data and the kernel a multidimensional array of learning parameters which are updated by learning algorithms. [10]
Discrete convolutions can be computed using matrix multiplications, which is useful in order to utilize the efficiency of GPUs when performing large scale computations [9]. This can be done for instance by using the Toeplitz matrix [3].

[1] `docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.ravel.html`
[2] `docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.ndarray.flatten.html`
[3] `http://www.numpy.org/`

Convolution of Higher Order

It is sometimes useful to perform the convolution operation for multiple axes simultaneously, for instance a two dimensional (black and white) image $\boldsymbol{I} \in \mathbb{R}^{H \times W}$ convoluted with a two dimensional kernel $\boldsymbol{K} \in \mathbb{R}^{H' \times W'}$, where $H' \leq H$ and $W' \leq W$, is defined as

$$\boldsymbol{S}_{i,j} = (\boldsymbol{I} * \boldsymbol{K})_{i,j} = \sum_m \sum_n \boldsymbol{I}_{m,n} \boldsymbol{K}_{i-m,j-n}. \tag{2.9}$$

Since convolution is commutative, it is possible to rewrite the operation for easier implementation in machine learning applications as

$$\boldsymbol{S}_{i,j} = (\boldsymbol{K} * \boldsymbol{I})_{i,j} = \sum_m \sum_n \boldsymbol{I}_{i-m,j-n} \boldsymbol{K}_{m,n}. \tag{2.10}$$

The same concept applies to higher order convolutions, where the input may be sets of images with multiple color channels convoluted with a multidimensional kernel. For instance, the convolution of the input $\boldsymbol{I} \in \mathbb{R}^{H \times W \times D}$ with the kernel $\boldsymbol{K} \in \mathbb{R}^{H' \times W' \times D \times D'}$ would have output size $\boldsymbol{S} \in \mathbb{R}^{(H-H'+1) \times (W-W'+1) \times D'}$. [10, 37]

Padding and Stride

As seen in the above example, using a kernel size larger than 1 spacial coordinate (1×1 in the 2D case) reduces the size of the output in comparison to the input. For some applications it may be preferable to keep the same input and output dimension in the convolution operation. This can be solved using *padding*, which extends the input size with zeros around the boarder of the input. The output size of the convolution operation can be calculated as

$$O = \frac{W - K + 2P}{S} + 1, \tag{2.11}$$

where $O \in \mathbb{N}$ is the output side length, W is the input side length (assuming equilateral input, $W = H$), P is the padding and S is the *stride length* [23]. The stride length is the amount of steps the convolution kernel is moved between each operation. [10, 37]

Activation Function

Each convolution layer is commonly followed by an activation function to introduce nonlinearity in the system, since the convolution operation is linear [26]. Nonlinear functions such as the hyperbolic tangent function and sigmoids were commonly used for this in the past, but *Rectified Linear Units*, abbreviated *ReLU*, were found to train faster without a significant decrease in performance [20]. ReLU implements the rectifier function $f(x) = max(0, x)$ to the output of the convolution.

2.3.3 Pooling

To reduce spatial size of the feature representation of the input inside of the network it is common practice to insert a *pooling layer* in-between successive convolutional

layers. Reducing the spatial size help reduce the amount of parameters and computation needed in the network which also serves to reduce overfitting [20, 23] and build robustness to small distortions in the input [13]. A commonly used pooling operation is *max-poling*, which similar to convolution can be thought of as iteratively applying a filter to sub-regions of the input, but instead of performing a computation simply choosing the maximum value of the sub-region as output [23]. Thus effectively downsampling the input without requiring any learning parameters. The stride length of the pooling operation is commonly set to the same as the filter size [13], but studies have shown that models trained with overlapping pooling in some cases are more resistant to overfitting [20].
It has been shown that pooling layers can be replaced by convolutional layers with increased stride length to reduce spatial size, without significant loss in performance. This approach strive for a simpler network architecture comprising solely of convolutional layers, which have shown competitive or state of the art results on several object recognition datasets. [33]

2.3.4 Fully Connected

Fully connected layers may be applied after a number of convolutions and pooling layers to classify the downsampled information into the final predictions [22]. In fully connected layers, each neuron has connections to all activations in the previous layer, similar to traditional ANNs as introduced in Section 2.2 [23, 35]. The dense connections in the fully connected layers make these computationally expensive to train [22, 23].
In order to reduce the computational complexity of the fully connected layers, attempts has been made to replace the fully connected layers with convolutional layers. It has been shown that the image area covered by topmost convolutional layer contain enough information to recognize the content it is trying to predict. The fully connected layers can then be replaced by simple 1-by-1 convolutions, maintaining high accuracy in object detection applications. [33]
The final layer in a network is commonly referred to as the *output layer* [23]. The output layer can be used as input to the *softmax function* [4], which generates the estimated posterior probability for each class in a classification setting [37]. Formally, the softmax function takes a K-dimensional vector $\mathbf{z}$ as input and outputs a K-dimensional vector $\sigma(\mathbf{z})$ of real values, where $\sigma_j > 0 \ \forall j$, $\sum_{j=1}^{K} \sigma_j = 1$ and $\sigma_j = P(class = j|input)$, by

$$\sigma_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \tag{2.12}$$

for $j = 1, ..., K$ [4].

2.3.5 Putting it All Together

When building a convolutional neural network, multiple rounds of convolutions, ReLU, pooling and fully connected layers are commonly stacked together in order to gain a good prediction of the problem it is trying to solve. The result is a network

of trainable weights, which requires tuning. When feeding an input through a CNN, consisting of randomly initialized weights, one usually obtains unsatisfactory results. In order for the network to learn a sufficient representation it needs to be trained using known examples. The predictions from the network are then compared to the desirable result, commonly using some kind of loss function, e.g. cross entropy, and the parameters in the network updated to minimize this loss function, using a gradient descent optimization algorithm. [37]
There are various ways to implement network architectures, loss functions and optimization algorithms, as well as various hyperparameters that may be tuned to improve the performance of the network, but these are not the topic of this thesis.

2.4 Embedding Space

A famous embedding example is the *word2vec* parameter learning model, which produce word embeddings in a vector space with a large corpus of text as input. These learned embeddings exhibit interesting semantic properties, such as the relation between a city and the country it belongs to, e.g. France is to Paris as Germany is to Berlin in the embedding space. Furthermore, simple algebraic operations can be applied to the embedding vectors of words, e.g. $X = vector(\text{"}biggest\text{"}) - vector(\text{"}big\text{"}) + vector(\text{"}small\text{"})$. When searching the embedding space to the closest word to X measured by cosine distance, it is possible to find the word *smallest* in a well trained model. [25]
Embedding in the context of this thesis refers to a feature map $f(x) \in \mathbb{R}^d$ from the output of any layer in a deep convolutional neural network for an input image x, where $d \in \mathbb{N}$ is the size of the embedding. This is sometimes referred to as *deep feature embedding* [12], *deep embedding* [38] or *deep features* [40], but is referred to as *embedding* in the context of this thesis and the space that is made up of multiple embeddings as *embedding space*. An embedding can be extracted between any two layers in a CNN, which exhibits different properties depending on where the embedding is extracted. In an image classification setting, the embedding present in the final layers need to be able to separate data of different classes in order to correctly classify objects [11].
Various studies have shown that it is possible to learn relations between images in the embedding space, which has been used to achieve prominent results in various object recognition tasks. These approaches are commonly referred to as *metric learning* [2,12]. Metric learning operates by minimizing the distance between examples of the same class and maximizing distance between examples of different class in the embedding space [2]. Distance is measured using various metrics. A metric commonly used to compare high dimensional data is *cosine similarity*, defined as

$$d(\mathbf{p}, \mathbf{q}) = cos(\theta) = \frac{\mathbf{p} \cdot \mathbf{q}}{\|\mathbf{p}\| \, \|\mathbf{q}\|}, \tag{2.13}$$

which measures angular relation [15]. A cosine similarity of 1 corresponds to two vectors being parallel. Sometimes it makes sense to relate distance to a distribution. *Mahalanobis distance* uncorrelate data from its distribution before calculating

Euclidean distance, formally defined as

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(\mathbf{p} - \mathbf{q})^T \mathbf{S}^{-1} (\mathbf{p} - \mathbf{q})}, \tag{2.14}$$

where $\mathbf{S}$ is the covariance matrix of the data points [2].
Depending on which metric is used, and how the model is trained, examples will exhibit different relationships in the embedding space. Studies have shown that models trained on classification of large datasets such as ImageNet [8] manage to cluster previously unknown examples in the embedding space. This show that these models manages to learn semantic relationships between similar data points in the embedding space, even if not explicitly trained to optimize a certain metric between training points [11, 38].

3
The Study of Influence

In statistics in general, and regression analysis in particular, it is common to talk about *outliers* and *leverage points*. Their definitions can be used to reason about the meaning of an example being influential or not. Both outliers and leverage points are closely related to influential examples, in the sense that both has the potential of being influential. The difference between these definitions and the intuition of influence can easily be explained using the illustrative example below, inspired by [39].
In Figure 3.1, four models are fitted to the data using linear regression. For each of the individual models except one, an example from the set $\{A, B, C\}$ is added. A, B and C can be considered to be outliers since they are positioned far away from the rest of the data, but not all of the examples are influential. B is located far away from the solid line and as can be seen in the figure it also makes the slope of the line change significantly when added to the data set, thus also the parameters of the model. This is equivalent to B being influential [39].

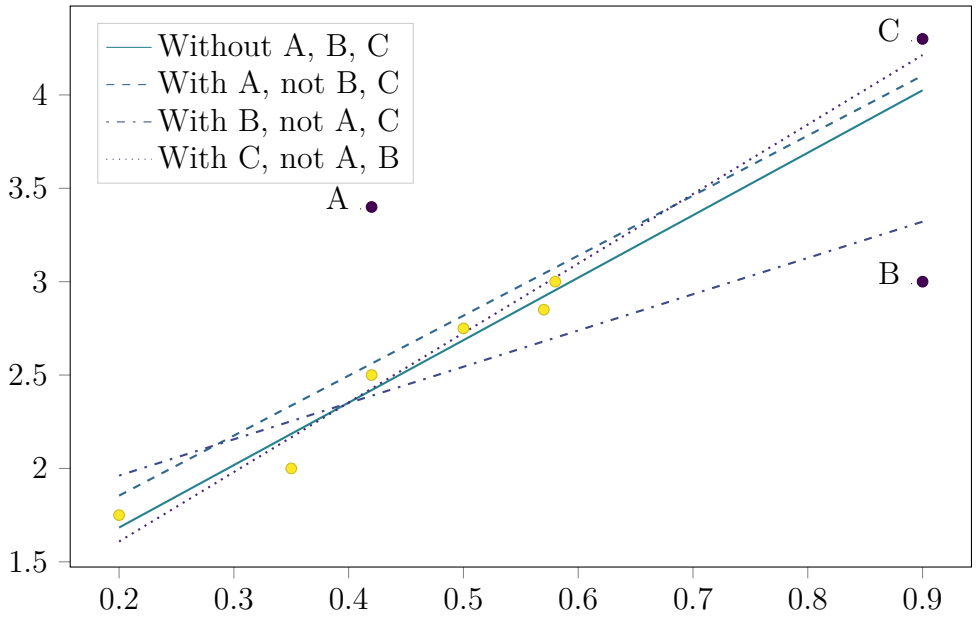


Figure 3.1: The best fitting regression models, found using ordinary least squares, for the dataset containing all yellow points in the scatter plot, and either one of A, B or C, or none of them.

