university of groningen / faculty of science and engineering / bernoulli institute

Department of Artificial Intelligence
University of Groningen, The Netherlands

and

Department of Intelligent Imaging
Netherlands Organization for Applied Scientific Research (TNO)
The Hague, The Netherlands

Uncertainty Estimation for Object Detection

Master's Thesis
Yannik Beckersjürgen
s2303779

Primary supervisor: Dr. M. A. Wiering
Secondary supervisor: F. Hillerström

Groningen, 15th June, 2019

Abstract

In this thesis, we investigate whether additional uncertainty estimates can be added to the output of a neural network that is used for object detection. We are mainly interested whether dropout, a stochastic regularization technique, can be utilized for this purpose. When applied to the decision referral task in which one wishes to identify and exclude the most difficult inputs from the dataset, dropout does not perform better than the standard network output. In addition, using dropout to estimate bounding-box regression uncertainty performed worse when compared to modeling the error with a Laplace distribution. However, we found that an adjusted form of dropout performed well for detecting a change in data distribution between daylight and night images. We further investigated whether regression uncertainty or an autoencoder reconstruction error could be used to identify novel object categories of interest, which was found not to be the case. Overall, we find that prior research on classification tasks does not hold for our object-detection context.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning (ML) is a branch of computing science that deals with algorithms and models that solve complex problems by learning from data. This is particularly relevant to real world applications in which it is impossible to determine the appropriate response for any state in the world that the system might encounter in advance. An example for this is autonomous driving, in which an autonomous vehicle needs to be able to perceive and navigate a complex and dynamic environment.

Being able to solve tasks which would be out of reach for traditional iterative algorithms by learning from data is the core strength of ML, but this makes it also difficult to analyze the risk of models that are employed in the real world. A well-known failure of an ML model occurred when an image-classification model by Google misclassified photos of black people as gorillas [50], resulting in accusations of racism. Another widely-reported failure case involves a self-driving car from Uber which hit and killed a pedestrian while driving at night [42]. As research on ML and artificial intelligence (AI) progresses, it is clear that ML algorithms will contribute to more and more tasks in which safety is a critical concern. For instance, just recently it was suggested by DARPA to use AI for autonomous air-to-air combat [1]. In light of these examples, it is not difficult to imagine a scenario in which the failure of a ML model may result in disastrous consequences.

In order to assess these risks, it is essential to deliberately address the uncertainty that is associated with the predictions of ML models in situations for which failure is associated with significant costs. This subject is of practical interest for the Intelligent Imaging department at TNO[1], where the experiments discussed in this thesis were performed.

1.1 Neural Networks

Artificial neural networks (NNs) are inspired by observations of biological neurons and learning. NNs are ML models which transform one or more inputs into one or more outputs by propagating and transforming activations through a network of artificial neurons. Standard feedforward NNs perform a sequence of linear transformations, possibly interspersed with non-linear element-wise activation functions. Training these models is done in a supervised fashion by tuning the weight parameters that determine the linear transformations such that the error between the predicted output and the ground truth on known data is minimized. As ML models, NNs have been very successful in the last few years in a variety of different domains. Two of the most prominent domains in which NNs achieved state-of-the-art performance are image recognition [36] and speech recognition [23]. The breakthrough

[1]Netherlands Organisation for Applied Scientific Research

performances in these tasks are a significant factor that led to the revival of NN research and successful application to a multitude of other tasks.

Despite their versatility, one of the largest drawbacks of deep NNs is that it is difficult to interpret their inner workings and how the model predictions were arrived at. Although there exist some domain-specific methods of gaining insights into deep NNs, such as feature visualization [66], generally the view of NNs as blackboxes remains relevant.

1.2 Uncertainty Estimation for Neural Networks

The two most common outputs of NNs are unbounded regression outputs (e.g. for predicting numerical values) and classification outputs with confidence scores ranging between 0 and 1. Generally, regression estimates do not provide any indicator for prediction uncertainty. For classification, the frequently used softmax output provides an indication of prediction confidence, but interpreting these as the probability of being correct on test data data can be problematic. Adversarial examples, that is inputs which have been deliberately altered to fool the ML system, illustrate the point that softmax confidence can be high even when the model clearly makes a wrong prediction [20]. Calibration methods [24] exist which transform the model output to closer match the actual probability of a correct prediction, but this relies on the availability of an additional validation set and the assumption that this set is representative for all the possible data that the ML system might have to deal with.

The learned parameter values of regular NN architectures are generally maximum-likelihood or a posteriori point estimates. This means that training a neural network will produce a single value for each parameter in the network, which are then used for making predictions.

In contrast, Bayesian approaches attempt to infer the posterior distribution of the weights given the data. Predictions are then obtained by sampling parameters from the posterior distribution and computing the prediction for each draw, thereby approximating the expectation of the prediction [3]. Arguably, this is a more robust way of making predictions because we are then also provided with the variance in the predictions that is due to changes in model parameters. The main drawback of Bayesian approaches is that it is often intractable to compute the posterior distribution of the model parameters. In practice, the posterior distribution can be approximated for neural networks and many other models, but this still comes at a relatively high computational cost (e.g. [22], [3]). In addition, the number of model parameters increases when distributions over parameter values are learned instead of simply learning the values itself. In [3], a Gaussian posterior distribution is learned for each weight, thereby doubling the number of parameters since each weight is now a distribution with mean and standard deviation as sufficient statistics.

