Post-hoc uncertainty calibration of neural networks via kernel density estimation

Lukas Kuhn

Bachelor Thesis

Bachelor of Science
(B.Sc.)

in Computer Science

submitted at the
Institute of Computer Science
Johann Wolfgang Goethe-Universität Frankfurt am Main

Submitted on: 27.07.2022
Supervisor: Prof. Dr. Florian Büttner

Supervisor

Prof. Dr. Florian Büttner
Professor for Bioinformatics in Oncology,
Goethe University Frankfurt

Sebastian Gruber
PhD Student, Goethe University Frankfurt

Abstract

In this bachelor thesis we propose a new methodology to calibrate modern neural networks via kernel density estimations. We provide a background on machine learning, neural networks as well as the calibration of neural networks. We also familiarize the reader with kernel density estimation and the theory behind the proposed methodology.

We test this proposed methodology on multiple different modern neural networks and calculate the Expected Calibration Error, the Maximum Calibration Error and the Brier Score, as well as the accuracy and present detailed empirical observations from these experiments.

From the observed data we draw the conclusion that kernel density estimation calibration consistently reduces the calibration error but underperforms Temperature Scaling and discuss advantages and disadvantages as well as future work on the proposed methodology.

Acknowledgments

I would like to express my deepest appreciation to Prof. Dr. Florian Büttner for giving me the opportunity to work with him and his brilliant team at the MLO Lab on such an interesting topic.

I am deeply indebted to Sebastian Gruber who generously provided knowledge, expertise and help and without whom this bachelor thesis would not have been possible.

Lastly, I would be remiss in not mentioning my family, especially my girlfriend, parents and sister. Their belief in me has kept my spirits and motivation high during this process.

Erklärung zur Abschlussarbeit

Gemäß § 35, Abs. 16 der Ordnung für den Bachelorstudiengang Informatik vom 17. Juni 2019:

Hiermit erkläre ich

(Nachname, Vorname)

Die vorliegende Arbeit habe ich selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst. Ich bestätige außerdem, dass die vorliegende Arbeit nicht, auch nicht auszugsweise, für eine andere Prüfung oder Studienleistung verwendet wurde. Zudem versichere ich, dass alle eingereichten schriftlichen gebundenen Versionen meiner vorliegenden Bachelorarbeit mit der digital eingereichten elektronischen Version meiner Bachelorarbeit übereinstimmen.

Frankfurt am Main, den 27.07.2022

Unterschrift der/des Studierenden

Contents

1 Introduction

Machine Learning algorithms are more and more entrusted with solving complex tasks in our day-to-day life. The rising accuracy of neural networks makes them the perfect tool for learning from the now existing large datasets and making predictions under uncertainty. Those predictions are embedded in sophisticated systems ranging from self-driving cars over voice assistants to advanced robotics.

Predictions made by machine learning models are now having wider implications for the general population. For example, a highly accurate neural network in medicine could help save lives by providing valuable predictions and information about the medical data of the patient to the decision maker (for example a doctor who needs to diagnose an illness). Most classification models are working with probabilities instead of giving binary predictions. A neural network that is for example scanning cell scans of patients for cancerous cells is returning a probability for "cancer" and for "not cancer" that together will add up to 1. The class with the highest predicted probability is selected as the predicted class by the neural network. But the probabilities themselves can also provide valuable information to other models or the decision maker.

Take a prediction that is slightly above 50% in the previous example for "not cancer". The selected predicted class is, therefore, "not cancer" but the underlying probabilities give a much better view of the real uncertainty behind this decision. A doctor equipped with such probabilities could run different tests to see if the client is healthy and maybe detect cancerous cells that would otherwise go undetected instead of just relying on the predicted class.

We have now established that the probabilities provided by a neural network are important valuable information. These probabilities should ideally be real measures of uncertainty as we generally understand it. If a model predicts probabilities that are close to the real underlying probabilities it is called well-calibrated. Many methods to calibrate neural networks have been proposed and research in this area is still ongoing [Guo+17].

In this bachelor thesis, we are proposing a new method to calibrate modern neural

networks after they have been trained and validated. We are proposing a method that is using kernel density estimations which are used in Econometrics[1] and Signal Processing.

In Chapter 2 we will explain the necessary background for machine learning, neural networks, calibration and kernel density estimation. In Chapter 3 we will illustrate the methodology behind Kernel Density Estimation Calibration (short KDE Calibration) and explain the theoretical background on which it is based. We will explain how we tested the proposed method on modern data and models, and how they were prepared, trained and optimized. In Chapter 4 we will clarify which current neural networks have been calibrated in validation experiments and in Chapter 5 we will present the results of those experiments. In Chapter 6 we will discuss the results and in Chapter 7 we will give a final conclusion.

1 The application of statistical methods to economic data

2 Background

In this Chapter the necessary background for understanding the proposed methodology is introduced. We start with a short introduction and history of machine learning and neural networks. In Section 2.4 we will then provide a detailed explanation about Confidence Calibration: Why it is a necessary tool to enhance the value of the prediction of Neural Networks, how it can be measured, where we are observing miscalibration in current and historic machine learning models and how current and historic calibration methods work. Following that in Section 2.5 there will be an introduction to Kernel Density Estimation, a crucial cornerstone of the proposed methodology.

2.1 Machine Learning

A modern definition of machine learning is provided by Tom Mitchell in his 1997 book "Machine Learning" where he states that "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [MM97].

Examples of this task T include but are not limited to:

- Classification: Deciding to which group out of n given groups an input belongs to
- Regression: Predicting continuous outcomes by investigating the relationship between given independent and dependent variables
- Clustering: Finding groupings of data and a label associated with the group
- Machine Translation: Translating texts from one language to another language
- Anomaly detection: Identifying unusual patterns that do not conform to the expected behaviour (for example used in credit card fraud detection)

This thesis focuses mainly on the calibration of neural networks used for classification of images.

Current machine learning models learn from thousands to millions of data points in a given dataset and try to maximize the accuracy of a prediction on data which was not part of the original dataset used for learning. The better a machine learning model predicts on this data, the higher its accuracy.

Machine learning models made huge leaps in accuracy in the past decade and are now frequently entrusted with making complex decisions in many different fields where decisions can have highly influential outcomes (for example patient diagnosis in medicine) [Guo+17].

2.1.1 History

In 1943 the American neurophysiologist Warren McCulloch and the American logician Walter Pitts created the first computational model for a neural network based on human learning and cognition. [MP43]

6 years later in 1949 Donald Hebb published "The Organization of Behaviour" [Heb49] detailing the mechanism of neural plasticity later known as Hebbian learning ("Cells that fire together, wire together"), which became a cornerstone in neuroscience as well as machine learning. The first computational network based on Hebbian learning was succesfully implemented at the Massachusetts Institute of Technology in 1954 [FC54].

In 1958 the psychologist Rosenblatt published "The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain" [Ros58]. The perceptron can be considered the prototype of modern artificial neural networks and mimicked the learning process of animals and humans. It has a simple input-output relationship modeled on a McCulloch-Pitts neuron [Ros58].

The Perceptron (as can be seen in Figure 2.1) had 4 input signals labeled X_1 to X_4 which were multiplied by the weights W_{k1} to W_{k4} respectively and summed up together with the bias b_k. After passing through an activation function the result was the output Y_k. The Perceptron would l earn through successively passed inputs while minimizing the difference between desired and actual output but only could learn separating classes linearly [Ros58].

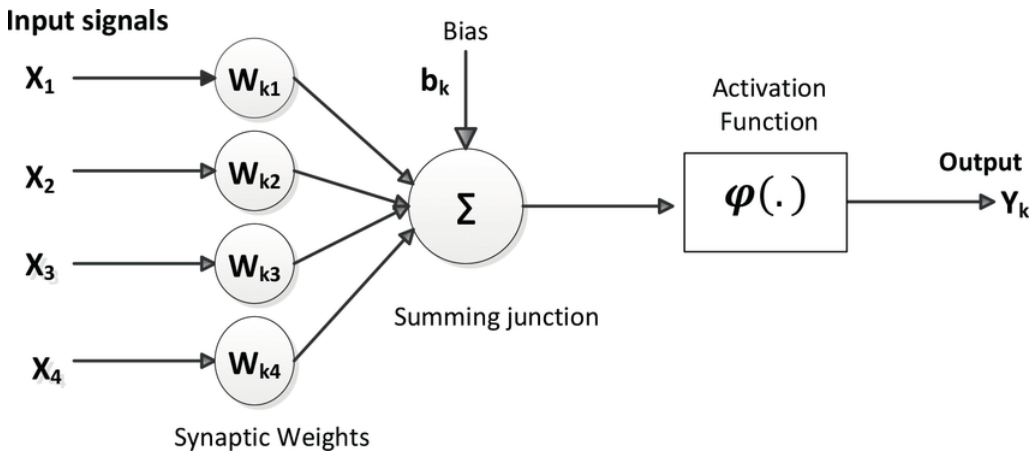

Figure 2.1: Perceptron by Rosenblatt [Ros58] adopted from Guesmi et al. [GFM18]

The term "Machine Learning" was first published in 1959 by Arthur Samuel in "Some Studies in Machine Learning Using the Game of Checkers" in the IBM Journal of Research and Development [Sam67]. The term "Self-teaching computers" was widely used as a synonym in the period before and shortly after this publication.

In 1969 Marvin Minsky argued in his book "Perceptrons" [MS69] that the Rosenblatt perceptron can never be used effectively in multi-layered neural networks because the computation times would be infinitely long. The conclusion was that the research at this time is not going to lead to any real results which was followed by a drastic cut of funding for machine learning research projects for the next decade, leading to what is now called the first "AI winter" [How94].

In 1981, Japan announced its intention to fund the fifth generation effort on neural networks. The Japanese Ministry of International Trade and Industry set aside 850 million dollars to fund projects to create machines that can do a multitude of speech recognition tasks, interpret pictures and reason similar to humans. Following this announcement, funding in the United States also started to increase again because of worries of falling behind the Japanese [Gar20].