The study of influence can be summarized as studying the variation in results depending on how the dataset is changed. An example of this is temporarily deleting

a data point from the dataset and then retraining the model, here referred to as *leave-one-out* retraining. Such a change may change the outcome of the experiment completely, but may also have almost no effect on the outcome at all. In other words, not all examples in a dataset are equally important. This motivates the fundamental concept of influence analysis. Influence is not necessarily measured by doing leave-one-out retraining, but could also be studied by e.g. moving or down-weighting examples. The result depends heavily on the perturbation scheme, as well as the method of measurement. However, as leave-one-out is commonly used in the field of influence analysis, this study is limited to using leave-one-out as the ground truth of influence. [7]

3.1 Leave-one-out

By introducing small perturbations in the dataset, it is possible to draw conclusions about the influence of the data on the model. The rest of this section aim to develop the notation for the specific case of studying the influence by leave-one-out retraining in deep learning models.
Following previously introduced notation, let $\mathbf{x} \in \mathbb{X}$ denote the input from an input space $\mathbb{X}$ and $y \in \mathbb{Y}$ denote the output from an output space $\mathbb{Y}$. The training data given to the model is denoted $\mathbf{z}_1, \ldots, \mathbf{z}_n$, where $\mathbf{z}_i$ consists of an input and its label, meaning $\mathbf{z}_i = (\mathbf{x}_i, y_i) \in \mathbb{X} \times \mathbb{Y}$. The optimal model parameters, $\hat{\theta}$, are found by minimizing the empirical risk, defined as the mean of the loss, $L(\mathbf{z}, \theta)$, for each training point

$$\hat{\theta} \coloneqq \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L(\mathbf{z}_i, \theta). \tag{3.1}$$

To measure the change in the model when a training point $\mathbf{z}$ is removed, one approach is to measure the change in its parameters. The total change of the model parameters can be denoted

$$\Delta\hat{\theta} \coloneqq \hat{\theta}_{-\mathbf{z}} - \hat{\theta}, \tag{3.2}$$

where $\hat{\theta}_{-\mathbf{z}} \coloneqq \underset{\theta}{\operatorname{argmin}} \sum_{\mathbf{z}_i \neq \mathbf{z}} L(\mathbf{z}_i, \theta)$ are the optimal parameters for the reduced dataset, not containing $\mathbf{z}$.
The intuitive approach to measuring leave-one-out influences would be to remove $\mathbf{z}$ from the data set, then retrain the model and measure the change in its parameters. However, retraining the model for each $\mathbf{z}$ is computationally heavy for large learning models and large data sets.

3.2 Influence Functions

Influence functions, as implemented in [18], aim to approximate leave-one-out retraining in a computationally efficient way. The implementation makes use of the idea that the effect of a training data point on a certain prediction can be studied by

increasing the *importance* of that specific training data point, meaning its weight in the loss function, and note how the parameters of the model change. If the increase in importance is defined as ϵ, removing a point $\mathbf{z}$ corresponds to upweighting it by $\epsilon = -\frac{1}{n}$. The parameters of the model with the importance of z being up-weighted is defined as

$$\hat{\theta}_{\epsilon,z} \coloneqq \left(\underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L(\mathbf{z}_i, \theta) + \epsilon L(\mathbf{z}, \theta) \right). \tag{3.3}$$

The influence on the parameters of a model is defined as the derivative of the parameters of the model, $\hat{\theta}_{\epsilon,z}$, with respect to ϵ

$$I_{up,params}(\mathbf{z}) \coloneqq \left. \frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon} \right|_{\epsilon=0} = -H_{\hat{\theta}}^{-1} \nabla_\theta L(\mathbf{z}, \hat{\theta}), \tag{3.4}$$

where $H_{\hat{\theta}} \coloneqq \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta^2 L(\mathbf{z}_i, \hat{\theta})$ is the Hessian, which is a function of the model parameters. $I_{up,params}$ describe how the parameters of the model change with respect to up-weighting the importance of a specific data point, and can also be used to derive the change in the loss of the model. The change in loss on $\mathbf{z}_{test}$, with respect to changing the importance of a point $\mathbf{z}$ is given by

$$I_{up,loss}(\mathbf{z}, \mathbf{z}_{test}) = -\nabla_\theta L(\mathbf{z}_{test}, \hat{\theta})^T H_{\hat{\theta}}^{-1} \nabla_\theta L(\mathbf{z}, \hat{\theta}). \tag{3.5}$$

Since the verification of these results are not the main focus of this thesis, the derivation is not included. See [18] for a more in depth derivation.
The number of operations needed to calculate and invert $H_{\hat{\theta}}$ quickly grows large for models made up of many parameters. A solution to this presented in [18], is that $I_{up,loss}$ can be calculated more efficiently by instead approximating $s_{test} =: H_{\hat{\theta}}^{-1} \nabla_\theta L(\mathbf{z}_{test}, \hat{\theta})$ and then using the same approximation to calculate

$$I_{up,loss}(\mathbf{z}, \mathbf{z}_{test}) = -s_{test} \cdot \nabla_\theta L(\mathbf{z}, \hat{\theta}), \tag{3.6}$$

for all $\mathbf{z}_i$. The two methods discussed in [18] are referenced as the *conjugate gradients* method and *stochastic estimation*. Both methods are based on the assumption that the empirical risk is twice differentiable and strictly convex in θ, which implies that $H_{\hat{\theta}}$ is positive definite.
This is not the case when obtaining the parameters $\hat{\theta}$ for non-convex objectives or when using stochastic gradient descent with early stopping, resulting in $H_{\hat{\theta}}$ having negative eigenvalues. It was suggested in [18] to use a convex quadratic approximation of the loss around $\hat{\theta}$. They did this by adding a damping term when calculating the loss if $H_{\hat{\theta}}$ has negative eigenvalues, corresponding to adding L_2 regularisation on θ.

3.2.1 Conjugate Gradients

Conjugate gradients (CG) utilize the fact that $H_{\hat{\theta}}$ is positive definite. Calculating $H_{\hat{\theta}}^{-1} v$ is then equivalent to minimizing $\frac{1}{2} t^T H_{\hat{\theta}} t - v^T t$ with respect to t, where v is an arbitrary vector. The downside of this method is that it is still computationally heavy for large models. However, it can be used to find an exact solution to the minimization problem stated above. [24]

3.2.2 Stochastic Estimation

Unlike CG, approximating s_{test} using stochastic estimation does not require using all examples from the dataset, but samples a single example per iteration, and is therefore a faster alternative for large datasets. Denoting $H_\theta = H$, T as being the number of samples from the dataset and R as the number of averages of the estimate, stochastic estimation can be formalized as in Algorithm 1. The method was named *LiSSA* (Linear time Stochastic Second-Order Algorithm). [18]

Algorithm 1 Stochastic estimation

$\tilde{H}_{est}^{-1} v = 0$
for r = 1 to R **do**
    $\tilde{H}_0^{-1} v \leftarrow v$
    for t = 1 to T **do**
        Sample $\mathbf{z}_s$ uniformly from the training data
        $\tilde{H}_t^{-1} v \leftarrow v + \left(I - \nabla_\theta^2 L(\mathbf{z}_s, \hat{\theta})\right) \tilde{H}_{t-1}^{-1} v$
    end for
    $\tilde{H}_{est}^{-1} v \leftarrow \tilde{H}_{est}^{-1} v + \tilde{H}_T^{-1} v$
end for
$\tilde{H}_{est}^{-1} v \leftarrow \tilde{H}_{est}^{-1} v / R$
return $\tilde{H}_{est}^{-1} v$

4
Method

This chapter present the various experiments conducted in order to examine influence, as well as the models and data used to obtain them. Two models were chosen and this chapter focuses on the setup used for those models, even though the approaches described generalize to other models and datasets. First a logistic regression model was used to verify the performance of influence functions and give an intuition about influential data in a way that can be easily visualized. Then a CNN was implemented in order to further examine how the embedding space relate to influence in deep learning models. Different datasets were used for the logistic regression and CNN model, both datasets were divided into subsets for training, validation and testing.

The code developed by [18], which was written in Python[1], was used as a starting point for the experiments. Python is an open-source programming language commonly used in machine learning and data science. Several Python frameworks were used to obtain the results, including the machine learning framework TensorFlow[2] and various Scipy[3] packages.

4.1 Evaluating Influence Using Logistic Regression

The purpose of implementing the logistic regression model was to easier reason about which training points in a dataset that might be influential outliers for a simple model, and then extend the concepts from the simple model to more complex models. This was done by plotting the synthetic dataset together with the influence of each example.

The model was implemented using the `LogisticRegression` method from a set of methods called Generalized Linear Models implemented in the Python package Scikit-learn[4]. `lbfgs` was used as the solver, tolerance was set to 10^{-8} and maximum iterations was set to $1,000$. The parameters were chosen so that the model converged to a *sufficiently* good solution, in the sense that it could be used in the experiments, while still being fast to calculate.

[1] `https://www.python.org/`
[2] `https://www.tensorflow.org/`
[3] `http://www.scipy.org/`
[4] `http://www.scikit-learn.org/`

4.1.1 Leave-one-out Retraining Experiment

In order to measure the quality of the results, leave-one-out retraining was defined as the ground truth of influence. However, as stated in in Chapter 3, the choice of perturbation scheme is arbitrary and it is not obvious that leave-one-out is the best choice.
For the logistic regression model, leave-one-out retraining was conducted by first training the model on the entire training dataset until convergence, meaning that either the tolerance criterion was fulfilled or that maximum iterations was exceeded. The loss $L(\mathbf{z}_{test}, \hat{\theta})$ of the test example was then calculated using the model with the optimal parameters $\hat{\theta}$. Thereafter, the weights of the model were reinitialized and retrained using the same stopping criteria, but on a subset of the training dataset containing all examples but $\mathbf{z}_i$. The leave-out-out influence for $\mathbf{z}_i$ was calculated as the change in loss using the optimal parameters for the two different models

$$I_{LOO} =: L(\mathbf{z}_{test}, \hat{\theta}) - L(\mathbf{z}_{test}, \hat{\theta}_{-\mathbf{z}_i}). \tag{4.1}$$

4.1.2 Influence Functions

Influence functions, as implemented by [18] approximate leave-one-out retraining. It was implemented and used to find the influence of every data point in the synthetically generated dataset, in order to understand if influence functions is a good approximation of leave-one-out retraining.
The implementation of the influence functions experiment make use of many features of TensorFlow. One important feature is that it makes calculating gradients easy. The experiment boils down to estimating the inverse Hessian vector product. For the case of logistic regression, estimating the inverse Hessian vector product is not computationally heavy since the model consist of very few variables, which means either CG or LiSSA could be used. Once the estimate of the inverse Hessian was calculated, the predicted influences were calculated one example at a time by calculating the dot product $-s_{test} \cdot \nabla_\theta L(z, \hat{\theta})$, as described in Equation 3.6.