1.2.1 Types of Uncertainty

The overall predictive uncertainty of a model can be decomposed into *aleatoric* and *epistemic* (or *model*) uncertainty [33]. While aleatoric uncertainty depends on characteristics of the input (such as sensor noise), epistemic uncertainty is caused by insufficient knowledge about which parameters best model the data. In contrast to aleatoric uncertainty, epistemic uncertainty can be reduced by collecting additional data. Bayesian approaches which learn distributions over the weights therefore mainly address the issue of epistemic uncertainty.

1.2.2 Dropout

Dropout is a regularization technique that was originally devised to prevent overfitting to the training data in NNs by randomly dropping activations during training, forcing the networks to avoid over-reliance on specific artificial neurons and connections [60].

Recently, it was shown [14] that using dropout in NNs approximates Bayesian inference in a deep Gaussian process [7]. In this interpretation, the distribution of the weights (subject to dropout) is an approximation to the posterior weight distribution. This means that model uncertainty can be obtained by performing forward passes through networks which differ in their weights due to dropout. Compared to traditional Bayesian NNs, this approach is easy to train while still providing theoretically grounded estimates of epistemic uncertainty at inference time. With respect to aleatoric uncertainty, it was suggested to let the model learn data-dependent uncertainty by itself [33]. In the regression case with a scalar output, the model output can for example be set to be Gaussian and the model is trained to predict both the mean and the variance of the prediction.

1.3 Object Detection Networks

In computer vision, image classification is a relatively simple task since only an image-level category needs to be correctly determined. In contrast, object detection systems need to indicate for each input image the location and classification of object categories of interest in the image. To localize objects, most frequently a two-dimensional bounding box is provided, although other shapes may be used in some contexts, such a three-dimensional bounding boxes or stick-figure like shapes for posture detection. Recent object-detection networks often provide dense predictions for images by predicting object category and bounding box offsets for each of a large number of default bounding boxes of different sizes, which cover the image in a grid-like fashion (e.g. [44], [46]).

1.4 Research Questions

In this thesis we investigate the research question of how additional uncertainty estimates can be incorporated into an object-detection network and whether they provide added value compared to the standard NN output. Due to the large model size of object-detection networks, it is promising to consider dropout uncertainty as it does not lead to an increase in model complexity compared to other Bayesian NNs.

To obtain evaluation metrics for the uncertainty measures, we can consider applications in which additional uncertainty estimates may be of value and then examine whether this expectation is met. We aim to answer the following sub-questions:

1. **Calibration**. Are the mean predictions of the stochastic forward passes better calibrated compared to standard forward passes? In other words, do predictions, when binned into confidence ranges, have an empirical accuracy that corresponds to the confidence scores when evaluated on a test set?

2. **Decision referral**. Can the uncertainties be used for decision referral in object detection? This deals with the question as to whether accuracy is improved

when we refer decisions to an expert or *oracle* based on the computed uncertainty statistics, relative to the standard NN classification output. Using dropout uncertainty for decision referral has been investigated in an image classification task in a medical context and was found to work well [41]. In contrast, using the standard classification output of the network as the referral criterion was not sufficient to observe a reliable improvement in performance. We will later examine whether these findings can be extended from the image classification to the object-detection domain.

3. **Detecting covariate shift.** Domain shift, also known as dataset shift or distributional shift, describes the situation in which an ML model encounters out-of-distribution (OOD) data which differs from the distribution of the dataset that was used for training and validation. Covariate shift is a special case of dataset shift, in which only the input distribution changes. More specifically, while the distribution of the inputs to the model changes, the conditional probability of the outputs given the inputs stays constant [54].

   From a practical point-of-view, it would be useful to be able to determine when such a distributional shift takes place. One way to approach this problem is to use the predictive uncertainty estimates to discriminate between in-distribution and OOD data, following the assumption that OOD data will be associated with higher predictive uncertainty, on average. This has been investigated on a simple image classification data set using the predictive uncertainty derived form ensembles [38], Bayesian NNs [47], and dropout [41].

   In object detection, the ground truth class for the large majority of inputs to the system corresponds to the background class, which is the ground truth for all parts of the image which do not contain an object of interest. This means that object-detection systems encounter many inputs that do not belong to one of the target classes during training. It is plausible that this makes the classification part more robust when compared to standard classifiers which perform mutually-exclusive and exhaustive classification (without a generic non-target class).

   In this context we aim to answer the following question: Can dropout uncertainty be used to detect when covariate shift takes place?

4. **Regression uncertainty.** Can dropout uncertainty be used to estimate the uncertainty of the bounding box location and dimensions that are predicted in object detection? How does this compare to learned (aleatoric) uncertainty estimates that are obtained by explicitly assuming and predicting a distribution for the regression residual?

1.5 Outline

The thesis is structured as follows: In chapter 2, we begin by discussing the theoretical background for NNs and describe how modern NNs are set up and trained. In chapter 3, we describe uncertainty estimation for NNs in more detail, including Bayesian approaches and dropout uncertainty. In chapter 4, we discuss recent object-detection networks and in chapter 5 we describe and discuss the experiments that were performed to answer the research questions that were posed above. Finally, in chapter 6 we provide a conclusion on our findings and suggest future work.

Chapter 2

Neural Networks

2.1 Supervised Learning

The notion of a label or annotation is an important concept in ML. For instance, in a dataset consisting of images of one car per image, the labels may for instance be the known (ground-truth) brand of the car depicted in the image. Supervised learning is one of the three branches of ML [1] and deals with the question how an ML system can learn to approximate a function that maps inputs to the correct outputs, given an annotated set of input-output data [51].