In 1985, David B. Parker rediscovered backpropagation [Par85], a method first proposed for the usage in neural networks by Paul Werbos in 1974 in his Ph.D. thesis "Beyond regression: New tools for prediction and analysis in the behavioral sciences" [Wer74], which he published in a report at MIT in 1985 (not knowing

about Werbos discovery 11 years earlier) calling it Learning Logic. Together with gradient descent, generally attributed to the French mathematician, engineer and physicist Augustin-Louis Cauchy, who first suggested the idea in 1847, backpropagation forms the cornerstone of modern neural networks.

In 1986, the term "Deep Learning" was first introduced by Rina Dechter in her work "Learning while searching in constraint-satisfaction problems" [Dec86]. 3 years later, in 1989, Yann LeCunn developed a deep neural network that could (after 3 days of training) recognize handwritten ZIP codes on mail [LeC+89].

Between the late 1990s and early 2000s AI became widely used in larger systems. Many concepts that are still in use today have been discovered in this period including recurrent neural networks, long short-term memory, convolutional neural networks and more. Much progress has since been made in deep learning and machine learning in general by different research groups all over the world. Starting in 2011, funding for AI startups grew from 670 million dollars to 38 billion dollars in 2021 [Tho22].

2.1.2 Categories

Machine Learning can generally be split into three different categories: Supervised, Unsupervised and Reinforcement learning.

Supervised Learning

In Supervised Learning the task for the algorithm is to learn a function that maps an input to an output based on example input-output pairs [Rus10]. The training data consists of labeled examples mapping a given input to a desired output. The function is optimized in regard to a given loss function. A loss function is a function that assigns a loss (or cost) to a given event based on the expected output. The learned function is tested on a test dataset where it is only given the input and the predicted output is compared to the desired output.

This thesis is proposing a method to recalibrate supervised learning algorithms.

Unsupervised Learning

Unsupervised learning is not based on example input-output pairs. It learns a function from untagged input data. Unsupervised learning models are representing

the data internally and try to capture patterns by estimating probability densities [HS99] or learning neural network features.

Reinforcement Learning

Reinforcement learning algorithms are only given a guidance score (called reward) how well they are performing in a given environment. Algorithms can take actions, based on the environment, at a given time interval (discrete or continuous) and receive a reward (negative or positive) which the model will use to optimize the underlying function to receive maximum rewards over a set period of intervals or until a predefined goal is reached[SB18].

2.1.3 Training and Validation

To train and validate a model the labeled input data (assuming it is a supervised machine learning algorithm) has to be split into different subsets. The first set is the training set, which usually contains around 70-80% of the input data. The training data is used to learn from each input x to predict the desired output y.

The second set is the validation set, which is used to select the best model parameters based on how well the model predicts data from the validation input compared to the desired output.

The model with the selected parameters is then tested on the test set, which usually consists of up to 20% of the labeled input data. The accuracy of the model is being calculated by predicting a z_i for each input $x_1, x_2, ..., x_i$ and comparing it to the desired output y_i from the data.

2.1.4 Overfitting and Underfitting

Machine learning models have the goal to generalize well on unseen data. The learned function should not correspond too closely to the data it was trained on, otherwise it would not predict well on unseen data that was not part of the training data. This problem is called "Overfitting". The inverse of this problem is called "Underfitting" and means that the function is too simplistic to accurately predict the data. Both Overfitting and Underfitting can be seen in Figure 2.2.

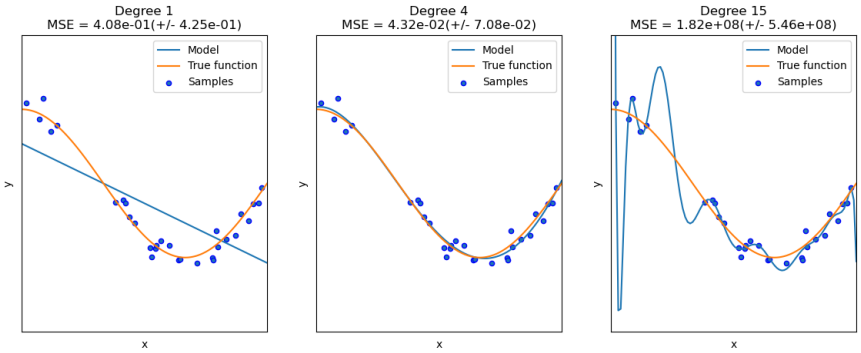

Figure 2.2: Underfitting, good fit and Overfitting adopted from [Doc22]

There are different methods to avoid Over- and Underfitting, like Regularization or Feature Engineering, which we will not discuss in detail.

2.1.5 Neural Networks

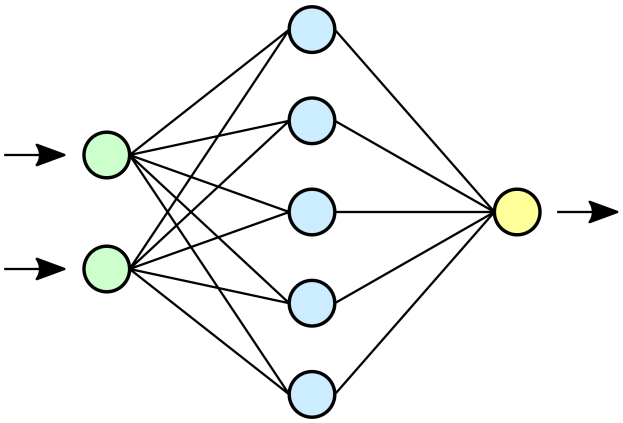

Figure 2.3: Simple neural network as a graph. Source: [Wik22]

Neural Networks are composed of layered artificial neurons. They learn from labeled input data by optimizing weights between the neurons to predict a desired output. Figure 2.3 shows a simple feed-forward neural network consisting of two input neurons (green), a layer with 5 neurons called "hidden layer" (blue) and one output neuron (yellow). These neurons are connected (black lines). Each line has a corresponding "weight" assigned to it which is multiplied with the result of the

previous neurons calculation. Neurons can also apply an activation function (e.g the sigmoid function) to the results. This results in one or multiple predicted output values after the output layer (in Figure 2.3 there is only one output neuron [yellow], so only one output value).

To optimize the weights after each iteration the backpropagation algorithm is used which uses gradient descent to tune the weights between each neuron from back to front to minimize a loss function between current output and desired output.

2.2 Datasets

In the following Section we will describe the datasets that will later be used in the experiments to validate Kernel Density Estimation.

2.2.1 CIFAR-10 & CIFAR-100

The CIFAR-10 dataset [KH+09] consists of 60.000 colored images, each with a size of 32x32 pixels, with 10 different classes. Each class consists of 6.000 images. It is split into 50.000 images for training and 10.000 images for testing. The classes consist of 4 different vehicles and 6 different animals.

CIFAR-100 [KH+09] is built the same as CIFAR-10 but consists of 100 classes instead of 10. Each class consists of 600 images, where 500 images are for training and 100 for testing. Classes include different animals, household items, people, vehicles and more.

2.2.2 ImageNet

The ImageNet dataset [Rus+15] consists of 10.000.000 hand-labeled images with over 10.000 object categories for training and testing. It is used in the yearly ImageNet competition where neural networks compete against each other to correctly classify and detect objects and scenes. The challenge uses fewer classes (one thousand) than are available in the main dataset.

2.2.3 Caltech-UCSD Birds

Caltech-UCSD Birds [Wel+10] consists of 200 different bird species, mostly from North America. The nearly 12.000 images have been hand-labeled with a bounding box, a rough bird segmentation, and a set of attribute labels.[2]

2.2.4 Street View House Numbers (SVHN)

SVHN [Net+11] is a dataset consisting of over 600.000 labeled images. The images contain house numbers obtained from Google Street View. There are 10 classes in total for each digit 1, 2, ...10.

2.3 Neural Networks

In the following Section we will describe the neural networks that produced the logits that will be used in the experiments to validate Kernel Density Estimation Calibration.

2.3.1 DenseNet

DenseNet [HLW16] (published in January 2018) utilises dense connections between layers inside a dense block. All layers are directly connected to all other layers in each block. Each layer obtains inputs from all preceding layers in the same block and passes on its own outputs to all subsequent layers in the same block in typical feed-forward nature.

2.3.2 ResNet

ResNet[He+15] are Residual Networks, that are trying to solve the problem of vanishing gradients. Backpropagation is not working on deeper models because the gradient gets close to zero before it reaches the input layer. Residual Networks have "skip connections" where the gradient can flow backwards from later layers to input layers [Rui19].

2 https://vision.cornell.edu/se3/caltech-ucsd-birds-200/

2.3.3 ResNet SD

ResNet SD [Hua+16] stands for Residual Network with Stochastic Depth. It was first proposed in July 2016 by Huang et al. and tries to combine the usefulness of very deep neural networks with faster training times and better gradients on shallow networks by dropping a subset of layers in training and bypass them with the identity function [Hua+16]. This reduces training time and improves test error.

2.3.4 LeNet

LeNet [Lec+98] is a neural network structure first proposed in 1989 by LeCun et al. It is a simple feed-forward convolutional neural network which was one of the first to perform well on large image datasets.

2.4 Confidence Calibration

As established in the introduction, because neural networks are now used in highly influential decision making, the models must not only be accurate, but also should indicate when they are likely to be incorrect. Model calibration refers to the accuracy with which the scores provided by a machine learning model reflect its predictive uncertainty [Min+21].

We would regard a model as perfectly calibrated if the outputted probability is corresponding to the true correctness likelihood as we generally interpret it. This means that if the model outputs 0.8 as a probability for a class A we expect class A to be the correctly classified label in 80 out of 100 classifications.

We have established that the goal of confidence calibration is predicting probability estimates representative of the true correctness likelihood. These probabilities are important because humans have a natural cognitive intuition for probabilities [CT96] [Guo+17]. Confidence calibration ensures that decisions with the help of machine learning are based on the true probability (or at least as close as possible to the true probability).

Reconsider the example proposed in the introduction: a medical diagnosis by a doctor that uses a neural network image classifier for the detection of cancer in patient images. If the probability is 0.99 that the patient does not have cancer and the model is well-calibrated and has a high accuracy, the doctor can come to the reasonable conclusion that the patient indeed does not have cancer. If the

probability is only 0.60 that the patient does not have cancer the doctor can use other forms of diagnosis to ensure the health of his patient.