4.1.3 Dataset

The synthetic dataset was generated with interpretability of the logistic regression model in mind. In other words, the goal was to generate data which could easily be understood simply by inspecting it visually and at the same time be modeled using a logistic regression model. The dataset contains datapoints each consisting of two independent variables, x_1 and x_2, and a one-hot encoded label $\mathbf{y}$ depending on the two variables.
The dataset was generated as two distinct point clusters, where the values of the mean of the clusters was simulated using bivariate normal distributions with equal diagonal covariance matrices but different means. The response variable Y was then generated as a Bernoulli random variable taking the value 1 with probability $\pi(\mathbf{x})$ and 0 with probability $1 - \pi(\mathbf{x})$, where

$$\pi(\mathbf{x}) = 1/(1 + \exp(-(1 + \beta_1 x_1 + \beta_2 x_2))) \tag{4.2}$$

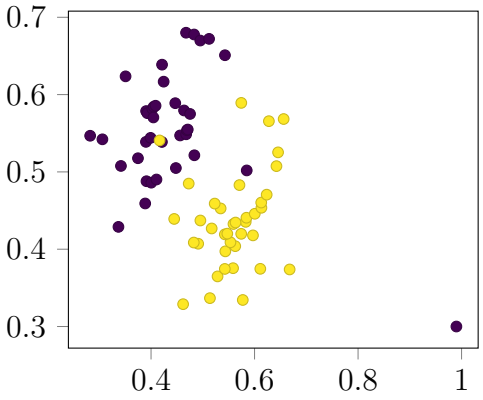


Figure 4.1: A dataset consisting of two distinct clusters of data belonging to two different classes, as well as an extreme oulier.

is the sigmoid function. Lastly, an outlier was added to the dataset. The resulting dataset is presented in Figure 4.1.

4.2 Evaluating Influence Using a Convolutional Neural Network

A CNN was implemented to predict the label of images depicting handwritten numbers in a 10-class classification setting of the MNIST dataset, described further below. The images were used as input to the network and the output was a vector of length 10 with the values in each position corresponding to the certainty that the input belongs to corresponding class, where a higher value corresponds to a higher certainty. The final prediction was obtained by the argmax of the output vector, resulting in an integer between 0 and 9.

4.2.1 MNIST Dataset

MNIST [22] is a well known dataset containing grayscale images of handwritten digits from 0 to 9. The database consists of 70,000 images of size $28 * 28$ pixels, which has been split into a training set containing 60,000 examples, and a test set containing 10,000 examples. It originates from NIST's database, but was slightly modified and mixed to better fit as a dataset for applying pattern recognition methods and image processing techniques. To reduce training time, only a subset consisting of 5,500 training examples from the MNIST dataset was used. Figure 4.2 show an example from each class of the MNIST dataset.

4.2.2 All Convolutional Network

To allow for efficient training and retraining of the model, the *All Convolutional Net* (all-CNN) [33] model was used. The architecture is implemented in the same way as suggested by Pang Wei Koh and Percy Liang: "*The network had 7 sets of convolutional layers with tanh(·) non-linearities, modeled after the all-convolutional*

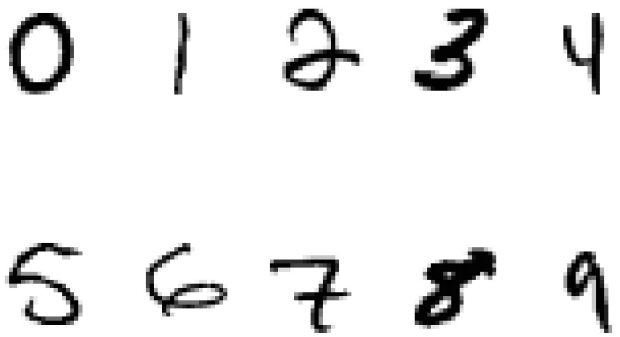


Figure 4.2: Example digits from the MNIST dataset containg one example from each class.

network from [33]. For speed, we used 10% of the MNIST training set and only 2,616 parameters, since repeatedly retraining the network was expensive. Training was done with mini-batches of 500 examples and the Adam optimizer [17]. The model had not converged after 500k iterations; training it for another 500k iterations, using a full training pass for each iteration, reduced train loss from 0.14 to 0.12." [18] This enabled convenient comparison of results.
The comparatively low amount of parameters in the model contributed to fast training of the model. Training the model for 500,000 iterations took about 2-3h on a Tesla K40m GPU. Embedding dimensions as extracted after each layer in the all-CNN architecture can be seen in Table 4.1. The learning rate was updated with a factor 0.1 after 10,000 iterations and 0.01 after 20,000 iterations. L_2 regularisation was implemented to counteract overfitting and cross entropy was used as loss function. The values of all hyperparameters used in the implementation are listed in Table 4.2.

4.2.3 Leave-one-out Retraining Experiment

Leave-one-out retraining was implemented similarly to the approach described in Section 4.1.1. Due to the non-convexity of the system, the model could not be retrained with parameter reset. Retraining was instead implemented by training 30,000 additional iterations from the checkpoint of the trained model (trained for 500,000 iterations). After each example had been left out and trained for 30,000 iterations, the parameters of the model were reset to the checkpoint of the trained model.
Since leave-one-out retraining is very computationally inefficient, it would take a very long time to calculate the influence for the entire training dataset. Therefore, influence functions as described below were used to find a prediction of the 500 most influential datapoints (250 with highest positive and 250 with highest negative pre-

Table 4.1: Embedding dimensions for layers of the all-CNN architecture applied to the MNIST dataset. The first element refer to the amount of examples in the batch n. Elements two and three are the dimensions of the convolution kernel and element four refer to the amount of filters. The $h3_d$ layer is the average over the filter dimensions followed by a fully connected layer producing the *logits* output, these two layers thus have lower dimensions.

Layer	Dimensions
$h1_a$	$[n, 26, 26, 8]$
$h1_c$	$[n, 12, 12, 8]$
$h2_a$	$[n, 10, 10, 8]$
$h2_c$	$[n, 4, 4, 8]$
$h3_a$	$[n, 2, 2, 8]$
$h3_c$	$[n, 2, 2, 10]$
$h3_d$	$[n, 10]$
$logits$	$[n, 10]$

Table 4.2: Hyperparameters used in the All CNN architecture implemented in the experiment.

Parameter	**Value**
Initial learning rate	0.001
L_2 weight decay	0.001
Damping	0.01
Input side length	28
Input channels	1
Input dimensions	784
Layer 1 hidden units	8
Layer 2 hidden units	8
Layer 3 hidden units	8
Convolution patch size	3

dicted influence). These 500 points were then used to calculate the actual influence through leave-one-out retraining, which took about 3 days on a Tesla K40m GPU.

4.2.4 Influence Functions Experiment

The CG implementation introduced in Section 3.2.2 was used to calculate the predicted influence for the all-CNN model. A damping factor (see Table 4.2) was introduced to account for the non-convergence of the model, as described in Section 3.2. Other than that, the implementation followed the same approach as described in Section 4.1.2.

4.2.5 Embedding Experiment

The embedding representation of the input images were extracted from the output of the layers in the all-CNN model. The model consisted of 7 convolutional layers and one spatial average, resulting in 8 representations of the input throughout the network. The embedding representation of the images were obtained by feeding images through the network and extracting the embedding values for all images in the set simultaneously. This resulted in 8 tensors for each example, with dimensionality depending on the output shape from each layer. The shapes of the outputs are summarized in Table 4.1. In layers where the embedding for a single example was not an order one tensor, the tensor was vectorized into an order one tensor.

Metrics

Different metrics were used to evaluate the embedding representation of the images. Cosine similarity was used to measure distance between the embedding representation of train and test images, as well as between data points and cluster centroids of classes.
Measuring distances between the pixel values of images is referred to as distances in the *pixel space*. Where the images are vectorized and distance measured by cosine similarity between image vectors.
Another metric that was used is the *centroid relationship*, which account for the relation between a train and test example. The distance measured by cosine similarity from a train image to the cluster centroid of its corresponding class in the embedding space is denoted as $d_i^{train-cluster}$. The distance from a test image to the cluster centroid belonging to the class of a specific train image is referred to as $d_i^{test-cluster}$. The centroid relationship is defined for a training and a test example as: $d_i^{test-cluster}/d_i^{train-cluster}$.
Furthermore, Mahalanobis distance as introduced in Section 2.4 was used to find outliers in the class clusters taking the variance of the data into account.

TensorBoard Projector

The TensorBoard projector is a tool in TensorFlow that can be used to visualize data. In order to get a intuition about the layout of the embedding space, the TensorBoard projector was implemented to visualize the representation of the images

Table 4.3: ANN model architecture and hyperparameters used to predict influence on a test point given the embeddings and labels of test points.

Parameter	**Value**
Layers	5
Nodes per layer	100
Activation	ReLU
Kernel initializer	Uniform
Loss	Mean absolute error
Optimizer	Adam
L_2 regularization	0.0001
Epochs	250
Batch size	25
Learning rate	0.001

in the embedding space using dimensionality reduction techniques such as PCA and t-SNE.

Training an ANN to Predict Influence From Embedding

In addition to the various metrics evaluated, an ANN was used as an approach of learning a metric. The embedding representation of the 500 most influential training examples were used to predict their influence on the test example.
The embedding together with its label for the 500 most influential training data points were used as input to the ANN, trained as a regression problem to predict the influence of the corresponding training data points on the test point. The ANN architecture and hyperparameters are described in Table 4.3.

5
Results

This chapter begin by presenting results verifying the reliability of influence functions. The first section presents results generated using both the logistic regression model, evaluated on the synthetic dataset, and the all-CNN model, used to classify digits from the MNIST dataset. All other sections solely contain results produced using the all-CNN model.
The subsequent sections compare relationships between influence and the input representation as well as the embedding representation of the images. Furthermore, results of how different metrics perform as approximations of influence are presented. Lastly, the chapter presents some results showing challenges with influence functions, convergence of models and the embedding of images.
Two test images from MNIST test dataset was used to obtain the results for the all-CNN model, one example of the number 6 incorrectly classified as 2 (same example as in [18]) and one example of the number 6 correctly classified as 6, both seen in Figure 5.1. When obtaining embedding results in this chapter, only results from the *logits* layer are displayed.

Figure 5.1: Test image with label 6 incorrectly classified as 2 (left) and test image with label 6 correctly classified as 6 (right) by the model.

5.1 Validation of Influence Functions

The logistic regression model was used to calculate the influence of every example in the synthetic dataset. A plot of the predicted influence as a function of the actual influence can be see in Figure 5.2. The synthetic dataset consists of two distinct clusters and an additional outlier. This fact is also apparent in the plot, since it is possible to distinguish the outlier from the rest of the data in terms of influence. The plot also shows that the predicted influence is highly correlated to the actual influence.