2.1.1 Classification

In classification tasks, the ML system should learn to determine to which categories out of a predetermined list of possible categories a given input pattern belongs. The aforementioned dataset of images of cars from different brands is an example for such a classification task. The dataset can be written as

$$\mathcal{D} = \{X, Y\} = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) \right\}_{i=1}^{N}$$

Here $\mathbf{x}^{(i)}$ denotes the input in a vector representation that will be presented to the system. $y^{(i)} \in \{1, ..., C\}$ denotes the corresponding ground truth class membership out of the C classes. The classification task is therefore to train an ML system such that it learns to perform a mapping from input to label $f : \mathbb{R}^{\mathrm{d}} \to \mathbb{R}$, where d is the dimensionality of the input. If C = 2, this is called binary classification and when a pattern may belong to more than one category this is referred to as multi-label classification, in which case we can write

$$\mathcal{D} = \left\{ \left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)} \right) \right\}_{i=1}^{N}$$

using a one-hot encoding to allow for membership of multiple categories.

2.1.2 Regression

Supervised regression is very similar to classification, but instead of predicting a class, the ML model should predict a numeric value or vector of values, such as the stock price of different companies. The aim is therefore for the model to learn the

[1]The other two branches are unsupervised learning, which has the goal of identifying structure in unlabeled data, and reinforcement learning, which concerns itself with how artificial agents can adjust their behavior by learning from interacting with their environment while receiving rewards and punishments.

mapping

$$f : \mathbb{R}^{\mathrm{d}} \to \mathbb{R}^{\mathrm{c}},$$

where c is the dimensionality of the codomain, the regression target. The annotated data for training and testing the regression estimator can again be written as

$$\mathcal{D} = \left\{ \left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)} \right) \right\}_{i=1}^{N}.$$

2.1.3 Training and Validation

All supervised learning algorithms rely on annotated data for training and evaluating the performance of the ML model. It is common practice to split the dataset into three mutually exclusive subsets of data points and corresponding annotations. The first subset is the training set, which is used by the training algorithm itself during training. Next, the validation set is a test set that can be used to assess the performance of the model throughout the training on data other than the one that is used for the actual training. The validation set can also be used for optimizing the hyperparameters of the model and training procedure, meaning the parameters which have to be set before the optimization routine commences. Finally, the test set is used to get an objective indication of model performance by looking at data that was not used during training at all, neither for the training algorithm itself nor for hyperparameter selection.

2.2 Neural Networks

2.2.1 Perceptron

As the name suggests, the building blocks of NNs are artificial neurons. The Rosenblatt perceptron [56] is a simple linear model and can be considered a milestone in the development of ANNs. Mathematically, the perceptron is a function that maps the input vector $\mathbf{x}$ to a binary output scalar:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Here, the input vector $\mathbf{x}$ has real values and represents the encoding of the problem input, for instance a list of grades which are used to predict the binary target value y of passing an exam. In this example, a prediction of $\hat{y} = 1$ corresponds to a predicted pass, while $\hat{y} = 0$ predicts the student to fail. The real-valued weights $\mathbf{w}$ of the perceptron (one weight per input dimension) determine how the individual input variables predict the outcome and the bias scalar b determines the magnitude of the dot product $\mathbf{w} \cdot \mathbf{x}$ that is required for a positive prediction. Thus, this model is very similar to a logistic regression, but providing the most likely outcome as output instead of the outcome probability. Rosenblatt provided an algorithm for finding the weights to solve a problem like the one above. In the *perceptron training rule*, the weights are updated iteratively, which is typical for ML models. The availability of this procedure contributed to the popularity of this simple model. Moreover, convergence is theoretically guaranteed if the training data is linearly separable, i.e. there exists a hyperplane in the input dimensions that can separate data points belonging to the positive and the negative class.

After the initial excitement surrounding the perceptron and its learning algorithm, it later became clear that the perceptron as a machine learning model is subject to important limitations [25]. Given two boolean input variables, the model cannot replicate the XOR function, meaning that the model should provide a positive response if and only if exactly one of the two inputs is positive. This is just one instance that illustrates the limitation that the perceptron can only solve problems that are linearly separable, which was pointed out in [49].

This finding contributed to stagnation in research on artificial neural networks for several years.

2.2.2 Multi-layer Perceptrons (MLPs)

Progress was made by arranging artificial neurons into layers (meaning that the input to the layer is connected to all neurons in the layer) and feeding the neuron activations of each layer as input to the subsequent layer. The number of nodes[2] in a layer is called its *width*. Such configurations are known as feedforward NNs, or multi-layer perceptrons (MLPs). MLPs have one or more hidden layers that connect the input layer to the output layer.

Neural networks are called networks because they are formed by a composition of functions [21]. An MLP with one hidden layer can be written as $f^{(2)}(f^{(1)}(\mathbf{x}))$, where $\mathbf{x}$ denotes the input and $f^{(1)}$ and $f^{(2)}$ denote the hidden and output layer respectively. The number of layers of a neural NN is called its *depth*. Frequently, only the number of hidden layers are counted since all NNs have an input and output layer. The lth layer of a MLP is a function

$$f^{(l)}(\mathbf{x}; \mathbf{W}^{(l)}, \mathbf{b}^{(l)}) = a(\mathbf{W}^{(l)\top}\mathbf{x} + \mathbf{b}^{(l)}) \tag{2.2}$$

Here, $\mathbf{W}^{(l)}$ is the weight matrix that performs a linear transformation on the input $\mathbf{x}$ to the layer. This is very similar to Equation 2.1, but since layers consist of multiple neurons, there is a vector for each neuron in the layer that describes how the neuron responds to different input values. Together, all the weights are arranged in the matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{m \times n}$, where m is the number of input dimensions and n is the number of nodes in the layer. Thus, each column of $\mathbf{W}^{(l)}$ is a vector of weights which determine how a different neuron responds to the layer input. Similary, the bias term $\mathbf{b} \in \mathbb{R}^n$ is now a vector instead of a scalar. Finally, $a(\cdot)$ denotes a pointwise activation function. If no special activation operator is used, this is just the identity function, though non-linear activation functions like the sigmoid are frequently used in the hidden layers.