If the probability in the given example is not well-calibrated the doctor could not reasonably use the outputted probability and a valuable piece of information would be missing. The uncalibrated outputted probability could make the classified label seem much more certain or uncertain than it is.

Well-calibrated models can also be used to incorporate those models into other probabilistic models [Guo+17].

2.4.1 Measuring Calibration

The quality of calibration can be measured using different approximation tools. In this thesis we will mainly use three well-known measures of calibration:

- ECE (Expected Calibration Error)
- MCE (Maximum Calibration Error)
- Brier Score, a strictly proper scoring function

The *Expected Calibration Error* (ECE) approximates the calibration error by partitioning predictions into M equally space bins and then taking the weighted average of the bins accuracy/confidence difference. It is calculated as follows:

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} |acc(B_m) - Conf(B_m)|$$

The *Maximum Calibration Error* (MCE) calculates the highest gap over all calibration bins. It is calculated as follows:

$$MCE = max_{m=1}^{M}(|acc(B_m) - Conf(B_m)|)|$$

The *Brier Score* [Bri+50] is a strictly proper scoring function, meaning a function that evaluates probabilistic predictions and encourages the reporting of the true probability distribution. It is calculated as follows:

$$BS = \frac{1}{n} \sum_{j=1}^{r} \sum_{i=1}^{n} (f_{ij} - E_{ij})^2$$

Where n is the number of observations and r the number of classes. f_{ij} is the forecasted probability that the event of class j will happen on occasion i. E_{ij}, takes the value 1 or 0 according to whether the event occurred in class j or not [Bri+50].

2.4.2 Observing Miscalibration

Niculescu-Mizil and Caruana [NC05] showed in 2005 that models at this time produced reasonably well-calibrated probability outputs on binary classification tasks.

Guo et al. [Guo+17] assumes that confidence calibration decreased drastically in the last decade and provide a comparison between LeNet from 1998 by LeCun et al. [LeC+98] and a 110-layer ResNet developed by He et al. [He+16] in 2016 on the CIFAR-100 dataset which can be seen in Figure 2.4.

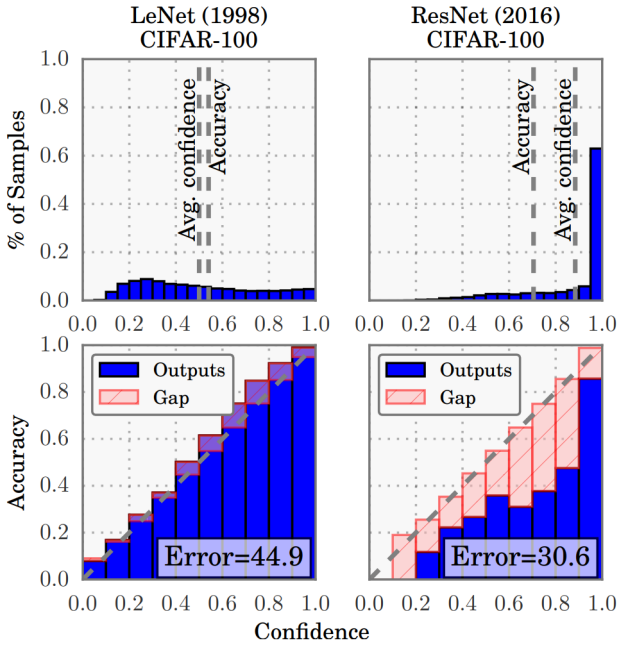

Figure 2.4: Accuracy/Confidence Comparison between LeNet and ResNet obtained from Guo et al. [Guo+17]

The top row in Figure 2.4 shows the distribution of confidence of the predictions from 0.0 (highly uncertain) to 1.0 (completely certain) between LeNet (1998) and ResNet (2016).

"The average confidence of LeNet closely matches its accuracy, while the average confidence of the ResNet [He+16] is substantially higher than its accuracy." Guo et al. [Guo+17]. ResNet (2016) seems to be completely or near completely certain

much more often than LeNet (1998) (based on the increase of predictions falling into the last bin of the prediction histogram)

The differences between the calibration of those two models can also be seen in the bottom row of Figure 2.4, which shows accuracy as a function of confidence. LeNet (1998) "closely approximates the expected accuracy" [Guo+17] whereas ResNet (2016) is much more accurate than LeNet, but the gaps between the confidence and the accuracy are much bigger. Reasons why newer models could be less calibrated than previous models will be discussed in the next Section.

Minderer et al. [Min+21] provides a systematic comparison of recent image classification models and come to a different conclusion for modern neural networks. They come to the conclusion that the trend suggested by papers like Guo et al. [Guo+17], that modern neural networks are less well calibrated than previous generations, seems to be negligible in-distribution and reverses under distribution shifts[3].

Minderer et al. [Min+21] provided a similar histogram as Guo et al. [Guo+17], which can be seen in Figure 2.5.

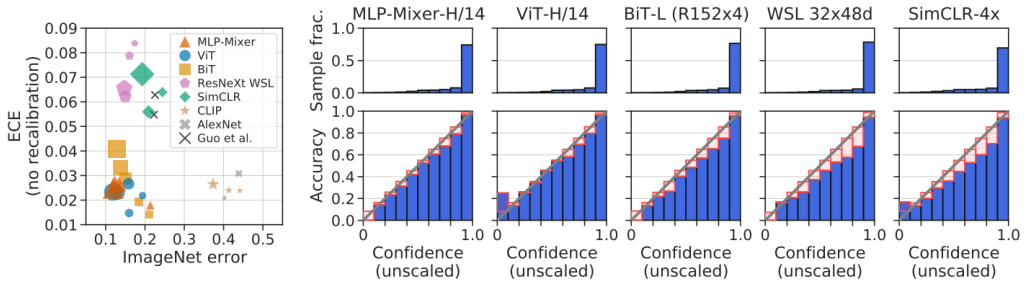

Figure 2.5: Reliability Diagrams by Minderer et al. [Min+21]

They compared and plotted the ECE [4], accuracy and calibration as a function of accuracy for different models:

3 "Data distribution shift refers to the phenomenon in supervised learning when the data a model works with changes over time, which causes this model's predictions to become less accurate as time passes." Huyen [Huy22]

4 The ECE has been estimated using equal-mass binning and 100 bins (for more information about the ECE see Section "Measuring Calibration")

- MLP-Mixer - an architecture developed by Tolstikhin et al. [Tol+21] that does not use convolutions and is instead based on Multi-Layer Perceptrons (MLPs)
- ViT - short for Vision Transformer and developed by Dosovitskiy et al. [Dos+20] is using a Transformer-like architecture on visual tasks
- BiT - a ResNet based architecture developed by Dosovitskiy et al. [Dos+20]
- ResNet-WSL - based on the ResNeXt architecture and developed by Mahajan et al. [Mah+18]. It was trained on billions of hashtags of social media images with weak supervision [Min+21]
- SimCLR - a ResNet based architecture developed by Chen et al. [Che+20] is a ResNet based architecture, pretrained with an unsupervised constrastive loss [Min+21]
- CLIP - developed by Radford et al. [Rad+21] is pretrained on raw text and imagery using a constrastive loss [Min+21]
- AlexNet - was the first ImageNet winning architecture that used convolutions, developed by Krizhevsky et al. [KSH12]

The comparisons seems to indicate that those newer models are all reasonably well-calibrated. Minderer et al. [Min+21] also tried Temperature Scaling, a calibration method that we will discuss in the Section "Calibration Methods", on the past and current models and came to the conclusion, that even after the calibration the most recent models still perform better than the past ones as can be seen in Figure 2.6 provided by Minderer et al. [Min+21].

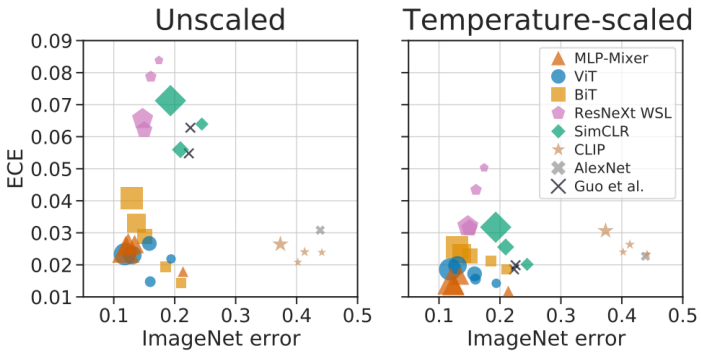

Figure 2.6: Uncalibrated vs Temperature Scaled by Minderer et al. [Min+21]

There seems to be no consensus if modern neural networks are calibrated well.

Nonetheless, both papers come to the conclusion that recalibrating the probabilities can lead to even better calibrated results in modern neural networks.

2.4.3 Reasons for Miscalibration

Guo et al. [Guo+17] suggested that model size could be a reason for the occurring miscalibration. Minderer et al. [Min+21] also found that within most models, larger size tends to correlate positively with the calibration error. Minderer et al. [Min+21] states that there could be a trade-off between model accuracy and calibration here, because larger size usually correlates with lower classification errors. Larger models seem to be overconfident on average.

Guo et al. [Guo+17] also observed that Batch Normalization [IS15], a technique that normalizes each mini-batch in the training of a neural network to achieve better accuracy with fewer training steps, also seems to negatively impact calibration.

Less Weight Decay also was observed as having a negative impact on calibration by Guo et al. [Guo+17]. Models with more Weight Decay in training seem to have better modeled probabilities.

Guo et al. [Guo+17] provided plots for varying depth, width, normalization and weight decay in correlation to the ECE and Error which can be seen in figure 2.7.