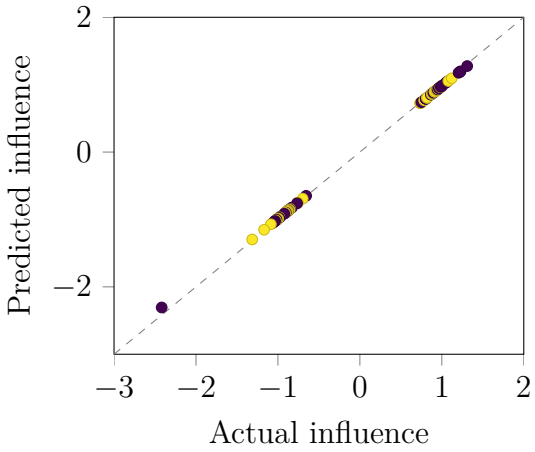


Figure 5.2: Influence of all training examples from the synthetic dataset, for the logistic regression model. The color of the points represent the class they belong to. The actual influence calculated by leave-one-out retraining is compared to the predicted influence calculated using influence functions. There is one apparent outlier in terms of influence.

Scatter plots presenting the same results produced for the logistic regression model, but for the all-CNN model are presented in Figure 5.3. As explained in Section 4.2.3, the actual influence was calculated using leave-one-out retraining for the 500 most influential training examples from the MNIST dataset, approximated by influence functions. The predicted influence was calculated using CG. As can be seen in the figure, the results are significantly less correlated than for the case of using the logistic regression model. However, it is still apparent that the predicted influence approximates the actual influence to some degree.
Images with the largest positive influence, as well as the largest negative influence are presented in Figures 5.4 and 5.5 for the incorrectly classified 6 and the correctly classified 6 respectively. There are notable differences between the two figures, e.g. the set of images with high negative influence for the incorrectly classified 6 are rather hard to interpret and classify for humans. Moreover, the same set of images includes numbers from several different classes, whereas for the correctly classified 6 most of the influential images are either 6s or 0s.

5.2 Naive Relationships With Influence Functions

The results of influence functions presented in the previous section are in this section related to naive metrics in the input and embedding representation of images. Distance between the pixels of the training images and the test image were measured by vectorizing the images and calculating cosine similarity. Figure 5.6 shows the distance between training and test images compared to their actual influence by a scatter plot. No clear relation can be seen for the incorrectly classified 6, but examples with label 6 seem to be grouped together to the upper right for the correctly classified 6.
The same pixel distance as used above was then compared to distances for the

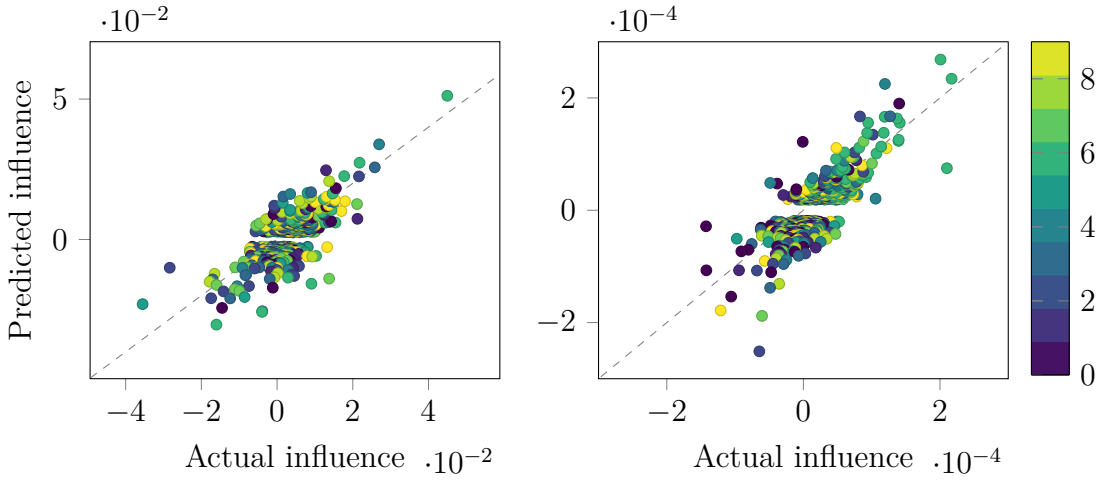


Figure 5.3: Influence of the predicted 500 most influential training examples for the incorrectly classified 6 (left) and correctly classified 6 (right), using the all-CNN model. Predicted influence approximated using the conjugate gradients method as a function of the actual influence calculated using leave-one-out retraining. The color of the points represent their respective label.

Figure 5.4: The images with the largest positive influence (left) and the largest negative influence (right) on the prediction of the incorrectly classified 6.

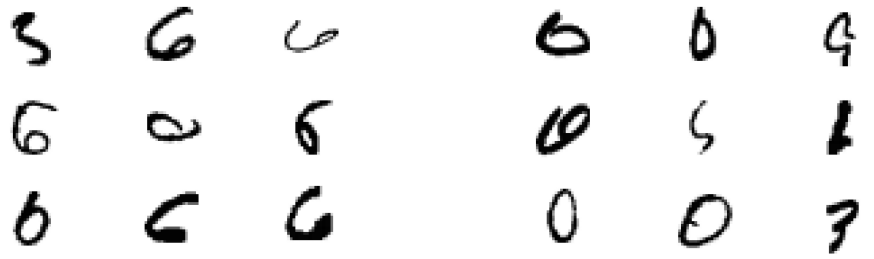


Figure 5.5: The images with the largest positive influence (left) and the largest negative influence (right) on the prediction of the correctly classified 6.

embedding representation of the images using cosine similarity between the test

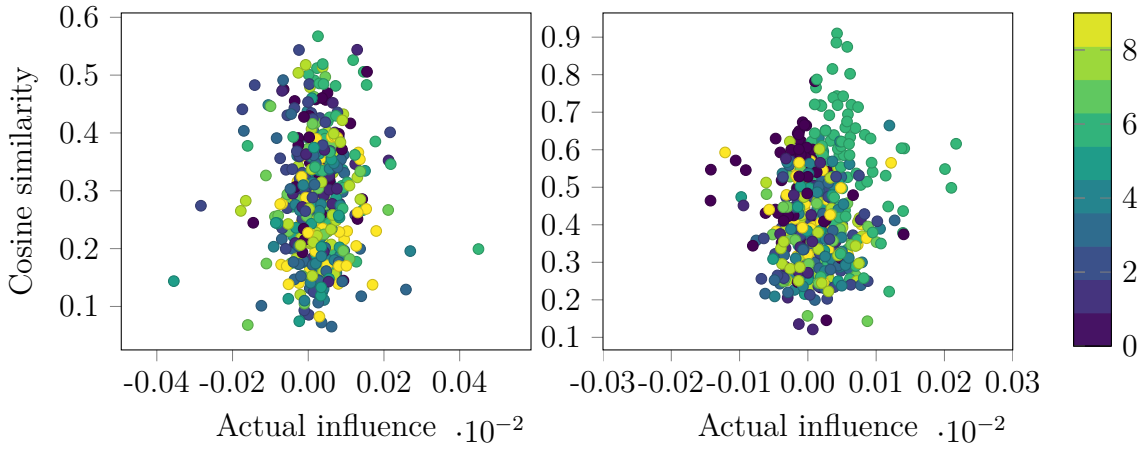


Figure 5.6: Distance between pixels of test images to images in the training dataset measured by cosine similarity, compared to the actual influence for the incorrectly classified 6 (left) and the correctly classified 6 (right). The color of the points represent their respective label.

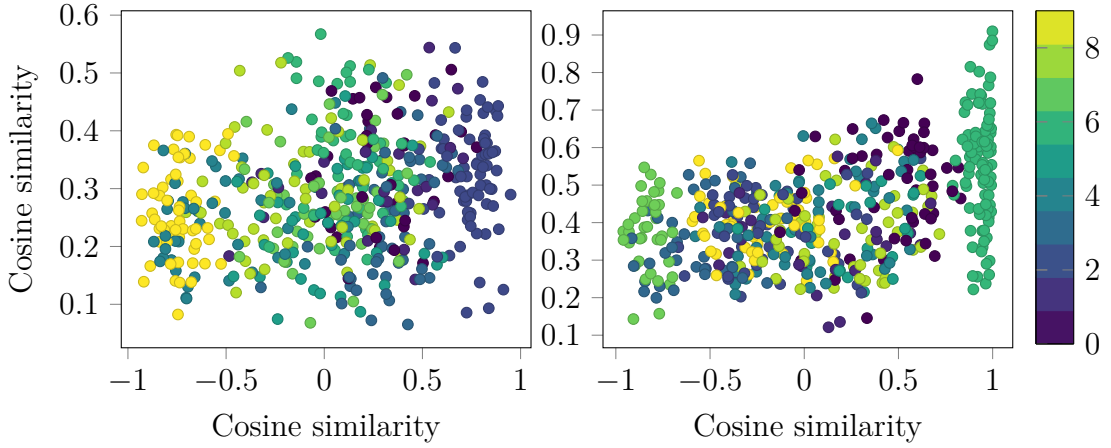


Figure 5.7: Distance between images in training dataset to the test images measured in pixel distance is shown on the y-axis, compared to distance in the embedding space on the x-axis. Distances are measured for the incorrectly classified 6 (left) and for the correctly classified 6 (right). The color of the points represent their respective label.

images and images in the training dataset. The pixel distance is plotted against the distance in the embedding space in Figure 5.7 which show little correlation, but the embedding distance group classes of the same label.

A visualization of the training images representation in the embedding layer in two dimensions, found using t-SNE, is shown in Figure 5.8. The t-SNE algorithm was run for 143 iterations with perplexity 19, learning rate 10 and supervise 0 in the TensorBoard projector, where the values were chosen by experimentation to achieve interpretative results. The data points are colored according to their predicted label.

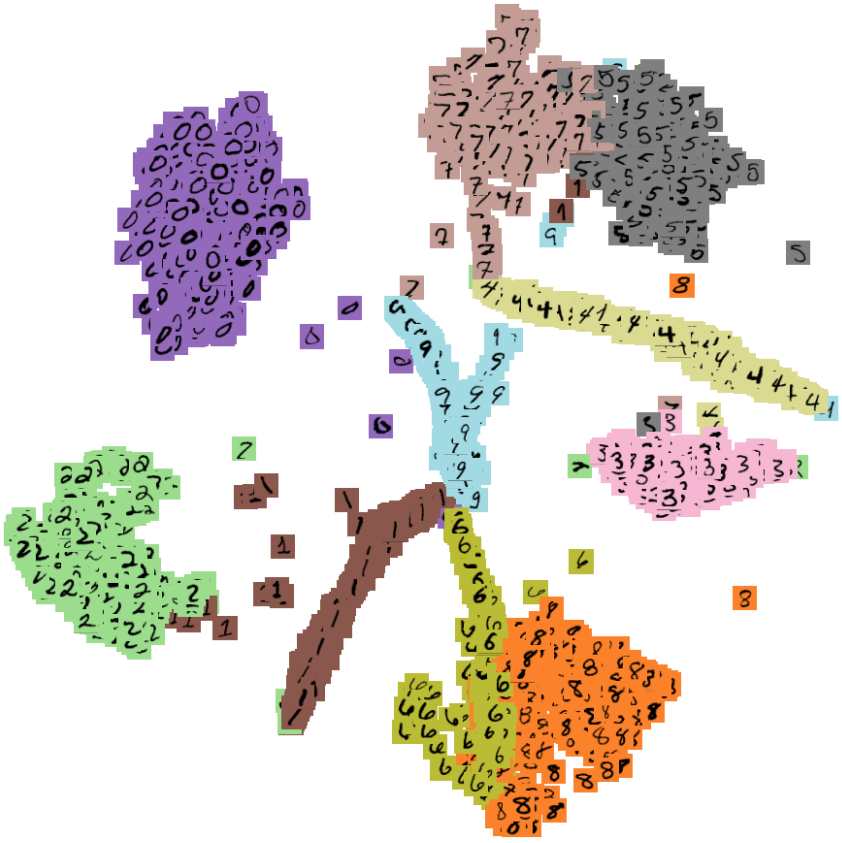


Figure 5.8: Visualization of the embedding representation of training images using t-SNE in the TensorBoard projector, colored by their predicted label by the all-CNN.