These functions determine *how* neuron firing (or activation) is implemented in the NN. For instance, the conditional expression in Equation 2.1 indicates that the perceptron has a step activation (all-or-nothing) function. Activation functions are discussed in more detail in section 2.5. In the output layer, the number of nodes and the activation function depends on the dimensionality and type of the required output. By combining multiple layers with non-linear activation functions MLPs are now capable of approximating complex non-linear functions.

[2]We use the terms neurons, nodes, and units interchangeably.

2.3 Likelihood and Loss

As we have seen previously, NNs are parametrized by a weight matrix and bias vector for each layer after the input layer. It was shown that the feedforward architecture of MLPs makes them universal function approximators [31]. In other words, under mild assumptions, MLPs with one or more hidden layers and a sufficient number of neurons are capable of representing any continuous function. The caveat here is that this is a only a theoretical guarantee that suitable parameter values exist and the question as to whether they can be found algorithmically remains. How would one proceed to find these values? In order to do so, one needs a criterion to evaluate how well a set of parameters solves the task at hand.

Cost functions (also known as loss functions) are used to assign ML model parameters a numeric score based on how the model performs on a set of data. Considering that the cost of a set of parameters depends on the agreement between model predictions and ground truth labels, different cost functions are used depending on the NN output.

2.3.1 Classification

For classification, it is common that the model outputs a vector of probabilities predicting class membership for all categories. For each input pattern $\boldsymbol{x}^{(i)}$ that the model classifies, we then have the predicted probability for each class c of the pattern belonging to that category, which we write as

$$P_{model}\left(y^{(i)} = c|\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right),$$

where $\boldsymbol{\theta}$ is the set of all learned parameters of the model. The conditional likelihood function describes the probability that the model with parameters θ assigns to a dataset $\mathcal{D}$:

$$\mathcal{L}(\theta,\mathcal{D}) = P(\boldsymbol{Y}|\boldsymbol{X};\boldsymbol{\theta}) \tag{2.3}$$

$$= \prod_{i=1}^{N} P_{model}\left(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right) \tag{2.4}$$

Note that the second equality assumes that the data is independently and identically distributed (i.i.d) [21]. For parameter selection, one wants to choose parameters which have a high likelihood for the dataset. For classification, as can be seen from Equation 2.4, this is the case when the model assigns high probabilities to the ground truth labels $y^{(i)}$ given the input vectors $\mathbf{x}^{(\mathbf{i})}$. For numerical reasons, it is common to use log-likelihood (LL):

$$\text{LL} = \sum_{i=1}^{N} \log P_{model}\left(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right) \tag{2.5}$$

By convention, a smaller loss indicates better model performance. Since the sum in Equation 2.5 lies in the range $(-\infty, 0]$, the negative log-likelihood (NLL) is a more suitable loss function since it lies in the range $[0, \infty)$, where perfect model accuracy produces a loss of 0. In information theory, the NLL is also known as cross entropy.

$$\text{NLL} = -\sum_{i=1}^{N} \log P_{model}\left(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right) \tag{2.6}$$

Given this cost function, the maximum likelihood estimate (MLE) for the model parameters is given by

$$\begin{aligned}\boldsymbol{\theta}_{\mathrm{ML}} &= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{N} P_{model}\left(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right) \\ &= \arg\min_{\boldsymbol{\theta}} -\sum_{i=1}^{N} \log P_{model}\left(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right)\end{aligned} \tag{2.7}$$

2.3.2 Regression

For regression, the mean squared error (MSE) is one of the most frequently used cost functions. Given a dataset containing N data points, the MSE is obtained by averaging the squared error between the prediction $\hat{\mathbf{y}}^{(i)}$ and the corresponding ground truth regression target $\mathbf{y}^{(i)}$.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}\right)^2 \tag{2.8}$$

By squaring the error, large deviations are much more impactful compared to the similar mean absolute error (MAE). Minimizing the MSE and the MAE is equivalent to minimizing the L2 and L1 norm of the error respectively. Note that the MSE is always positive and is small for good models and model parameters because deviations from the ground truth will be small, on average.