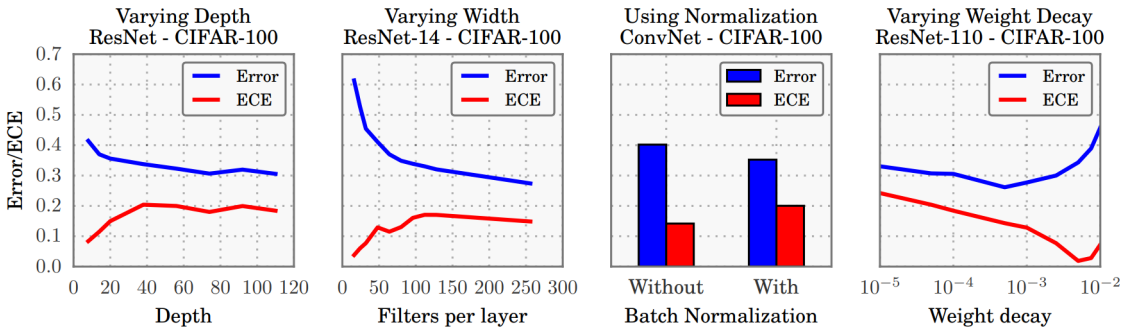

Figure 2.7: Reasons for Miscalibration by Guo et al. [Guo+17]

From this empirical data we can conclude that the deeper and wider a model gets, the worse it calibration gets. Also batch normalization seems to decrease the overall classification error but increase the expected calibration error.

2.4.4 Calibration Methods

To better understand where *Kernel Density Estimation Calibration* is fitting in the current landscape of calibration methods we will discuss some historical and some of the more well-known and used methods today.

Histogram Binning

Histogram Binning [ZE01] is a non-parametric calibration method. It is seldomly used nowadays.

The uncalibrated predictions will get divided into m mutually exclusive bins. Each bin $(B_1, ..., B_m)$ will be assigned a calibrated score θ_m which minimizes the bin-wise squared loss corresponding to the average number of positive-class samples in B_m.

The predicted calibrated probability for a given x in a bin B_i is the assigned score θ_i.

Isotonic Regression

Isotonic regression is also a non-parametric calibration method and according to Guo et al. [Guo+17] the most commonly used non-parametric method. It is based on learning a piecewise constant function to transform the uncalibrated outputs. Isotonic regression tries to minimize the square loss:

$$Loss = \sum_{i=1}^{n} (f(p_i) - y_i)^2$$

It is regarded as a strict generalization of histogram binning in which the bin boundaries and predictions are jointly optimized [Guo+17]

Platt Scaling

Platt scaling is a parametric approach to calibration. It can only be used on binary classification. Platt scaling is based on using the non-probabilistic outputs of a classifier and using them as features in a logistic regression model [Guo+17].

The logistic regression model is trained on the validation logits to output the calibrated probability.

The usage on neural networks was first proposed by Niculescu-Mizil and Caruana [NC05]. The accuracy of the network does not change with Platt scaling.

Temperature Scaling

Temperature scaling is a multiclass extension of Platt scaling proposed by Guo et al. [Guo+17]. It was previously commonly used in knowledge distillation and statistical mechanics.

The method uses a single scalar parameter T called the temperature. T is used to "soften" the softmax function [Guo+17] meaning it will rescale logit scores before applying the softmax function. The softmax output with scaling has a monotonic relationship with the unscaled output, meaning it does not change the maximum of the softmax function and therefore will not change the predicted class.

At $T = 1$ the original probabilities will be returned. A $T > 1$ (in an overconfident model) the recalibrated probabilities will have a lower value than the unscaled probabilities, and will be more evenly distributed. T is optimized on the validation set by optimizing the Negative Log Likelihood.

Comparison

Guo et al. [Guo+17] compared the different methods in calibrating the ResNet-110 (SD) model trained on the CIFAR-100 dataset. Temperature Scaling outperformed Histogram Binning and Isotonic Regression significantly as can be seen in the reliability diagrams of Figure 2.8.

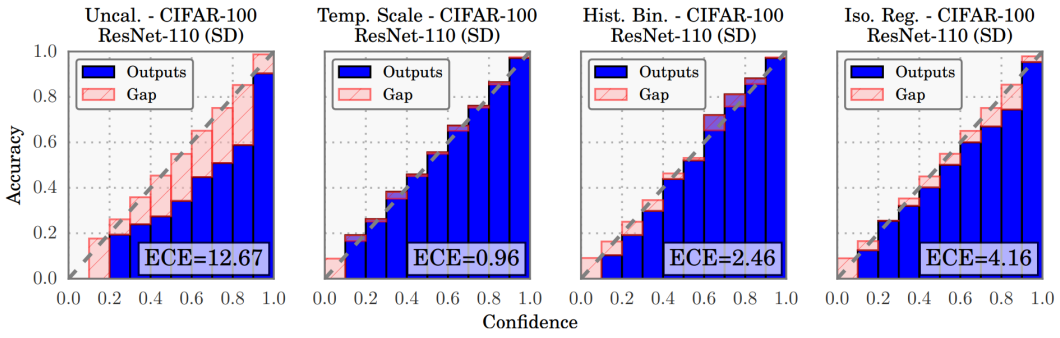

Figure 2.8: Comparison Calibration Methods by Guo et al. [Guo+17]

Guo et al. [Guo+17] comes to the conclusion that Temperature Scaling is outperforming every other tested method on vision tasks and comparably on Natural Language Processing tasks. Temperature scaling since has been widely used and tested for different neural network architectures and is therefore a method which we will compare the proposed KDE Calibration method to.

2.5 Kernel Density Estimation

2.5.1 Overview

Kernel Density Estimation, KDE for short, is a non-parametric way to estimate a probability density function of a random variable [Ros56]. The basic idea is to estimate the probability density function at a point x using neighboring observations [ZD12].

Unlike in a histogram the estimates are not build up according to bin edges but according to the rules of each kernel (Kernels will be explained in Section 2.5.2)

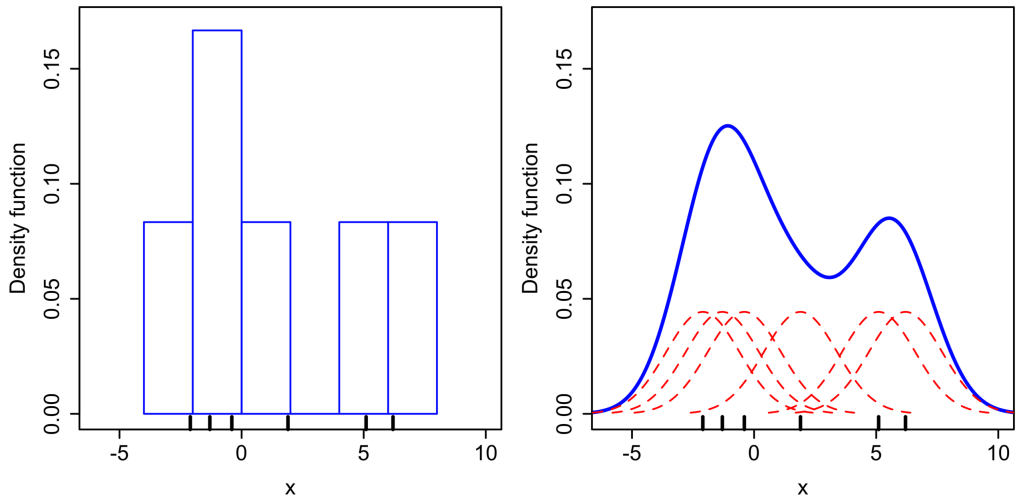

Figure 2.9: Comparison of 1D Histogram and KDE Wikipedia [Wik22]

The estimation is build up by weighting the distance of all the data points around point x based on the Kernel function as can be seen on the right side of Figure 2.9.

We examine the Naive Kernel as an example: The naive kernel function uses

each point x as the center of a bin with width $2h$ where h is the bandwidth (will be discussed in Section 2.5.3). The Kernel function (weighting of the different distances) therefore is:

$$K(x) = \begin{cases} \frac{1}{2}, & \text{if } |x| < 1 \\ 0, & \text{otherwise} \end{cases}$$

This kernel function is then used for the kernel estimate:

$$f(x) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - X_i}{h})$$

The naive kernel is the simplest kernel because it uses a simple bin with width 2h centered at x [ZD12].

Kernel Density Estimation is currently used in Econometrics [ZD12] and Signal Processing.

2.5.2 Kernels

The Naive Kernel, is only one of many kernels used in Kernel Density Estimation. Different kernels provide different weightings of the distances from point x and will therefore result in different probability density functions. The goal of the Kernel Density Estimation is to represent the underlying probability density as closely as possible.

Figure 2.10 shows a list of the most commonly used kernels in Kernel Density Estimation together with a visual representation of the resulting weights.

Kernel weight	K(x)
Uniform	$\frac{1}{2}\mathbb{1}(\|x\| < 1)$
Gaussian	$\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^2}$
Epanechnikov	$\frac{3}{4}(1-x^2)\mathbb{1}(\|x\| \leq 1)$
Biweight	$\frac{15}{16}(1-x^2)^2\mathbb{1}(\|x\| \leq 1)$
Triweight	$\frac{35}{32}(1-x^2)^3\mathbb{1}(\|x\| \leq 1)$

Tabela 1: Most common Kernel weights

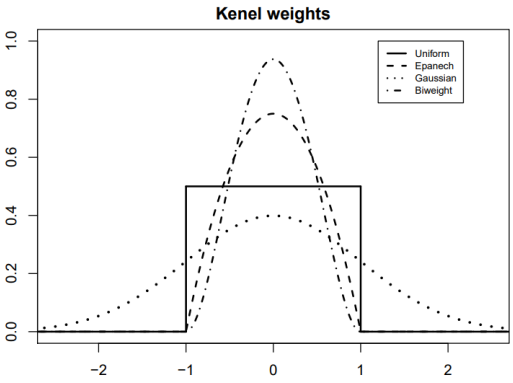

Figura 3: Kernel weight functions

Figure 2.10: KDE Kernel Weights Comparison by Zambom and Dias [ZD12]

As can be seen in Figure 2.11, different kernels (in this example the Naive Kernel and the Epanech Kernel) will result in different probability density functions.