The figure shows that examples of the same label are clustered together, with some examples as outliers from their respective classes.

Images with low absolute value of cosine similarity to the test images, as well as images with high absolute value of cosine similarity to the test images are plotted in figures 5.9 and 5.10. The figures show that some images are hard to interpret, similar to the case of the images with high negative influence for the incorrectly classified 6 shown in figure 5.4. Furthermore, many images close to the test images seem to belong to the same class as the predicted class of the test image.

The distance between training and testing images in the embedding space were compared to the actual influence. Distances were measured by cosine similarity, then compared to the actual influence obtained through leave-one-out retraining, shown in Figure 5.11. Examples in the figures are colored according to their actual label. The figure shows little correlation for the incorrectly classified 6, except that classes of the same label seem to be grouped together. For the correctly classified 6 the figure shows that examples of class 6 are clustered close to the test image with a positive influence, this indicate that examples of the same label as the correctly classified example have high influence.

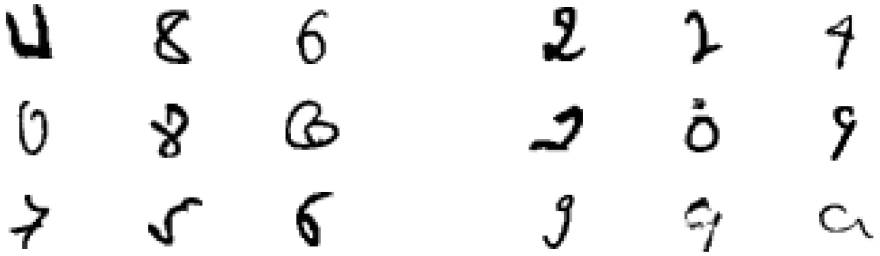


Figure 5.9: The images with low absolute value of the cosine similarity with the incorrectly classified 6 (left) and the largest cosine similarity (right) with the incorrectly classified 6.

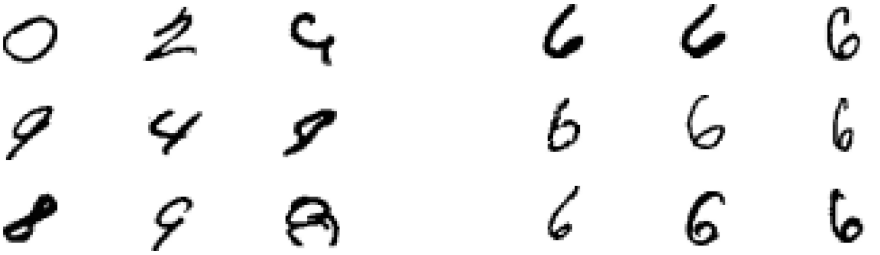


Figure 5.10: The images with low absolute value of the cosine similarity with the correctly classified 6 (left) and the largest cosine similarity (right) with the correctly classified 6.

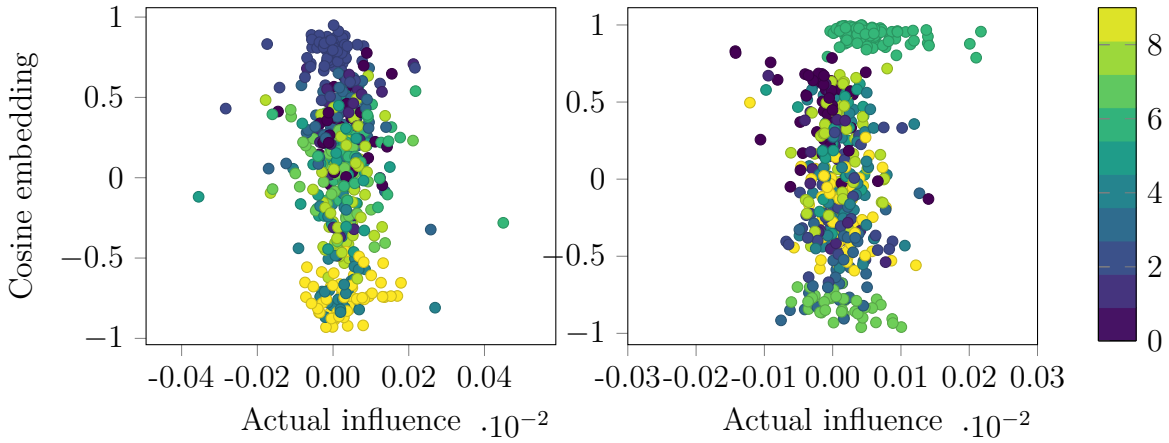


Figure 5.11: The actual influence compared to distance between train images to the test images in the embedding space measured by cosine similarity, for the incorrectly classified 6 (left) and for the correctly classified 6 (right). The color representing the label of the example.

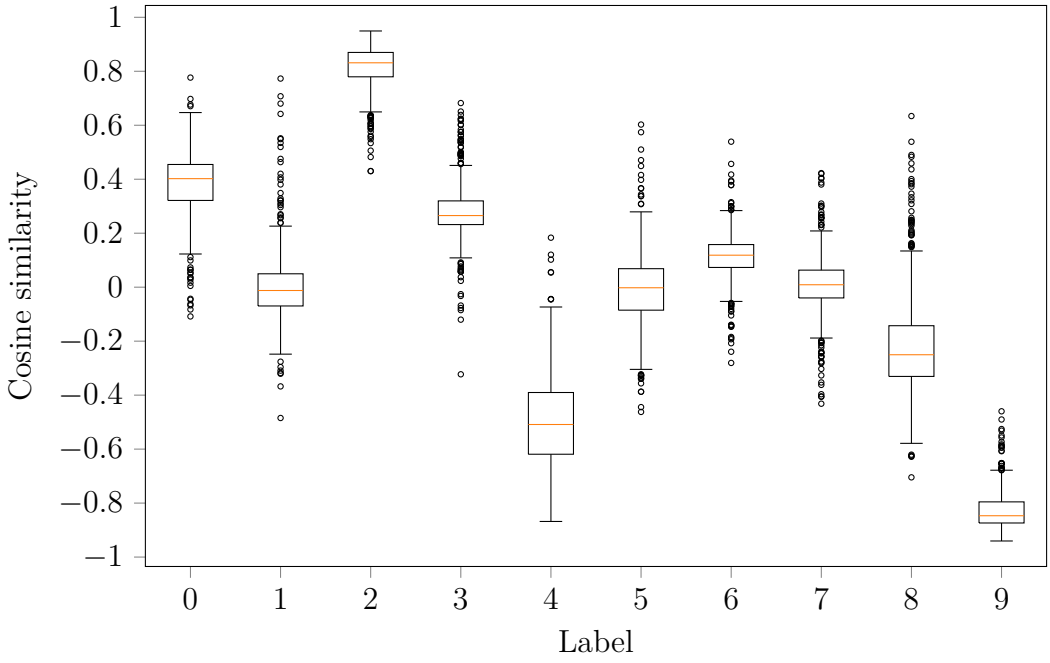


Figure 5.12: Cosine similarity from the incorrectly classified test image embedding to the training images embedding, grouped by class. It appears that some classes can be distinguished from others using cosine similarity.

5.3 Relationships in the Embedding Space

Results from the previous section show that the representation of images in the embedding space seems to group similar examples. This section present further results of relationships in the embedding space and compare them to the influence of training images on a test image.

Distances in the embedding space between training images and the test image was calculated for the incorrectly classified 6 as well as the correctly classified 6. The distribution of distances for each class of training images are presented as boxplots in Figures 5.12 and 5.13 respectively. Both plots indicate that some classes are distinguishable in terms of distance to the test image, whilst the distances to other classes are overlapping.

Furthermore, the influences for each class were investigated. The results are presented in Figures 5.14 and 5.15, where it can be seen that the influence does not differ much between different classes for the incorrectly classified 6. However, the influence of training data with label 6 is higher than the influence of any other class on the correctly classified 6.

For the 500 most influential training examples, the absolute value of the influence for the top 10% furthest away from its respective class centroid (outliers), as measured by Mahalanobis distance, was compared to the influence of the rest of the training images. The results for both of the test images are shown in Figure 5.16. In the box plots, it can be seen that the outliers from the cluster centroids also seem to capture the outliers in terms of influence for both example images. It is also possible to see that the median of the outliers is slightly larger than for the rest of the images,

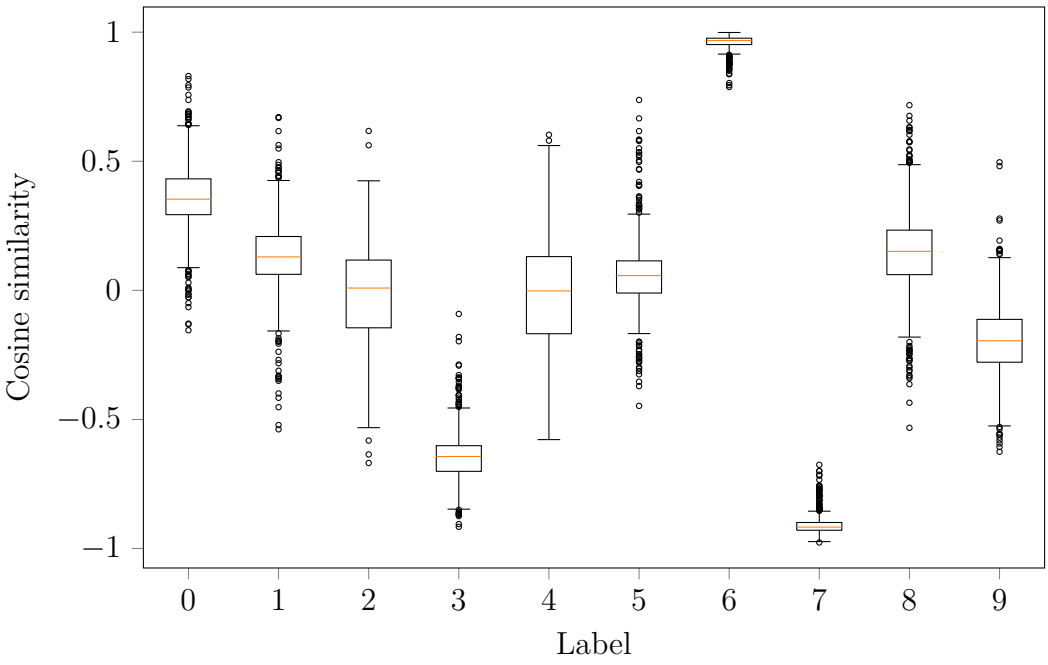


Figure 5.13: Cosine similarity from the correctly classified test image embedding to the training images embedding, grouped by class. It appears that some classes can be distinguished from others using cosine similarity.