2.3.3 Bayesian Model Selection

Above, we have discussed the likelihood, the probability of observing the (training) data $\mathcal{D}$ given the values of the model parameters, as a criterion for model selection. Although it is straightforward to formulate a suitable loss function for MLE, we implicitly assume the distribution of the parameters themselves does not play a role. Applying Bayes' theorem to the problem of model selection opens up a different perspective. Recall that Bayes' theorem (in this context) can be stated as

$$\begin{aligned}P(\boldsymbol{\theta}|\mathcal{D}) &= \frac{P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\mathcal{D})} \\ \text{posterior} &= \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}\end{aligned} \tag{2.9}$$

The *evidence* is the probability of the training data, marginalizing the conditional probability over all possible parameter values:

$$P(\mathcal{D}) = \int P(\mathcal{D}|\boldsymbol{\theta}) \cdot P(\boldsymbol{\theta})d\boldsymbol{\theta} \tag{2.10}$$

Note that for models with a large parameter space like NNs, computing this probability is infeasible as it would require to evaluate the likelihood for all parameter values, which quickly becomes intractable when the parameter space is continuous. Finding the posterior distribution of the model parameters is the main goal of Bayesian inference. If we provide the posterior distribution, it is straightforward to sample parameters from this distribution and by comparing the predictions from these different models, we can compute the mean prediction and the variance (or uncertainty) around this prediction. This is arguably much more robust than just using a single configuration of parameters that is obtained by MLE. However, due to the difficulty of evaluating the *evidence*, more involved techniques are needed in order to approximate the posterior. This will be discussed in more detail in section 3.4.

A somewhat simpler and therefore more common way of leveraging Bayes theorem is maximum a posteriori (MAP) estimation. The MAP parameters are given by

$$\begin{aligned}
\boldsymbol{\theta}_{\text{MAP}} &= \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{\theta}|\mathcal{D}) && (2.11)\\
&= \arg\max_{\boldsymbol{\theta}} \frac{P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\mathcal{D})} && (2.12)\\
&= \arg\max_{\boldsymbol{\theta}} P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta}) && (2.13)\\
&= \arg\max_{\boldsymbol{\theta}} \log(P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})) && (2.14)\\
&= \arg\max_{\boldsymbol{\theta}} \log P(\mathcal{D}|\boldsymbol{\theta}) + \log P(\boldsymbol{\theta}) && (2.15)
\end{aligned}$$

Note that the MAP parameters are just the mode of the posterior distribution. Since the *evidence* is just a normalization factor, we do not have to evaluate it in order to perform MAP estimation. The MAP estimate differs from the MLE by placing a prior distribution over the parameters, e.g. independent Gaussian distributions. This prior distribution is a reflection of our believes about the model parameters before seeing any data. If we think that the model should not have too many weights with large magnitudes, we could use a Gaussian distribution with a small standard deviation for example. Note that in Equation 2.15, the argument to the arg max operator is just the log-likelihood from Equation 2.5 with an added term that penalizes parameters that are given a low probability by the prior distribution. As a cost function, we would just take the negative of the argument.

2.4 Optimization

After specifying a model family and choosing a suitable cost function, such as the MSE for regression and the NLL for classification tasks, we need to find a way of finding values for the model parameters which minimize the cost function with respect to the training data. In contrast to simple linear models like linear regression, the cost function of MLPs is generally non-convex due to the non-linearity of the models. As a consequence, finding the optimal model parameters analytically is infeasible and there is no guarantee that the best solution will be found, in contrast to other ML models with convex losses, such as support vector machines (SVMs) for example [21]. These circumstances make iterative optimization strategies the main choice for training NNs. In general, they start with initial parameter values and repeatedly adjust these values such that the loss is minimized as far as possible.

2.4.1 Parameter Initialization

The way in which the weights and biases of NNs are initialized is an important requirement in order for the training to be successful. In practice, NN parameters are most commonly initialized by some random procedure in order to aid the functional differentiation of individual neurons. If all weights were initialized with equal values, they would all contribute to the output in the same way, thereby making optimization based on the contribution of each artificial neuron to the output impossible.

A common initialization method is to use a Gaussian distribution with zero mean and a small standard deviation, e.g. $\sigma = 0.01$ [36]. As far as biases are concerned, setting them to 0 or 1 are the most common choices. In order to ensure that the input signal does not grow or shrink exponentially as it flows through the network, He et al. proposed to initialize the weights as follows [27]:

$$w_{lk} \sim \mathcal{N}(0, \sqrt{2/n_l}) \tag{2.16}$$

Here, w_{lk} is the value of the *kth* weight in the *lth* layer of the NN and n_l is the dimensionality of the input to neurons in layer l (assuming that the input is a single vector). All biases are set to zero. This approach takes into account how the variance of the activations changes as they are passed from layer to layer in the NN and is specifically tailored to work well with the ReLU activation function that will be described in section 2.5. Using somewhat more complex initialization strategies like He initialization is especially important in deep neural networks since any problems caused by bad parameter initialization are bound to be compounded by the layered architecture of NNs.

2.4.2 Gradient Descent

Formally, in the supervised setting, the cost $\mathcal{J}$ on the training set $\mathcal{D}$ for the model parameterized by $\boldsymbol{\theta}$ is given by

$$\mathcal{J}(\boldsymbol{\theta}, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{J}\left(\boldsymbol{\theta}, \mathbf{x}_i, \mathbf{y}_i\right) \tag{2.17}$$

This equality usually holds true because cost functions are often formulated such that the total cost is the average of the cost incurred for individual data points of $\mathcal{D}$. This average is also known as the empirical risk in decision theory [51].