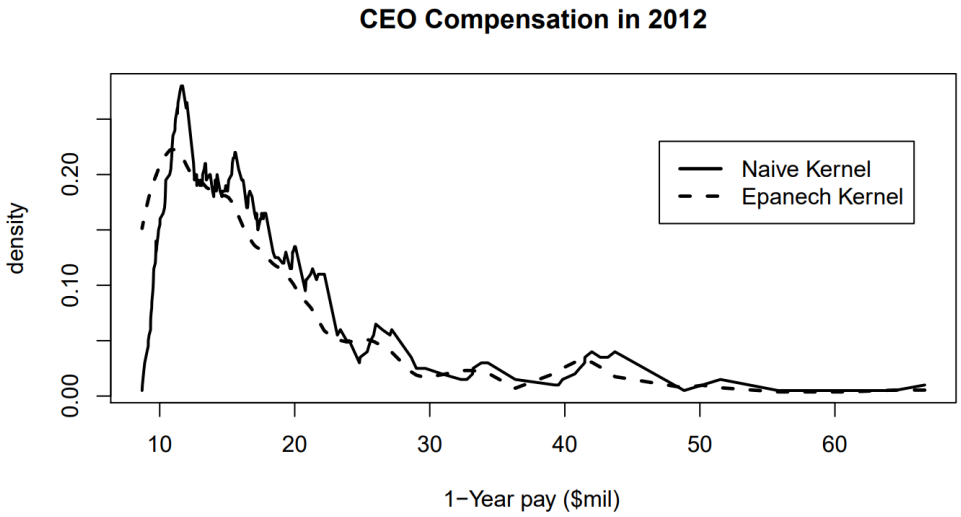

Figure 2.11: Comparison of two Kernels on CEO Compensation Data by Zambom and Dias [ZD12]

For the proposed Kernel Density Estimation Calibration method we use different kernel functions which we cross-validate to find the best fit to the data. All kernels usually satisfy:

$$\int_{-\infty}^{\infty} K(x)\, dx = 1$$

It is also assumed that Kernel K is a unimodal probability function that is symmetric about 0. This guarantees that $K(x)$ is a valid probability density.

2.5.3 Bandwidth

The concept of weighting the distances of our observations from a specific point, x, can be expressed mathematically as follows:

$$f(x) = \sum_{observations} K\left(\frac{x - observation}{bandwidth}\right)$$

The bandwidth $h > 0$ is a smoothing parameter. A kernel with the bandwidth h is called a scaled kernel which is defined as $K_h(x) = \frac{1}{h}K(\frac{x}{h})$. The parameter h can be optimized to achieve a better density estimate as can be seen in Figure 2.12.

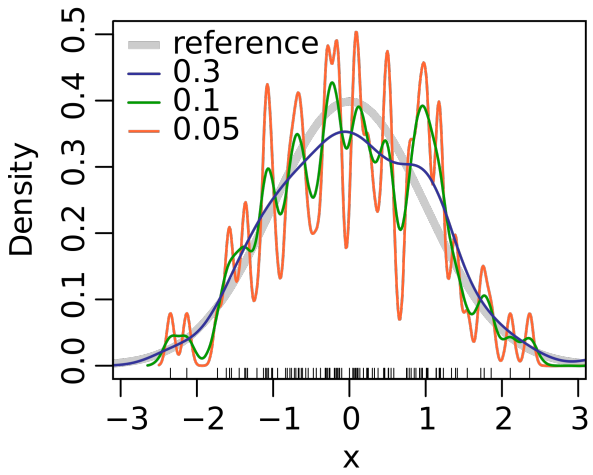

Figure 2.12: Kernel Density Estimation Bandwidth Comparison by Wikipedia [Wik22]

Figure 2.12 displays different bandwidth parameters using a normal kernel trying to estimate the probability density of the reference.

The higher the bandwidth parameter h, the smoother the probability estimation. This can lead to undersmoothing as can be seen with h being 0.05 (orange line) or oversmoothing with a parameter $h > 0.3$ (blue line).

We will estimate and cross-validate different Bandwidth parameters to best fit to the real underlying probability density (see Chapter 3.4.1).

3 Methodology

First we will give an overview about the theory behind KDE Calibration. Then we will explain the coding framework and its three main stages: Data Preparation, Training and Optimization. We will discuss each stage in depth in the next sections.

3.1 Theory of KDE Calibration

KDE Calibration is using Kernel Density Estimation as an estimator in a Bayes classifier. Bayes theorem can be rewritten as:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

Instead of using $P(X|y)$ we are using a kernel density estimation of X given y. With an accurate kernel density estimation we expect to receive an accurate $P(y|X)$.

If we are for example searching for the calibrated confidence $P(3|X)$ of class 3 in a multiclass prediction problem, where X are the logits obtained from validating the neural network, and $P(y = i)$ is obtained by diving the count of all datapoints by the number of datapoints in class i, we have the following equation:

$$\mathbb{P}(y = 3|X) = \frac{\mathbb{P}(X|y = 3)\mathbb{P}(y = 3)}{\sum_i \mathbb{P}(X|y = i)\mathbb{P}(y = i)}$$

In the numerator we are multiplying the probability that X occurs given class y is 3, times the probability that class y is 3. In the denominator we are summing up all probabilities that X occurs given the class y is $1, 2, ..., i$ multiplied by the probability that class i occurs in the dataset. Since we can't use the exact probability

of $P(X|y = 3)$ we will use a fitted multivariate kernel density estimation given that class y is 3, which we note as KDE_3. We now have the following equation:

$$\mathbb{P}(y = 3|X) = \frac{KDE_3(X)\mathbb{P}(y = 3)}{\sum_i KDE_i(X)\mathbb{P}(y = i)}$$

This method depends on an accurate estimation of $P(X|y)$ which we try to achieve with a kernel density estimation. The Kernel Density Estimator gets better with more data. In the Sections "Experiments" and "Results" we will verify the accuracy of this method of calibration in a practical setting.

3.2 Framework Overview

In the Data Preparation stage we unpickle the logits that got saved in the process of validation of each model, we split them into training and test sets for later usage.

In the Optimization stage we use the *Optuna* [Aki+19] hyperparameter optimization framework to fit multiple KDE Calibrations with different bandwidth scaling parameters $\sigma_1, \sigma_2, ..., \sigma_n$ and cross-validate them on the training set to find the optimal bandwidth to minimize a given score (ECE, MCE, Brier, see section 2.4.1).

After we have found the optimal bandwidth scaling parameter σ we train the final multivariate kernel density estimations for each class in the Training stage.

In the evaluation stage we calculate different metrics to evaluate the performance of the method and compare these results to the Temperature Scaling method [Guo+17] (see section 2.4.4 for a detailed explanation).

The framework is written in the Python programming language.

3.3 Data Preparation

Most logits have been made available in a pickled file format. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream. We therefore have to unpickle the logits to use them for the kernel density estimation. For this we use a method written by Markus Kängsepp for his neural network

calibration project available on Github[5].

After unpickling the logits we split them into a train and test set, where 80 percent is reserved for training and 20 percent for testing.

We fit the multivariate Kernel Density Estimations on the raw logits so we don't need any more preparation steps before training.

3.4 Optimization

3.4.1 Bandwidth Scaling

We use Statsmodels KDEMultivariate function [Sta22] to do all kernel density estimations. The function is providing us with an estimate of a fitting bandwidth for each KDE.

We take this estimate bw and scale it between $0.75bw$ and $1.5bw$ as we have seen the best results between this range as part of our hyperparameter tuning. To find the best scaling parameter σ we run between 50 to 100 trials in the hyperparameter optimization framework Optuna [Aki+19]. After each run we validate either the ECE, MCE or Brier (depending on our minimization goal) and after all runs are finished we return the trial run (including the scaling parameter σ) with the best metric.

3.4.2 Optuna

Optuna, developed by Akiba et al. [Aki+19], is a hyperparameter optimization framework, which provides an efficient way to construct a parameter search space and implements efficient algorithms for searching and pruning of parameters [Aki+19]. We use a Tree-structured Parzen Estimator, which is the standard in Optuna, to estimate the bandwidth scaling parameter σ. The Tree-structured Parzen Estimator uses Gaussian Mixture Models to estimate and optimize the parameter σ in each run [Ber+11].

5 https://github.com/markus93/NN_calibration/blob/master/scripts/utility/unpickle_probs.py

3.5 Training

We first calculate the probabilities $p_1, p_2, ..., p_k$ (for k classes) that a given class p_y occurs in the dataset and save them for later usage (see Chapter 3.1 for a detailed explanation about the theory behind this method).

Then we have to calculate a multivariate Kernel Density Estimation for each class in a given dataset. We are using Statsmodels KDEMultivariate function [Sta22] to calculate those. We are giving the function the unpickled logits as a 2D numpy array and set the variable type to continuous for each class.

After each multivariate KDE is fitted we will scale the bandwidth with the scaling parameter σ found during the Optimization stage.

3.6 Testing

For each dataset (set of logits) we are using the fitted Kernel Density Estimation Calibration to predict probabilities for the reserved test data points (see section "Data Preparation").

4 Experiments

As explained in the Methodology section we evaluated Kernel Density Estimation on logits from different common neural networks. In this chapter we will explain the experiment setup and list all the datasets and neural networks that have been used to evaluate Kernel Density Estimation Calibration.

We are predicting the probabilities for each datapoint with the previously described method in the test set and are evaluating the resulting probabilities by calculating the Expected Calibration Error (ECE), the Maximum Calibration Error (MCE), the Brier Score (BS) and the Accuracy. These results are then getting plotted on a reliability diagram and are shown in Chapter 5.

We are evaluating all the previously described obtained results by calculating the same metrics (ECE, MCE, Brier Error) for the Temperature Scaling method and comparing the results to the proposed KDE Calibration.

4.1 Overview

The neural networks that we evaluated Kernel Density Estimation on have been trained on the following datasets:

- CIFAR-10 and CIFAR-100 [KH+09]
- ImageNet [Rus+15]
- Caltech-UCSD Birds [Wel+10]
- Street View House Numbers (SVHN) [Net+11]

The following is a complete list of all the neural networks that have been evaluated with Kernel Density Estimation Calibration:

- DenseNet on CIFAR-10 & CIFAR-100 [HLW16]
- ResNet on CIFAR-10 & CIFAR-100 [He+15]

- ResNet on Birds [He+15]
- ResNet on ImageNet [He+15]
- ResNet SD on CIFAR-10 & CIFAR-100 [Hua+16]
- ResNet SD on CIFAR-10 & CIFAR-100 [Hua+16]
- Wide ResNet on CIFAR-10 & CIFAR-100 [ZK16]
- LeNet on CIFAR-10 & CIFAR-100 [Lec+98]

4.2 Parameters

We optimized the bandwidth parameter σ in 50 trials in the Optuna framework [Aki+19]. We evaluated the results (ECE, MCE and Brier Score) against a Temperature Scaling implementation provided by Markus Kängsepp on his Github repository[6].