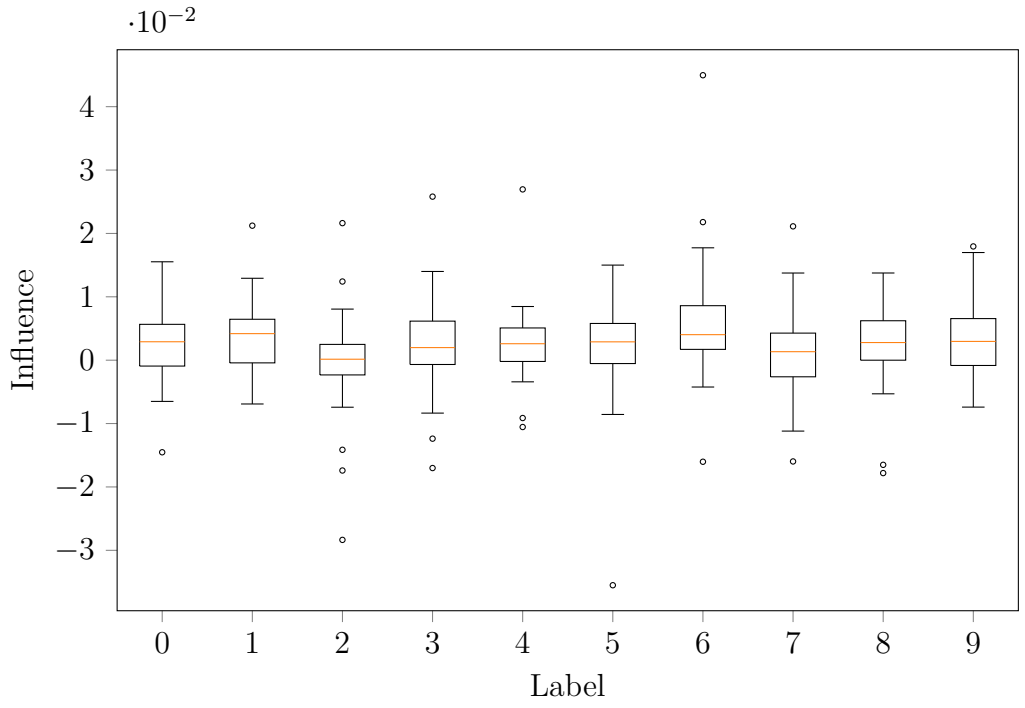


Figure 5.14: Influence on the incorrectly predicted 6, grouped by class. The influence does not differ much between different classes.

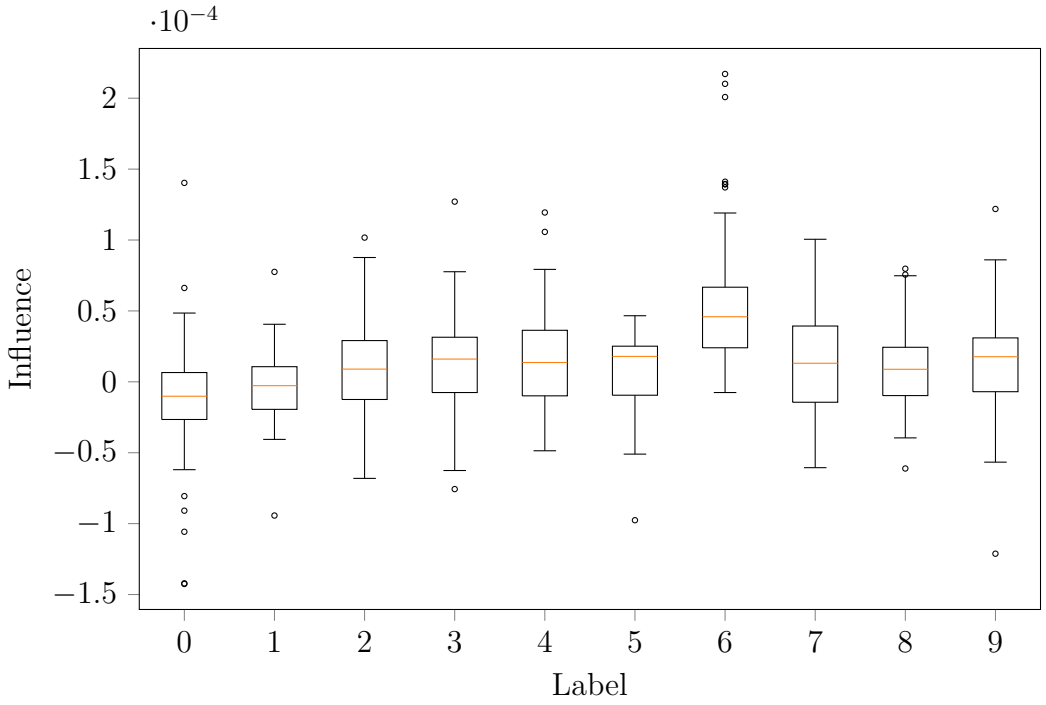


Figure 5.15: Influence on the correctly predicted 6, grouped by class. The class with label 6 has higher influence than any other class. All other classes have similar influence.

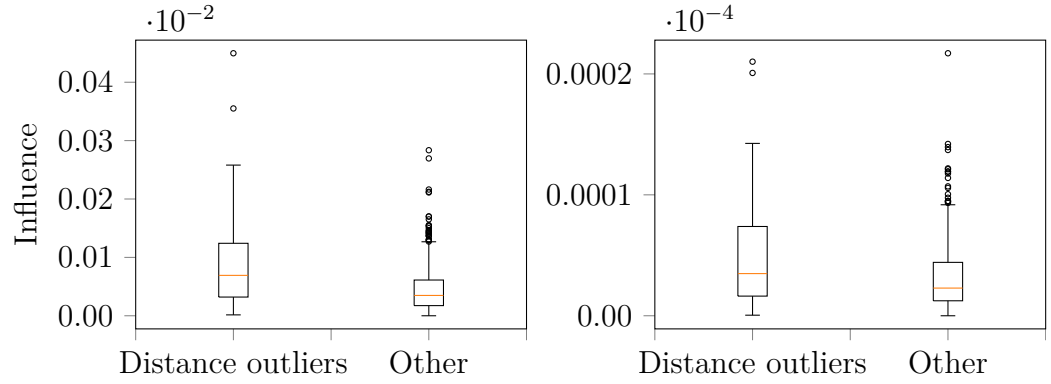


Figure 5.16: Absolute value of the influence of the 10% images in each class that are furthest away from the cluster centroid compared to the influence of the rest of the images. The left plot show results for the incorrectly classified 6 and the right plot show results for the correctly classified 6.

although the boxes still overlap.
Mahalanobis distance to respective class centroid was calculated for all examples in the training dataset and the 500 examples with highest distance were extracted. These 500 examples were compared to the 500 most influential examples calculated through leave-one-out retraining. 261 of the examples with highest Mahalanobis distance was also included in the examples with highest actual influence for the incorrectly classified 6. For the correctly classified 6, 259 of the most influential

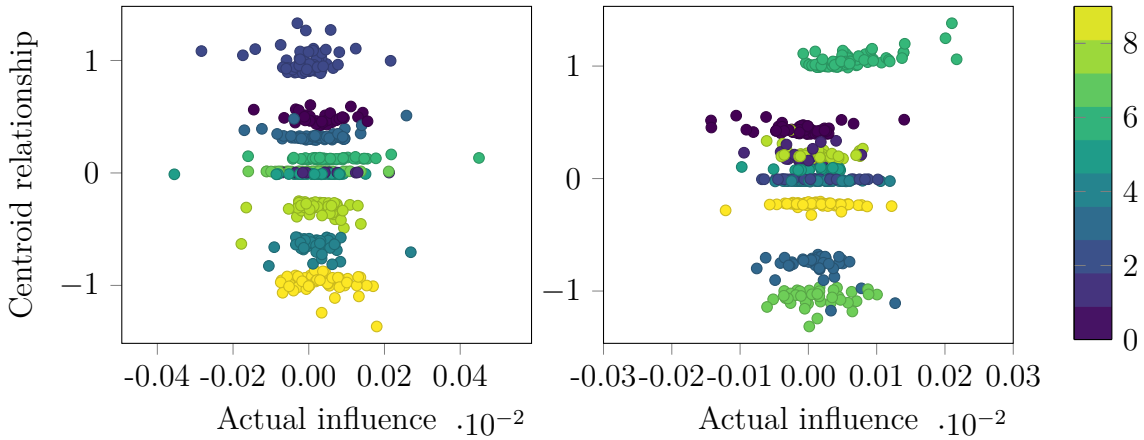


Figure 5.17: The centroid relationship in the embedding space, for the incorrectly classified 6 (left) and for the correctly classified 6 (right), compared to the actual influence, colored by the label of each training point.

examples images could be identified using the Mahalanobis distance, showing that about half of the most influential training examples could be identified by looking solely at the examples with high Mahalanobis distance in the embedding space.
The centroid relationship was calculated for the training images representation in the embedding space. The centroid relationship for each training image was compared to the actual influence of the same train image, visualized as a scatter plot shown in Figure 5.17. Each example is colored by their actual class and the figure shows clear separation of classes in the centroid relationship metric. For the incorrectly classified 6 the figure show little correlation between the centroid relationship and the actual influence, but for the correctly classified 6 images with high centroid relationship also have high actual influence.

5.4 Additional Observations

All metrics presented in this chapter approximate the actual influence for train images on a test image. Pearson correlation coefficients was calculated for all metrics and the actual influence to give an overall score of their respective correlation, presented in Table 5.1. The table shows a high correlation between the actual influence and the predicted influence. Otherwise, there is no significant correlation between any other metric and the actual influence. Notably, the centroid relationship is the embedding metric with highest correlation with the actual influence, both for the incorrectly and correctly classified 6.
Using the embedding representation of the images, an ANN was trained to predict the influence on the incorrectly classified 6 using the 500 most influential training images. The ANN started to overfit the validation set after about 15 iterations of training then stabilized to a validation loss around 0.005 and a train loss around 0.001. A majority of the examples in the train set were correctly predicted by the trained model, but it did not perform well on the test set. The loss and predictions

Table 5.1: Pearson correlation between actual influence and the metrics used used to obtain the results.