Gradient descent is a widely applicable optimization algorithm that is used to find a local minimum of a function and is also used to train NNs. In gradient descent, we wish to find

$$\min_{\boldsymbol{\theta}} \mathcal{J}\left(\boldsymbol{\theta}, \mathcal{D}\right) \tag{2.18}$$

Recall that in vector calculus, the gradient is a generalization of the derivative and points in the direction of steepest increase in the function value. If the loss function is differentiable with respect to its parameters (which is generally the case for NNs) and given the parameters at iteration t, the parameter updates are performed as follows:

$$\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t - \eta \nabla \mathcal{J}(\boldsymbol{\theta}_t, \mathcal{D}) \tag{2.19}$$

Thus, for any given point in the parameter space, gradient descent takes a step of a previously determined proportion (η) in the opposite direction to the gradient of the function at that point, thereby systematically finding parameters values which decrease the value of the function. Here, $\nabla \mathcal{J}$ denotes the gradient of $\mathcal{J}$. This is repeated for a certain number of steps or until the solution converges. In this manner, a local minimum will be found if the step size is chosen appropriately, the function is sufficiently regular, and the initialization is not too far from the local optimum [4]. The backpropagation algorithm [57] is a practical implementation of gradient descent for MLPs and works by *backpropagating* the observed prediction error through the network and computing the gradients for each layer by recursively applying the chain rule from calculus.

Gradient Descent for Neural Networks

Neural networks are parameterized by the weights and biases of the neurons in the network. Given the cost function for the NN, such as MSE or the NLL, the parameters are updated by computing the partial derivatives of the cost function with respect to each weight and bias in the network and consequently performing the update in Equation 2.19.

Stochastic Gradient Descent

Gradient descent, as described above, takes the complete training set into account when computing the loss at each step. If the step size η is not too large, the updates performed by gradient descent will reduce the loss in a greedy fashion in each iteration. However, there are a number of drawbacks as well. Using standard gradient descent, computing the gradient has computational complexity of $\mathcal{O}(N)$ which may become prohibitive for large datasets [21]. Since the empirical risk is just an average, one may instead attempt to approximate Equation 2.19 by computing the gradient over a subset $\mathcal{B}$ of the training set for each parameter update. Such subsets are called minibatches and are typically drawn randomly from the training set and the batch size depends on the model, the problem type, and memory restrictions. In general, it is common to use batch sizes that vary from 1 to hundreds of data points for each minibatch [21]. Gradient descent using minibatches instead of the entire training set is known as stochastic gradient descent (SGD). While larger batch sizes require more computation per parameter update, the updates are also more precise. While this may seem to be an advantage compared to smaller batch sizes, it has been found that large batch sizes correspond to *sharper* minima in the loss function compared to smaller batches [34]. Compared to narrow minima, flat minima are less specific to the training data and therefore typically generalize better to new data. This finding can be attributed to the fact that the stochastic updates of SGD find more robust solutions by occasionally performing updates which increase the loss when computed over the whole dataset.

Gradient Descent with Momentum

A few modifications to the vanilla gradient descent algorithm have been suggested which improve the training performance for NNs in practice. A simple and commonly used change is the addition of a momentum term to the procedure described above. More specifically, parameter updates are made as described above, but the algorithm also keeps track of a momentum term which is added to the update. The momentum term is an exponentially weighted first moment of the last few gradient updates and its contribution to the parameter update is controlled by a decay parameter α. In other words, if several gradient updates are made in the same direction, the updates increase in velocity in that direction due to the momentum term and if the gradient suddenly changes direction, the update has some inertia in the diretion of the previously accumulated velocity. The addition of the momentum term can accelerate the learning procedure and help the algorithm to traverse saddle points and local minima in the loss function and comes at little computational cost.

2.4.3 RMSprop and ADAM

The Root Mean Square Propagation (RMSprop) algorithm [30] is another modification of gradient descent which improves the learning procedure by keeping track of an exponentionally decaying second moment (pointwise square) of the gradient and divding the gradient by the square root of this moving average. This has the effect of attenuating gradient updates when large steps have been made and increasing the magnitude if previous updates have been small. Note that this effect takes place on a per-parameter level, thereby effectively providing an adaptive learning rate for each parameter in the model. This change can dampen oscillations (think of updates with opposite sign repeatedly overshooting the optimal path through the loss landscape) and increase the speed of learning in flat parts of the loss function. *Adam* [35] is a more advanced gradient optimization algorithm and derives its name from *adaptive moment estimation*. It effectively combines the idea of SGD with momentum and the per-parameter learning rate of the RMSprop algorithm and adds a few minor improvements, such as correcting for bias in the second moment during early episodes of training [21].

2.5 Activation Functions

Activation functions have been described in Equation 2.2 as (generally pointwise) functions that determine how *firing* of artificial neurons is implemented. They are an important part of NNs because non-linear activation functions allow neural networks to learn a wide array of mappings that cannot be approximated by a series of linear transformations.

The sigmoid activation function is also known as logistic function and is perhaps the best-known activation function for NNs. Given a scalar input x, the activation is given by

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.20}$$

In the context of NNs, x corresponds to input to the neuron, after taking the inner product with the weight vector and adding the bias term. Thus, the activation ranges from 0 (for large negative values of x) to 1 (for large positive values of x). For neurons with sigmoid activation, gradient descent can be used to find the weights and biases of the network since the activation function is differentiable with respect to its input. The derivative is bell-shaped and has its maximum for $x = 0.5$. Note that the sigmoid is a continuous approximation of an all-or-nothing response.