6 https://github.com/markus93/NN_calibration/blob/master/scripts/calibration/cal_methods.py

5 Results

In this chapter we will present the results obtained from Kernel Density Estimation Calibration.

We will present the obtained data from ECE, MCE and Brier Score first in tabular and visual form and then will present reliability diagrams and confidence histograms for some obtained results.

5.1 Expected Calibration Error

First we present the results with respect to the *Expected Calibration Error* (ECE). The best result is displayed in bold.

Results with ECE				
Neural Network	Dataset	Uncalibrated	KDE Calibration	Temperature Scaling
Wide ResNet	CIFAR-10	0,0450	0,0080	**0,0078**
LeNet	CIFAR-10	0,0518	0,0272	**0,0166**
DenseNet	CIFAR-10	0,0550	0,0106	**0,0094**
ResNet	CIFAR-10	0,0475	0,0162	**0,0113**
ResNet SD	CIFAR-10	0,0411	0,0073	**0,0055**
ResNet SD	SVHN	0,0086	0,0068	**0,0061**

Table 5.1: Results according to the ECE

We noticed that Kernel Density Estimation Calibration underperforms against Temperature Scaling by a small margin of -0,003 and outperforms the uncalibrated network by a margin of 0,0287.

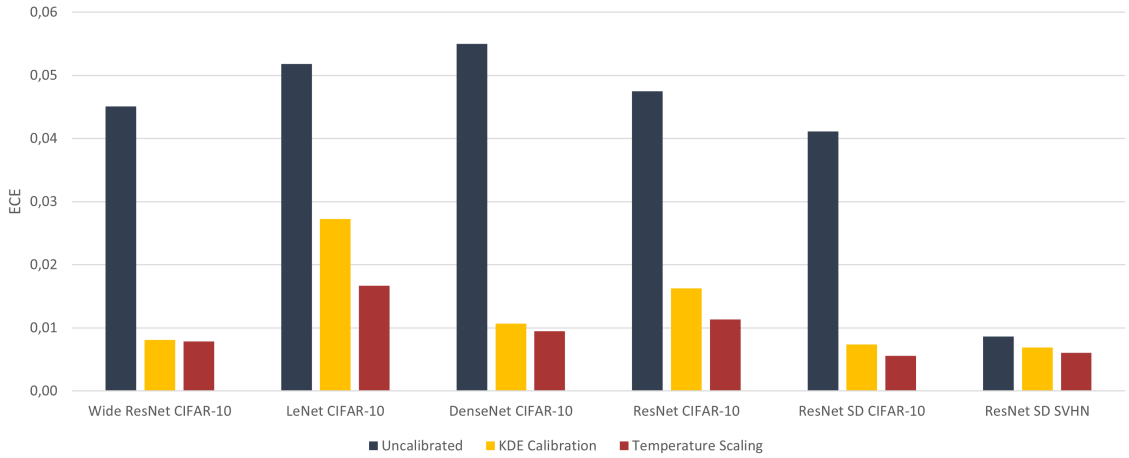

Figure 5.1: Visual Comparison of Uncalibrated, KDE Calibration and Temperature Scaling for the ECE

5.2 Maximum Calibration Error

Presented next are the results based on the *Maximum Calibration Error* (MCE).

Results with MCE				
Neural Network	Dataset	Uncalibrated	KDE Calibration	Temperature Scaling
Wide ResNet	CIFAR-10	0,3721	0,1323	**0,0705**
LeNet	CIFAR-10	0,1128	**0,0621**	0,0915
DenseNet	CIFAR-10	0,3339	0,1481	**0,0992**
ResNet	CIFAR-10	0,2957	0,2586	**0,2364**
ResNet SD	CIFAR-10	0,3248	0,2461	**0,0782**
ResNet SD	SVHN	0,2503	**0,1298**	0,1824

Table 5.2: Results according to the MCE

We examine that Kernel Density Estimation Calibration sometimes outperforms Temperature Scaling in regards to the Maximum Calibration Error.

It outperforms the uncalibrated network by a margin of 0,1187 on average and underperforms Temperature Scaling by -0.0364.

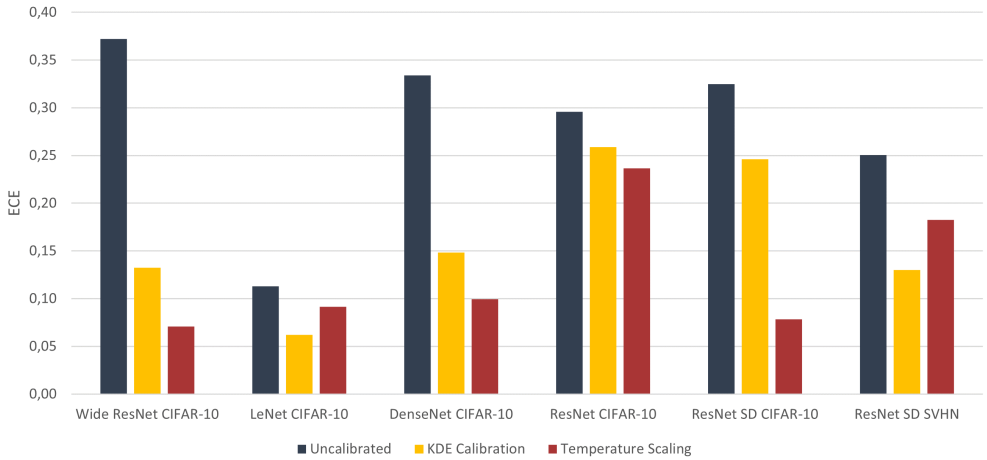

Figure 5.2: Visual Comparison of Uncalibrated, KDE Calibration and Temperature Scaling for the MCE

5.3 Brier Score

Presented next are the results based on the *Brier Score*.

Results with Brier Score				
Neural Network	Dataset	Uncalibrated	KDE Calibration	Temperature Scaling
Wide ResNet	CIFAR-10	0,1046	0,1044	**0,0923**
LeNet	CIFAR-10	0,3787	0,4070	**0,3747**
DenseNet	CIFAR-10	0,1273	0,1397	**0,1099**
ResNet	CIFAR-10	0,1101	0,1085	**0,0978**
ResNet SD	CIFAR-10	0,0981	0,1121	**0,0873**
ResNet SD	SVHN	0,0297	0,03394	**0,02914**

Table 5.3: Results according to the Brier Score

We can see (plotted on Figure 5.3) that Kernel Density Estimation Calibration not once performed better than Temperature Scaling according to the Brier Score and sometimes even performs worse than the uncalibrated network.

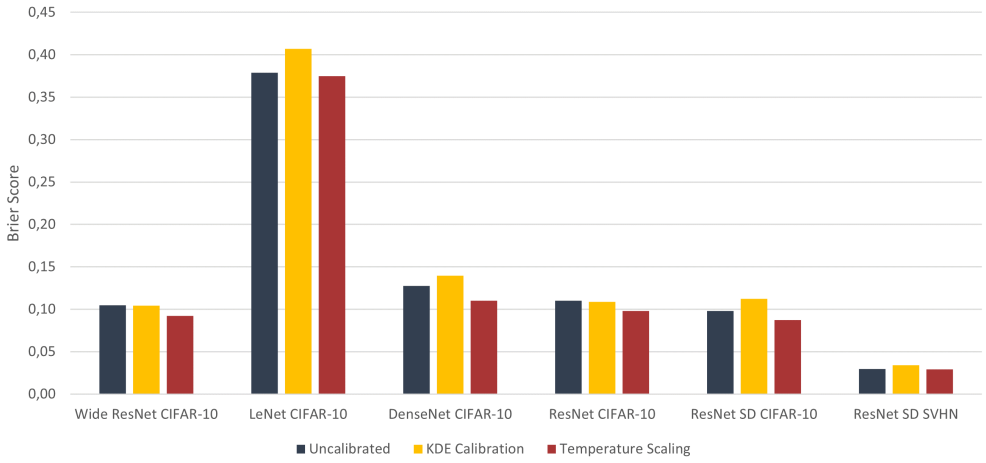

Figure 5.3: Visual Comparison of Uncalibrated, KDE Calibration and Temperature Scaling for the Brier

5.4 Accuracy

Presented next are the results based on the *Accuracy*.

Results with Accuracy				
Neural Network	Dataset	Uncalibrated	KDE Calibration	Temperature Scaling
Wide ResNet	CIFAR-10	**0,9394**	0,9299	**0,9394**
LeNet	CIFAR-10	**0,7275**	0,6968	**0,7275**
DenseNet	CIFAR-10	**0,9243**	0,9041	**0,9243**
ResNet	CIFAR-10	**0,9356**	0,9286	**0,9356**
ResNet SD	CIFAR-10	**0,9405**	0,9251	**0,9405**
ResNet SD	SVHN	**0,9816**	0,9795	**0,9816**

Table 5.4: Results in regards to accuracy

We can see (plotted on Figure 5.4) that Kernel Density Estimation Calibration is decreasing the accuracy of a neural network by approximately 1.4% while Temperature Scaling has the same accuracy as the uncalibrated network.

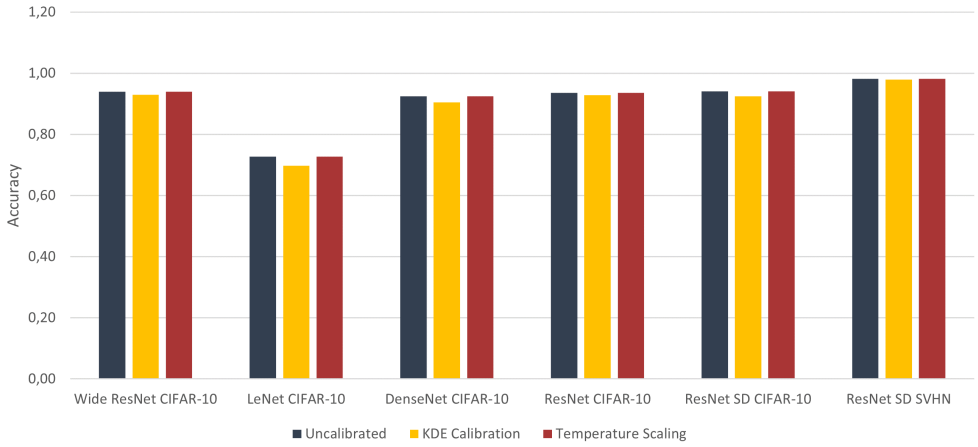

Figure 5.4: Visual Comparison of Uncalibrated, KDE Calibration and Temperature Scaling for the Accuracy

5.5 Comparison of Reliability

If we plot the Reliability Diagrams we can see the differences in Kernel Density Estimation and Temperature Scaling.