	Incorrectly classified 6	**Correctly classified 6**
Predicted influence	0.742	0.805
Mahalanobis distance	0.054	0.090
Cosine similarity input	0.032	0.095
Cosine similarity	-0.110	0.201
Centroid relationship	-0.116	0.251

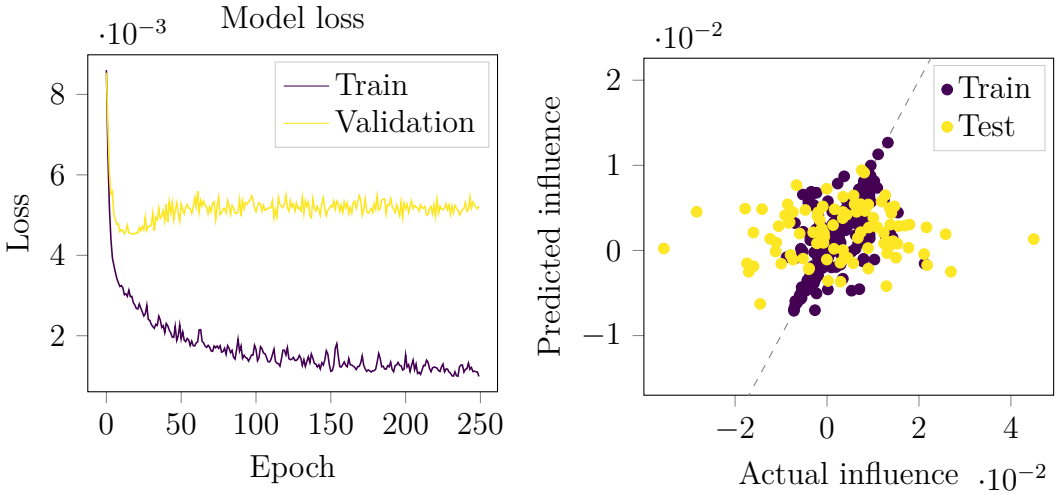


Figure 5.18: The embedding representation of the images in the training dataset was trained with an ANN to predict the influence on the test image. Left: the train and test loss were measured during training of the ANN. Right: the trained ANN was used to predict the influence for the training dataset, compared to the actual difference in loss.

can be seen in Figure 5.18.

5.5 Model Convergence

Some problems related to model convergence in leave-one-out retraining of the all-CNN model appeared. Changes in parameters from the checkpoint of the fully trained model were measured for two cases, by comparing the weights at the checkpoint to the weights obtained after retraining the network with one training image left out. One case with parameter reset and one case continuing retraining from the checkpoint. The parameter change was measured by the difference in weight vectors (the concatenation and vectorization of all weights) of the checkpoint parameters compared to the retrained parameters using Euclidean distance. The difference for parameter reset was 4.491 and the difference for continued retraining from checkpoint was 0.0319. The differences in weights were much larger for parameter reset than retraining from a checkpoint. The individual weight changes can be seen in

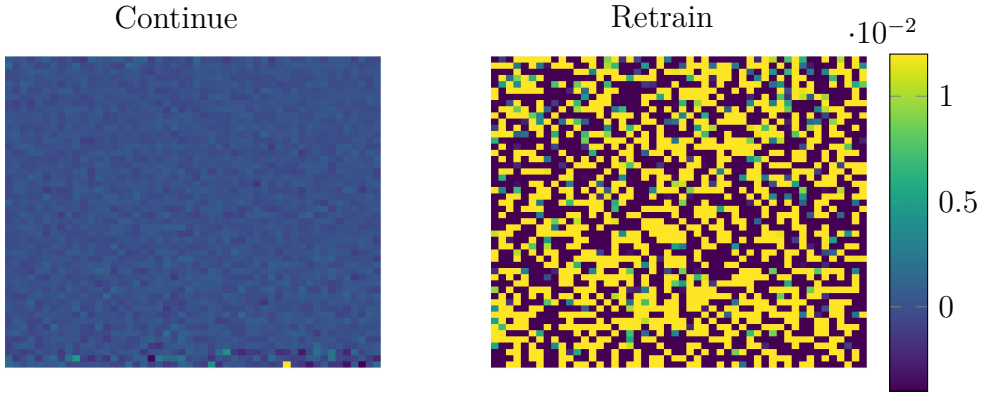


Figure 5.19: The weight changes for continued retraining (left) and complete retraining (right). Each pixel is a weight in the all-CNN network, and its color represent the weight change of the parameters after retraining.

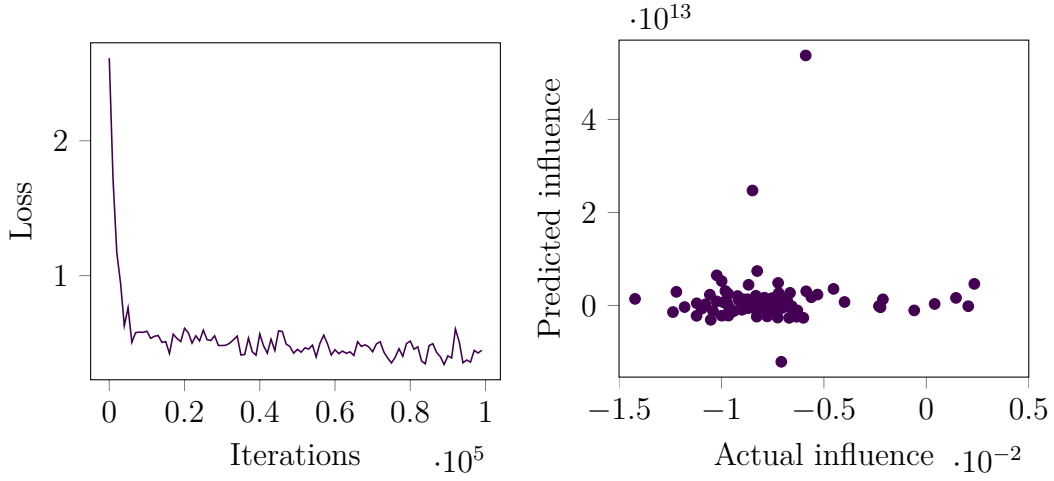


Figure 5.20: An all-CNN model was trained for 100,000 iterations on 100 training examples of the MNIST dataset, which seemed to converge to a loss around 0.5 (left). Leave-one-out retraining with 6,000 additional iterations was compared to the predicted influence using influence functions (right), which did not correlate.

Figure 5.19.
The same all-CNN model was applied to a smaller subset of the MNIST dataset containing 100 training examples. The model was trained for 100,000 iterations, with decay in learning rate after 1,000 and 2,000 iterations. The model had a somewhat noisy loss, that seemed to converge around 0.5, seen to the left in Figure 5.20. The actual influence from leave-one-out retraining with 6,000 additional iterations was compared to the predicted influence using influence functions, which can be seen to the right in Figure 5.20. The figure shows that influence functions cannot correctly approximate the actual influence in this case.

6
Discussion

In this chapter, the findings presented in Chapter 5 are discussed. It begins by reasoning about which test examples and layers were evaluated, then the intuition of the results that were produced to validate the implementation of influence functions. In the subsequent two sections, the performance of various metrics as approximations of influence are discussed. The chapter is concluded by discussing difficulties and problems that were encountered throughout the study.

6.1 Choice of Evaluated Examples and Layers

Two test images were used when obtaining the results presented in Chapter 5; one incorrectly and one correctly classified 6. The incorrectly classified 6 was the same test example as used in [18], but when inspecting this image it is apparent that it is hard to interpret also for humans. It was therefore decided to also investigate a correctly classified example, to see if it yielded the same results. The correctly classified 6 was chosen randomly from all correctly classified examples of class 6 and the results differed substantially between the correctly and incorrectly classified test example.
Only results obtained from the output of the *logits* layer were presented when evaluating the embedding space. This is because of the intuition of the *logits* layer. In order for a model trained on a classification problem to output a correct prediction on the training examples, they needs to be linearly separable in their representation in the *logits* layer in order for a trained model to make a correct prediction. The training examples should therefore by intuition form clusters in the *logits* embedding space, which was confirmed by the results presented in Section 5.2 and 5.3. Embedding representations from the output of other layers in the network may exhibit other properties, which requires further research to exploit.

6.2 Validation of Influence Functions

The results obtained for calculating influence using the synthetic dataset presented in Figure 5.2, clearly show that the implementation of influence functions correctly approximate leave-one-out retraining. They also suggest that there exists a relationship between influence and outliers in relation to the rest of the data. The fact that both classes include examples with positive and negative influence is reasonable, since not all examples are correctly classified.

A correctly predicted test example should be negatively influenced by training examples which belong to the opposite class as the test example, but are positioned on the wrong side of the decision boundary. This is caused by the overall loss growing larger when the training examples are added, and thus forces the model to better fit the class that the training example belongs to. Furthermore, as shown in Figure 5.3 for the all-CNN model and the MNIST dataset, influence functions can give a noisy approximation of leave-one-out retraining. This behaviour is similar to the ones presented in [18].

6.3 Pixel Space and its Relation to Embeddings

The original aim of this project was to find a way to efficiently calculate the influence of training examples on a test example in a predictive model. A naive approach to this problem is to evaluate if there exists any relationships between the training data as represented in the input space and its influence on a test example. However, since a CNN operates on features that are extracted through multiple nonlinear operations, images close in the pixel space are not necessarily influential. This was confirmed by the results in Figure 5.6, where no clear relationship could be found between cosine distance in pixel space and the influence for the incorrectly classified 6. However examples of the same class as the correctly classified 6 were located closer to the test image, and had marginally higher influence than other examples. Furthermore, it was shown in Figure 5.7 that there was no clear relation in terms of distance between input images represented in the pixel space and in the embedding space. The same figure showed that training images with the same label seemed to have similar distances to the test image. This observation led to the assumption that images of the same label are clustered together in the embedding space.

6.4 Embedding Space Distance Metrics

The embedding projector in TensorBoard was used to visualize the embedding space. Figure 5.8 showed distinct clusters in the dimensionality reduction of the embedding space, where images with the same label were clustered together. The figure also showed some examples as outliers from the clusters.
To intuitively reason about the images close to the test image in the embedding space and far away from the test image in the embedding space, these were isolated and compared to influential images. It is notable that some of the images with high negative influence shown in Figure 5.4 and Figure 5.5 are rather hard for humans to interpret. Furthermore, images that are far away from from the test image in the embedding space are assumed to be dissimilar to the test image, and potentially also dissimilar to images from its respective class. Since the images shown in Figure 5.9 and 5.10 are also hard for humans to interpret, it was assumed that there is a relationship between the influence of training images on a test image and their respective representation in the embedding space.
This was not the case for the incorrectly classified 6, as shown in Figure 5.11, which did not show any apparent relation between influence and distance from the train

images to the test image. For the correctly classified 6 however, it could be seen that training examples close to the test example in the embedding space had a relatively high influence as compared to other classes. This may be explained by the test image being correctly classified by the model, if examples of the same class would not be in the training dataset this may cause the model to not correctly classify the test example, resulting in a high difference in loss and a high influence. Both for the correctly and incorrectly classified 6, the distances in the embedding space show that different classes of training images seem to be grouped together.
Studying the box plots in Figures 5.12 and 5.13, it is again apparent that classes are distinguishable in terms of cosine similarity. Using the hypothesis that both similar images or disparate images might be influential on the test image, this could imply that classes close from the test image might differ from others in terms of influence for some distance metric. By looking at Figure 5.15 it can be seen that examples of class 6 have higher influence than other classes for the correctly classified 6. However, this is not case for the incorrectly classified 6 as shown in Figure 5.14. The results presented in Figures 5.16 show the influence of outliers from each class, indicating that there might exist a relation between outliers in each class and their influence, both for the correctly and incorrectly classified 6. This is confirmed by the results showing that training examples which are far away from their class centroid measured by Mahalanobis distance identifies about half of the most influential examples. It can be argued that examples far away from class centroids are not similar to other examples of the same class, in order for the model to correctly classify such examples it need to change its parameters much, resulting in a high influence.
In short, these results indicate that training images similar to a test image can be influential in some cases, but outliers in a class have larger influence than other images for all cases. There might exist some other type of metric that further highlights the features of outliers in relation to the test image, that could better approximate influence.