One issue of the sigmoid activation function is that the function becomes saturated for very small and very large input values with its derivative being close to zero in both cases. This can be problematic during gradient descent. Since NNs are essentially a composition of functions, the gradient is computed during the backward pass of the backpropagation algorithm and requires the repeated application of the chain rule from calculus, meaning that the gradient is repeatedly multiplied by the derivative of the sigmoid. Since the derivative of the sigmoid function lies between 0 and 0.25, repeating this multiplication for several layers will decrease the magnitude of the gradient to very small values which effectively blocks the gradient from reaching neurons in earlier layers of the NN, preventing the network from learning in these layers. This phenomenon is known as the vanishing gradient problem and is especially problematic when many neurons produce saturated responses (with the gradient being close to zero) or when the network has a large number of layers. When used as an output layer, sigmoid activation is commonly used for binary classification. When classification is performed with more than two categories, the softmax function is the most common output layer. The softmax is defined as

$$\text{Softmax}\left(x_i\right) = \frac{\exp\left(x_i\right)}{\sum_j \exp\left(x_j\right)}$$

where x_j denotes the input to the softmax layer corresponding to class j. The softmax function is a generalization of the sigmoid function to multiple classes and they are identical in the binary classification case. In this thesis, we will focus our discussion on the sigmoid function, but an extension to softmax output is straightforward.

A more recent activation function is the rectified linear unit (ReLU). It is a piecewise linear activation function that is defined as follows:

$$g(x) = \max(0, x) \tag{2.21}$$

It has been pointed out in [18] that the ReLU activation function has three advantages compared to the sigmoid activation: The computation is faster since there is no exponential in the equation, it mitigates the vanishing gradient problem because the derivative can be equal to one, and is associated with a sparse response because neurons with negative inputs to the activation function produce a zero response. To illustrate this, for an untrained NN with randomly initialized weights and biases of zero, we would expect half of the neurons with ReLU activation to have a zero response for random inputs. This is clearly not the case for sigmoid activation. It has been argued that such sparsity is more biologically plausible and comes with a number of advantages compared to a more dense encoding, such as robustness to small changes due to a disentanglement of information [18]. As a result, ReLU activations have largely replaced sigmoid activations as the most popular choice for NNs.

2.6 Regularization

A core strength of deep NNs is their high capacity to learn complex input-output mappings from data. Yet, this property also makes deep NNs prone to overfitting to the training set, especially if the available training data is relatively small. Overfitting is a phenomenon in machine learning in which the model learns to solve a particular problem on the training data, but then fails to generalize to the test data. One way of viewing this situation is that the NN has failed to separate the signal from the noise during training, resulting in errors on different datasets that do not share the idiosyncrasies of the training data. In this context, regularization refers to techniques which are designed to mitigate the problem of overfitting in neural networks, generally by placing some constraints on the model during the training procedure.

2.6.1 Weight-decay

Weight-decay has been known for a while in the ML community to reduce overfitting in NNs [37]. It refers to the practice of adding a term to the loss function which penalizes large parameter values. The motivation for this is that, in order to prevent overfitting, we want to prevent the model output from being dominated by a small group of weights with a large magnitude. The two most commonly used weight decay procedures for NNs and other models like regression are L1 and L2 [3] regularization. If the original loss of the model is the NLL, the new loss functions for L1 and L2 are given by

$$J_{L1}(\boldsymbol{\theta}, \mathcal{D}) = -\sum_{i=1}^{N} \log P\left(y^{(i)} | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) + \lambda \sum_{i=1}^{M} |w_i| \tag{2.22}$$

$$J_{L2}(\boldsymbol{\theta}, \mathcal{D}) = -\sum_{i=1}^{N} \log P\left(y^{(i)} | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) + \lambda \sum_{i=1}^{M} {w_i}^2 \tag{2.23}$$

Here, w_i refers to the ith weight out of the overall M weights of the NN and λ is a parameter that controls the contribution of the weight penalty to the overall loss. As is common practice, the bias values are not penalized. Due to the squaring of the weights, L2 regularization penalizes large weight values more than L1, but L1 regularization has a stronger emphasis on inducing real sparsity in the weights.

It is worth mentioning that training with L1 regularization is equivalent to finding the MAP estimate for a prior distribution over the weights in which each weight is i.i.d drawn from a Laplace distribution. Similarly, assuming that the weights are drawn i.i.d from a Gaussian distribution is an instance of L2 regularization. In both cases, λ depends on the parameter that defines the spread of the distribution, with smaller spread being associated with stronger regularization.

2.6.2 Batch Normalization

Batch normalization [32] is a relatively recent technique which was originally developed for facilitating the training procedure, but was also found to have some regularizing effect on the networks as well. Batch normalization was devised to mitigate the adverse effects of what the authors refer to as *internal covariate shift*. This phenomenon refers to the property that the input distribution for a layer changes as training progresses and the previous layers in the network adapt their parameters to the training data. An additional source of variation occurs when training is performed using minibatches. In both cases, small differences in the input distribution are bound to be magnified when passing through the network from layer to layer. The key contribution of batch normalization is to normalize the outputs of each layer such that the mean and variance are mostly fixed between iterations. As the name suggest, inputs in each mini-batch are standardized by subtracting the mini-batch mean and dividing by the variance. In addition, two trainable parameters, β and γ are added to each layer. They shift and scale the output respectively, allowing the layer to learn the optimal mean and variance that should be passed to the next layer by gradient descent. The detailed procedure is presented in Algorithm 1.

These adaptations allow for the use of higher learning rates and make the success of the training procedure less dependent on the parameter initialization. The regularizing effect of batch normalization is due to the fact that the neural network is unlikely to encounter the exact same input twice during different training iterations because the normalization will change as the input is drawn together with different mini-batches across the iterations. This will prevent the network from memorizing any particular input.

[3] L2 is also known as Tikhonov regularization or ridge regression in statistics.