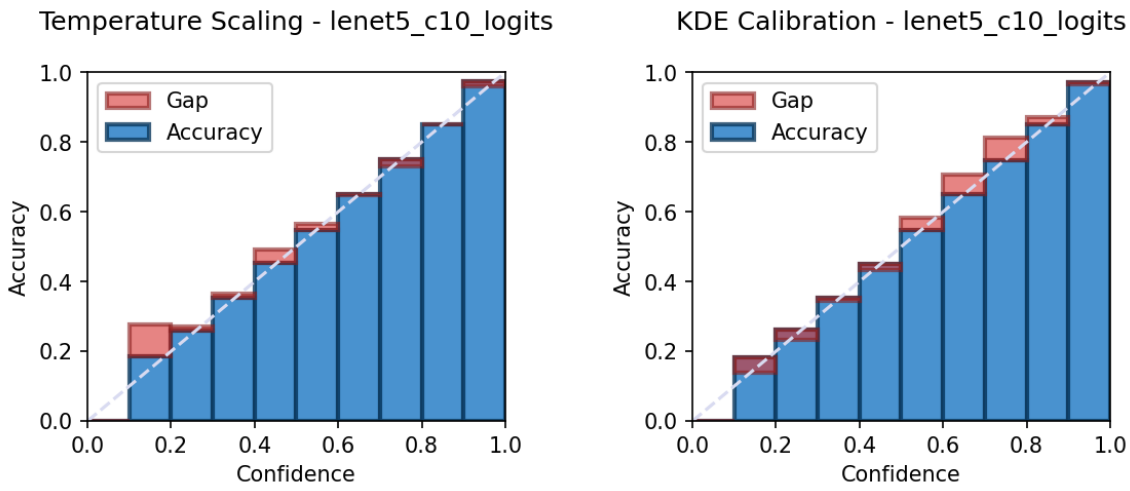

Figure 5.5: Left: Temperature Scaling of LeNet CIFAR-10.
Right: KDE Calibration of LeNet CIFAR-10

In the plots in Figure 5.5 we can examine that Temperature scaling is only slightly overconfident in the higher ranges, while slightly underconfident in the lower ranges. Kernel Density Estimation Calibration is a lot more underconfident in higher ranges while slightly overconfident in lower ranges. The difference in Expected Calibration Error for LeNet and CIFAR-10 is 0,0105 which is not insignificant.

In Figure 5.6 we can see that the average confidence of KDE Calibration is slightly lower than the accuracy for LeNet, while the average confidence and accuracy is nearly perfect for Temperature Scaling.

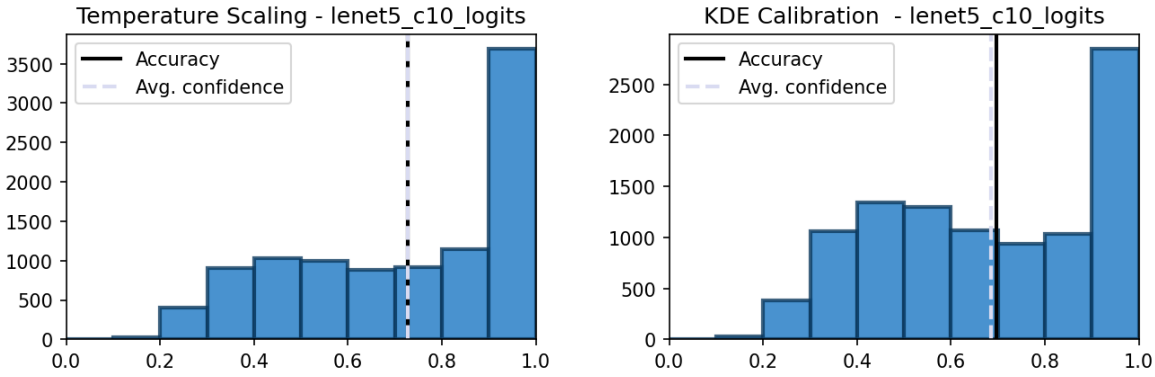

Figure 5.6: Left: Temperature Scaling of LeNet CIFAR-10.
Right: KDE Calibration of LeNet CIFAR-10

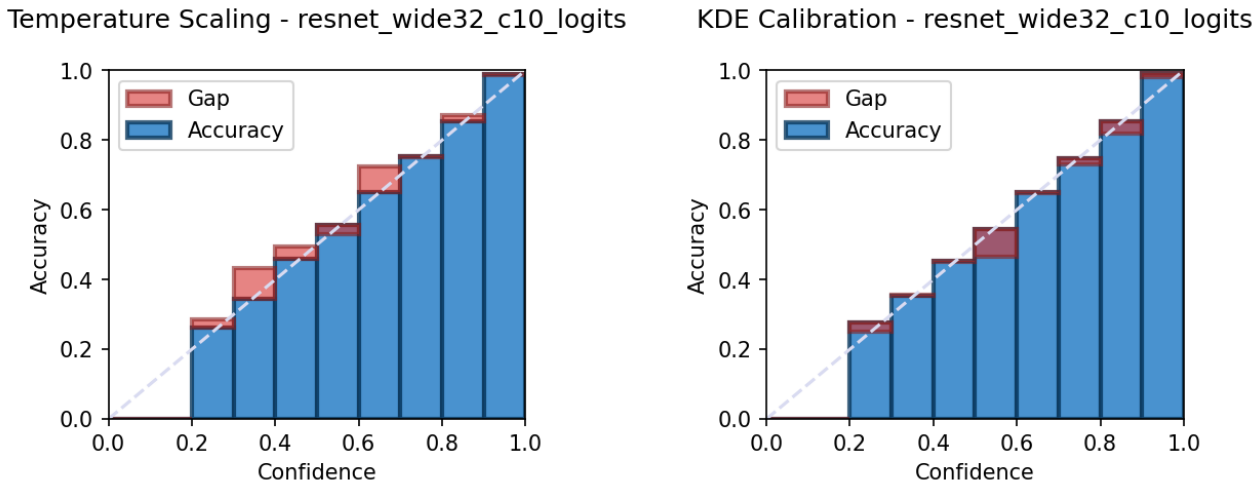

Figure 5.7: Left: Temperature Scaling of Wide ResNet CIFAR-10.
Right: KDE Calibration of Wide ResNet CIFAR-10

We can examine the same differences in the reliability diagrams for Wide ResNet CIFAR-10 in Figure 5.7. Temperature Scaling is slightly underconfident while Kernel Density Estimation is slightly overconfident.

5.5.1 Large multiclass datasets

We encountered an issue with datasets that have a high number of classes and a low number of validation logits per class. This includes CIFAR-100, ImageNet and Birds, all with 100 or more classes. We can not fit a multivariate kernel density estimation with more classes than observations per class.

For example, in the case of the CIFAR-100 dataset there are 5000 logits available for 100 classes, giving us 50 logits per class to fit the kernel density estimation to. This is not possible and we were therefore not able to provide results for these large multiclass datasets.

6 Discussion

In the following Chapter we will discuss the previous obtained results and draw conclusions about advantages and disadvantages of the proposed method of Kernel Density Estimation Calibration.

We can examine from the results obtained that the proposed kernel density estimation calibration is a viable method of recalibrating modern neural networks, even if it underperforms Temperature Scaling by a small margin.

The expected calibration error is consistently smaller by an average margin of approximately 0,03 to the uncalibrated network. It is only slightly worse than Temperature Scaling by a very small margin of 0,003.

The maximum calibration error is sometimes smaller then with Temperature Scaling, on average it underperforms Temperature Scaling only by approximately 0,04. It is consistently performing better than the uncalibrated networks in regards to the MCE.

On the other hand the Brier Score is consistently worse than the uncalibrated network while Temperature Scaling consistently outperforms the uncalibrated network.

The accuracy is also a lot worse with KDE calibration than it is with the uncalibrated network and with Temperature Scaling.

6.1 Advantages & Disadvantages

Even though Kernel Density Estimation calibration is viable for reducing the calibration error it also has many disadvantages to current calibration methods. Our implementation was much slower than Temperature Scaling. Especially the search for good hyperparameters was slow and should be parallelized in future implementations. The recalibration of all probabilities in a 10.000 logit test set takes around 30 seconds for KDE calibration compared to 1.5 seconds with Temperature Scaling.

It was not possible to recalibrate datasets with more than approximately 10 classes as we had too little datapoints to base the kernel density estimations on. A recalibration of datasets with many classes is possible with Temperature Scaling.

Another important disadvantage is, that the accuracy is lower after calibrating with kernel density estimation, whereas it is the same with Temperature Scaling. On average, kernel density estimation calibration decreases the accuracy by around 1.4%, which is quite high.

It could be viable to fit the multivariate kernel density estimations to more logits, as this should theoretically increase the accuracy of the underlying estimation and therefore decrease the ECE, MCE and Brier Score.

7 Conclusion

In the following section we will present the conclusion to this bachelor thesis by summarizing the proposed methodology and obtained results.

7.1 Proposed Methodology

After providing the reader with a theoretical background in machine learning, neural networks, calibration and kernel density estimation we examined the proposed methodology of fitting multivariate kernel density estimations to the logits of common neural networks to recalibrate the probabilities.

We are using a method similar to a Bayes Classifier to recalibrate the probabilities with multivariate KDEs as estimators for the dependent probability.

We explained the theory behind kernel density estimation and how it should reduce the calibration error of modern neural networks and described current recalibration methods that are being used such as Temperature Scaling, to which we later compared the proposed method.

7.2 Results

We examined that Kernel Density Estimation slightly underperforms Temperature Scaling in the important metrics Excpected Calibration Error and Brier Score as well as decrease the accuracy of the neural network by around 1.4% in our experiments. We discussed advantages and disadvantages of the proposed method.