6.4.1 The Centroid Relationship

In the example of the distance of the training images to their cluster centroids, the metric is independent of the test image. The centroid relationship was created to account for the influence on a specific test image.
It may be reasoned that a training image has the possibility to affect the decision boundary between its own class and adjacent classes in the embedding space. Furthermore it can be assumed that images which differ from the rest of a class distribution would have bigger influence on the decision boundary, meaning examples close to or far away from the decision boundary. Following this argument, a training image would affect the decision boundary of its own class to adjacent classes if left out. A test image close to such decision boundary would therefore be more affected than a test image further away. It was assumed that the distance to a class cluster centroid shares some properties with the decision boundary for that class. There may therefore be some relation between the distance from a train image embedding to the cluster centroid of its class and the distance from a test image embedding

to the same cluster centroid. This relationship is what is measured in the centroid relationship.
As shown in Figure 5.17, examples with high centroid relationship also have a relatively high influence for a correctly classified test example. This does unfortunately not extend to incorrectly classified test examples. As seen in the Pearson correlation shown in Table 5.1, the centroid relationship exhibits the highest correlation with the influence compared to the other metrics investigated both for the correctly and incorrectly classified test example. So there might be some metric or relationship in the embedding space that successfully approximate influence, though not yet found.
It is interesting to consider why the centroid relationship was not a valid approximator. One reason might be that it was assumed that points close or far away from a decision boundary in a high dimensional space would have larger influence on the decision boundary if left out. Since the all-CNN is a highly nonlinear system, it is hard to interpret the way it learn and make assumptions on how individual examples affect the decision boundary in a high dimensional space. Furthermore, the assumption that the distance to the class cluster centroid shares properties with the decision boundary might not be valid.

6.5 Additional Observations

The previous sections discuss the results of using different metrics and measuring their relationships to influence. Another approach of finding a metric is to try to learn it from the data. This method was approached by training the ANN.
Looking at the results of the training of the ANN presented in Figure 5.18, the model manages to approximate influence of the training set, which indicates that its architecture is sufficient for the model to learn features that characterizes the training data. However, using the same model to try to predict the influence of the test images shows opposite results. This could be a consequence of several different factors, e.g. that the model overfits the training data, that there is not enough data, or that it is not possible to learn a metric good enough to approximate influence. The training was stopped before the model started overfitting the validation set, which means that overfitting should not be the cause of the model performing badly. Insufficient training data could still be the problem, but the results show no indication that the model learns features that are present in both the training data and the test data.

6.6 Implementation Challenges

An attempt was made to replicate the results of [18] independent of their implementation. This turned out to be harder than expected. An independent model trained on the MNIST dataset was implemented, converging to a stable train loss and high accuracy. The initial approach to leave-one-out retraining was with parameter reset, which did not give satisfactory results. Even when leave-one-out training without parameter reset was implemented, influence functions did not manage to correctly approximate the actual loss difference.

Comparing the two results showing the difference in parameters when retraining the all-CNN model, presented in Figure 5.19, there is an obvious difference between the two approaches of retraining. This result can be explained by the non-convexity of neural networks. When retraining the model by resetting the weights and re-initializing them, there is a chance that the model will converge to a different optimum than in the first training round. By instead continuing the training, the model is more likely to stay close to the minimum, in terms of weight difference.
This opens an interesting problem: what is sufficient convergence? As seen in Figure 5.20, a model may have a low loss but still not converge to a minimum. Furtermore it is hard to know if a non-convex model has converged and if the found parameters are sufficient in order to apply influence functions. The intention of using a small subset for training was to make the model converge to a local minimum, optimal to the data. This can be compared to a real life example where one may not always have information about the training of the model. The influence functions approach requires the model to converge to a (preferably global) minimum in order to approximate the inverse Hessian. As described in Section 3.2, one could account for non-convexity and non-convergence by adding a damping term to the convex quadratic approximation of the loss. No reasoning regarding the size of the damping term or its relation to the size of the dataset were made. One can therefore infer that the influence functions approach only works under specific circumstances, with a certain set of hyperparameters.

7

Conclusion

This thesis has investigated the influence of training examples on predictions made by deep learning models, by reasoning from the behaviour of a simple logistic regression model how the representation of data could relate to influence. The actual influence of a training example on a test example is defined as the difference in loss on the test example between a model trained with all training examples and the same model trained with a training example left out. It was confirmed that data in the training dataset that intuitively should have high influence also were outliers in terms of influence. Furthermore, it was shown that influence functions can be used to approximate the actual influence, both for a logistic regression model and a convolutional neural network.

The embedding space was defined as the representation of an example from the output of a layer in a deep neural network. Examples close to the test image were shown to have higher influence than other examples if they were of the same class as the test image and the test image had been correctly classified by the model. This was not the case for incorrectly classified test examples.

Furthermore, examples identified as outliers in relation to their class distributions were shown to also be outliers in terms of influence, both for correctly and incorrectly classified test examples. About half of the most influential examples could be identified with this approach.

Since outliers from class distributions for training examples are independent of test examples, an attempt was made to account for their relation. This was done by introducing another metric, referred to as the centroid relationship. A high centroid relationship value for training examples of the same class as a test example were shown to have high influence if the test example had been correctly classified by the model, but not if the test example had been incorrectly classified. However, the centroid relationship showed higher correlation with the actual influence than any other investigated metric. It can be concluded that there exist some relation between examples in the embedding space and influence under certain conditions.

An ANN was not successfully trained to predict influence from the embedding representation of examples for the data available. There furthermore exist problems related to the reproducability of the existing implementation of influence functions, due to a high dependency on model and hyperparameters.

7.1 Future Extensions

As concluded in the previous section, influence functions can be used to find influential training examples on a prediction and the embedding space exhibits some relation with influence under certain conditions. This thesis has encountered various examples where further research could result in a better understanding of the study of influence.
It is interesting to think about what influence could be used for. As introduced in Chapter 1, finding influential training examples could aid understanding if an incorrect prediction is caused by insufficient training data, incorrectly annotated data or a poor model. It might also be possible use influential examples applied to active learning, related to the studies by Ksenia Konyushkova et al. [19].
Another interesting extension would be to investigate influence in other types of models and for different problems, such as semantic segmentation, object detection or natural language processing. These problems generally require models consisting of a large amount of parameters, resulting in long simulation times. This was avoided in this thesis by using the all-CNN architecture consisting of a moderate amount of parameters. It would furthermore be interesting to relate influence as defined in this thesis to other approaches, such as saliency maps [31] which is another approach to evaluate importance in predictions.
Reproducability has become increasingly problematic in the machine learning community, so also in this thesis where an independent replication of influence functions by [18] was not achieved. An independent reproduction of influence functions could aid understanding under which circumstances and with which hyperparameters the approach is applicable. This could also be a step in the direction of creating an end-to-end solution to calculate influence independently of used model.
Finally, as stated above, there exist some relation between the representation of examples in the embedding space and influence. Further exploration of the embedding space and experimentation with different metrics could find more robust relations.

Bibliography

[1] Srikar Appalaraju and Vineet Chaoji. Image similarity using deep cnn and curriculum learning. *arXiv preprint arXiv:1709.08761*, 2017.

[2] Aurélien Bellet, Amaury Habrard, and Marc Sebban. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*, 2013.

[3] Robert R. Bitmead and Brian D.O. Anderson. Asymptotically fast solution of toeplitz and related systems of linear equations. *Linear Algebra and its Applications*, 34:103 – 116, 1980.

[4] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer, 1990.

[5] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pages 737–744, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[6] R. Dennis Cook. Detection of influential observation in linear regression. *Technometrics*, 19(1):15–18, 1977.

[7] R. Dennis Cook and Sanford Weisberg. *Residuals and Influence in Regression*. New York: Chapman and Hall, 1982.

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

[9] K. Fatahalian, J. Sugerman, and P. Hanrahan. Understanding the efficiency of gpu algorithms for matrix-matrix multiplication. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '04, pages 133–137, New York, NY, USA, 2004. ACM.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[11] Joris Guérin, Olivier Gibaru, Stéphane Thiery, and Eric Nyiri. CNN features are also great at unsupervised classification. *CoRR*, abs/1707.01700, 2017.

[12] Chen Huang, Chen Change Loy, and Xiaoou Tang. Local similarity-aware deep feature embedding. *CoRR*, abs/1610.08904, 2016.

[13] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, Sept 2009.

[14] Mayank Kabra, Alice Robie, and Kristin Branson. Understanding classifier errors by examining influential neighbors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3917–3925, 2015.

[15] Douwe Kiela and Léon Bottou. Learning image embeddings using convolutional neural networks for improved multi-modal semantics. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 36–45, 2014.

[16] Minyoung Kim, Stefano Alletto, and Luca Rigazio. Similarity mapping with enhanced siamese network for multi-object tracking. *arXiv preprint arXiv:1609.09156*, 2016.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[18] P. W. Koh and P. Liang. Understanding Black-box Predictions via Influence Functions. *ArXiv e-prints*, March 2017.

[19] Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. Learning active learning from data. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4225–4235. Curran Associates, Inc., 2017.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[21] BG Kumar, Gustavo Carneiro, Ian Reid, et al. Learning local image descriptors with deep siamese and triplet convolutional networks by minimising global loss functions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5385–5394, 2016.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[23] Fei-Fei Li, Justin Johnson, and Serena Yeung. Convolutional neural networks (cnns / convnets). `https://cs231n.github.io/convolutional-networks/`, 2018. Accessed 07/05/2018.

[24] James Martens. Deep learning via hessian-free optimization. *ArXiv e-prints*, 2012.

[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[26] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[27] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[28] Michael Opitz, Georg Waltner, Horst Possegger, and Horst Bischof. Deep metric learning with bier: Boosting independent embeddings robustly. *arXiv preprint arXiv:1801.04815*, 2018.

[29] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[30] Burr Settles. Active learning literature survey. *ArXiv e-prints*, 52, 07 2010.

[31] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[32] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. *CoRR*, abs/1511.06452, 2015.

[33] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[34] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, June 2014.

[35] M van Gerven and S Bohte. Editorial: Artificial neural networks as models of neural information processing. *Artificial Neural Networks as Models of Neural Information Processing*, page 5, 2018.

[36] Mike Wojnowicz, Ben Cruz, Xuan Zhao, Brian Wallace, Matt Wolff, Jay Luan, and Caleb Crable. "influence sketching": Finding influential samples in large-scale regressions. In *Big Data (Big Data), 2016 IEEE International Conference on*, pages 3601–3612. IEEE, 2016.

[37] Jianxin Wu. Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, 2017.

[38] Junyuan Xie, Ross B. Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. *CoRR*, abs/1511.06335, 2015.

[39] Ke-Hai Yuan and Xiaoling Zhong. Outliers, leverage observations, and influential cases in factor analysis: using robust procedures to minimize their effect. *Sociological Methodology*, 38:329–XVI, 2008.

[40] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. *CoRR*, abs/1801.03924, 2018.