7.3 Future Work

We propose fitting multivariate kernel density estimations to a bigger set of logits than we had available for the experiments as this could increase the calibration and decrease the negative effects on accuracy. With more logits kernel density

estimation calibration could also be tested on large multiclass datasets like CIFAR-100, Imagenet and Birds.

Bibliography

[Aki+19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. **Optuna: A next-generation hyperparameter optimization framework**. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, 2623–2631 (see pages 28, 29, 32).

[Ber+11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. **Algorithms for Hyper-Parameter Optimization**. In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger. Vol. 24. Curran Associates, Inc., 2011. URL: https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf (see page 29).

[Bri+50] Glenn W Brier et al. **Verification of forecasts expressed in terms of probability**. *Monthly weather review* 78:1 (1950), 1–3 (see pages 12, 13).

[Che+20] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. **A simple framework for contrastive learning of visual representations**. In: *International conference on machine learning*. PMLR. 2020, 1597–1607 (see page 16).

[CT96] Leda Cosmides and John Tooby. **Are humans good intuitive statisticians after all? Rethinking some conclusions from the literature on judgment under uncertainty**. *Cognition* 58:1 (1996), 1–73. ISSN: 0010-0277. DOI: https://doi.org/10.1016/0010-0277(95)00664-8. URL: https://www.sciencedirect.com/science/article/pii/0010027795006648 (see page 11).

[Dec86] Rina Dechter. **Learning while searching in constraint-satisfaction problems** (1986) (see page 6).

[Doc22] Scikit Documentation. *Underfitting vs. Overfitting*. 2022. URL: https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html (see page 8).

[Dos+20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. **An image is worth 16x16 words: Transformers for image recognition at scale**. *arXiv preprint arXiv:2010.11929* (2020) (see page 16).

[FC54] B. Farley and W. Clark. **Simulation of self-organizing systems by digital computer**. *Transactions of the IRE Professional Group on Information Theory* 4:4 (1954), 76–84. DOI: 10.1109/TIT.1954.1057468 (see page 4).

[Gar20] Colin Shunryu Garvey. **"AI for Social Good" and the First AI Arms Race Lessons from Japan's Fifth Generation Computer Systems (FGCS) Project**. In: *4eba5de577e580fd5b664f1a516856fd59274f1a8ad6658796c6 7b2c 34 56de (2020)*. 4e00822c793e56e36cd54eba 4eba5de577e580fd5b664f1a. 2020, 2O1ES501–2O1ES501 (see page 5).

[GFM18] Latifa Guesmi, Habib Fathallah, and Mourad Menif. "Modulation format recognition using artificial neural networks for the next generation optical networks." In: *Advanced Applications for Artificial Neural Networks*. IntechOpen, 2018, 11 (see page 5).

[Guo+17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. **On Calibration of Modern Neural Networks**. *CoRR* abs/1706.04599 (2017). arXiv: 1706.04599. URL: http://arxiv.org/abs/1706.04599 (see pages 1, 4, 11, 12, 14, 15, 18–21, 28).

[He+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. **Deep Residual Learning for Image Recognition**. *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (see pages 10, 31, 32).

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. **Deep residual learning for image recognition**. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 770–778 (see page 14).

[Heb49] Donald O. Hebb. **The organization of behavior: A neuropsychological theory**. New York: Wiley, June 1949. ISBN: 0-8058-4300-0 (see page 4).

[HLW16] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. **Densely Connected Convolutional Networks**. *CoRR* abs/1608.06993 (2016). arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993 (see pages 10, 31).

[How94] Jim Howe. 1994. URL: https://www.inf.ed.ac.uk/about/AIhistory.html (see page 5).

[HS99] Geoffrey Hinton and Terrence J Sejnowski. **Unsupervised learning: foundations of neural computation**. MIT press, 1999 (see page 7).

[Hua+16] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. **Deep Networks with Stochastic Depth**. *CoRR* abs/1603.09382 (2016). arXiv: 1603.09382. URL: http://arxiv.org/abs/1603.09382 (see pages 11, 32).

[Huy22] Chip Huyen. *Data distribution shifts and monitoring*. Feb. 2022. URL: https://huyenchip.com/2022/02/07/data-distribution-shifts-and-monitoring.html#data-shifts (see page 15).

[IS15] Sergey Ioffe and Christian Szegedy. **Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift**. *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167 (see page 18).

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. **Learning multiple layers of features from tiny images** (2009) (see pages 9, 31).

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. **Imagenet classification with deep convolutional neural networks**. *Advances in neural information processing systems* 25 (2012) (see page 16).

[LeC+89] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. **Backpropagation applied to handwritten zip code recognition**. *Neural computation* 1:4 (1989), 541–551 (see page 6).

[LeC+98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. **Gradient-based learning applied to document recognition**. *Proceedings of the IEEE* 86:11 (1998), 2278–2324 (see page 14).

[Lec+98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. **Gradient-based learning applied to document recognition**. *Proceedings of the IEEE* 86:11 (1998), 2278–2324. DOI: 10.1109/5.726791 (see pages 11, 32).

[Mah+18] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens Van Der Maaten. **Exploring the limits of weakly supervised pretraining**. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, 181–196 (see page 16).

[Min+21] Matthias Minderer, Josip Djolonga, Rob Romijnders, Frances Hubis, Xiaohua Zhai, Neil Houlsby, Dustin Tran, and Mario Lucic. **Revisiting the Calibration of Modern Neural Networks**. *CoRR* abs/2106.07998 (2021). arXiv: 2106.07998. URL: https://arxiv.org/abs/2106.07998 (see pages 11, 15, 16, 18).

[MM97] Tom M Mitchell and Tom M Mitchell. **Machine learning**. Vol. 1. 9. McGraw-hill New York, 1997 (see page 3).

[MP43] Warren S McCulloch and Walter Pitts. **A logical calculus of the ideas immanent in nervous activity**. *The bulletin of mathematical biophysics* 5:4 (1943), 115–133 (see page 4).

[MS69] Minsky Marvin and A Papert Seymour. **Perceptrons**. *Cambridge, MA: MIT Press* 6 (1969), 318–362 (see page 5).

[NC05] Alexandru Niculescu-Mizil and Rich Caruana. **Predicting good probabilities with supervised learning**. In: *Proceedings of the 22nd international conference on Machine learning*. 2005, 625–632 (see pages 14, 20).

[Net+11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. **Reading digits in natural images with unsupervised feature learning** (2011) (see pages 10, 31).

[Par85] D.B. Parker. **Learning-logic: Casting the Cortex of the Human Brain in Silicon**. Technical report: Center for Computational Research in Economics and Management Science. Center for Computational Research in Economics and Management Science, Alfred P. Sloan School of Management, Massachusetts Institute of Technology, 1985. URL: https://books.google.de/books?id=2kS9GwAACAAJ (see page 5).

[Rad+21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. **Learning transferable visual models from natural language supervision**. In: *International Conference on Machine Learning*. PMLR. 2021, 8748–8763 (see page 16).

[Ros56] Murray Rosenblatt. **Remarks on some nonparametric estimates of a density function**. *The annals of mathematical statistics* (1956), 832–837 (see page 21).

[Ros58] Frank Rosenblatt. **The perceptron: a probabilistic model for information storage and organization in the brain.** *Psychological review* 65:6 (1958), 386 (see pages 4, 5).

[Rui19] Pablo Ruiz. *Understanding and visualizing ResNets*. Apr. 2019. URL: https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8 (see page 10).

[Rus+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. **ImageNet Large Scale Visual Recognition Challenge**. *International Journal of Computer Vision (IJCV)* 115:3 (2015), 211–252. DOI: 10.1007/s11263-015-0816-y (see pages 9, 31).

[Rus10] Stuart J Russell. **Artificial intelligence a modern approach**. Pearson Education, Inc., 2010 (see page 6).

[Sam67] Arthur L Samuel. **Some studies in machine learning using the game of checkers. II—Recent progress**. *IBM Journal of research and development* 11:6 (1967), 601–617 (see page 5).

[SB18] Richard S Sutton and Andrew G Barto. **Reinforcement learning: An introduction**. MIT press, 2018 (see page 7).

[Sta22] Statsmodels. *Statsmodels.nonparametric.kernel$_d$ensity.kdemultivariate00b6*. June 2022. URL: https://www.statsmodels.org/dev/generated/statsmodels.nonparametric.kernel_density.KDEMultivariate.html (see pages 29, 30).

[Tho22] Bergur Thormundsson. *Global AI funding by Quarter 2011-2021*. Mar. 2022. URL: https://www.statista.com/statistics/943151/ai-funding-worldwide-by-quarter/ (see page 6).

[Tol+21] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. *MLP-Mixer: An all-MLP Architecture for Vision*. 2021. DOI: 10.48550/ARXIV.2105.01601. URL: https://arxiv.org/abs/2105.01601 (see page 16).

[Wel+10] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. **Caltech-UCSD Birds 200**. Tech. rep. CNS-TR-201. Caltech, Jan. 1, 2010. URL: /se3/wp-content/uploads/2014/09/WelinderEtal10_CUB-200.pdf,%20http://www.vision.caltech.edu/visipedia/CUB-200.html (see pages 10, 31).

[Wer74] Paul Werbos. **Beyond regression:" new tools for prediction and analysis in the behavioral sciences**. *Ph. D. dissertation, Harvard University* (1974) (see page 5).

[Wik22] Wikipedia. *Kernel Density Estimation*. July 2022. URL: https://en.wikipedia.org/wiki/Kernel_density_estimation (see pages 8, 21, 24).

[ZD12] Adriano Zanin Zambom and Ronaldo Dias. *A Review of Kernel Density Estimation with Applications to Econometrics*. 2012. DOI: 10.48550/ARXIV.1212.2812. URL: https://arxiv.org/abs/1212.2812 (see pages 21–23).

[ZE01] Bianca Zadrozny and Charles Elkan. **Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers**. In: *Icml*. Vol. 1. Citeseer. 2001, 609–616 (see page 19).

[ZK16] Sergey Zagoruyko and Nikos Komodakis. **Wide Residual Networks**. *CoRR* abs/1605.07146 (2016). arXiv: 1605.07146. URL: http://arxiv.org/abs/1605.07146 (see page 32).