Algorithm 1 Batch Normalization, adapted from [32]

```
    Require: Mini-batch $\mathcal{B} = \{x_{1...m}\}$
    Require: Parameters to be learned: $\gamma$, $\beta$
    Returns: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$                              ▷ mini-batch mean
$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$                  ▷ mini-batch variance
$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$        ▷ normalize. $\epsilon$ is a small positive constant
                                                  to avoid division by 0.
$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$                  ▷ scale and shift
```

2.6.3 Dropout

Whereas L1 and L2 regularization originated in the context of regression, *dropout* is a regularization technique that was specifically devised for NNs. In dropout, a Bernoulli distribution, characterized by dropout probability p, is placed over all nodes of the network. For each forward pass through the network during training, nodes that draw the value 0 are removed from the network, along with all incoming and outgoing connections [60]. When the network is used for prediction after training, no dropout takes place and activations in the network are multiplied by $1 - p$ in order to compensate for the increase in the overall activation that is due to the use of all neurons.

This prevents overfitting to the training data by forcing the network to learn robust representations instead of relying on a selected few nodes and connections to perform most of the computation. Given the vast combinatorial space of possible networks created by random dropout, dropout training forces the network to learn more robust representations instead of relying on complex co-adaptations with other nodes in the network, which would be much more susceptible to overfitting to the noise in the training set [29].

It has further been argued that dropout improves model performance by essentially training an ensemble of different networks due to the difference in their structure that is caused by random dropout. Compared to other ensemble approaches which explicitly combine predictions of different models in some way, dropout can be viewed as training an ensemble of networks and implicitly combining these by using all nodes and rescaling the activations.

Due to these benefits, networks using dropout achieved state-of-the-art performance on many computer vision tasks (e.g. [36]). However, it has also been pointed out that the use of dropout may require more training time because parameter updates become noisy since a different model is trained in each iteration [60], in some cases doubling the number of iterations that are needed for the model to converge [36].

Other stochastic regularization techniques that resemble dropout have been studied for training NNs. An example is multiplicative Gaussian noise, in which the activation of hidden units is multiplied by a random variable $r' \sim \mathcal{N}(1, \sigma^2)$ during the forward pass through the network. Here, the standard deviation of the distribution

σ is a hyperparameter which determines the strength of the regularization, similarly to the dropout probability p described above. Although [60] found this approach to be equally effective, standard dropout was more widely adopted, perhaps due to the computational efficiency and theoretical simplicity of the approach.

2.7 Convolutional Neural Networks

The convolution is an operation that originates in signal processing. Since this thesis focuses on NNs in the computer vision domain, we will focus on explaining two-dimensional convolutions here. Given two scalar-valued functions $x(m, n)$ (the image) and $h(m, n)$ (the kernel or filter), the two-dimensional convolution is defined as:

$$y[i, j] = (x * h)[i, j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x[i - m, i - n]h[m, n] \tag{2.24}$$

Thus, convolving two functions x and h results in a new function that is the result of flipping the kernel and moving it across the image, advancing pixel-by-pixel. At each step, we compute the product of the overlapping pixels and then take the sum of these products. This sum then corresponds to the output for the pixel on which the kernel is currently centered. In this context, we can think of the kernel as a feature detector such as the well-known Sobel operator for edge detection [59], and the output of the convolution as a map that indicates how strongly the feature is present across the input.

Since the indices in Equation 2.24 range from negative infinity to infinity, we assume that any pixels that fall outside of the boundaries of the input matrix and the kernel are zero. Padding the input with zeros also allows us to compute the result of the convolution for pixels near the outside of the input matrix because kernels that are larger than one pixel would partly cross the boundary of the input matrix to evaluate these pixels. This method is known as zero padding. Another common method for obtaining an output matrix that has the same size as the input matrix is to assign pixels outside the boundary the value of the closest pixel that lies within the boundary. Alternatively, one may decide to perform a *valid* convolution, in which we only keep the output for locations where the kernel does not cross the boundary of the input matrix. However, this would reduce the size of the output.

Convolutional neural networks (ConvNets) are a special type of NN which have become very popular for solving computer vision tasks. The main difference between convolutional layers and the densely connected layers of standard MLPs is that nodes in convolutional layers have only local receptivity, meaning they receive input from a spatially connected part of the input to the layer. Additionally, nodes are part of a channel and all nodes within a channel share the same weights but are connected to different parts of the input. In this manner, each node within a channel can be viewed as applying the same feature detector (or filter) to a different part of the input, thereby effectively performing a convolution on the input. Since all neurons within a channel share the same weights, convolutional layers need far fewer parameters compared to a fully connected layer when both have to perform the task of finding a particular feature at different locations in the image.

In practice, images are treated as three-dimensional inputs, where the depth refers to the number of channels. For instance, an image with RGB encoding would have three channels in the input layer: red, green, and blue. The convolution kernels are therefore three dimensional, covering a part of the width and height of the input, but all of its depth. To illustrate, assuming that the image is square with a height and width of 50 pixels, the input shape would then be $50 \times 50 \times 3$. If we feed this input into a convolutional layer that performs a convolution with 10 distinct 5×5 filters, each filter would have dimensions $5 \times 5 \times 3$ and the output of the layer would have a shape of $50 \times 50 \times 10$, assuming that the input is padded to preserve the width and height. Thus, each node in the output volume effectively computes the value of the convolution of one of the 10 kernels at a particular location in the width/height grid of the input volume, covering the whole depth since the filters are three dimensional. Since the output of the layer comprises 10 channels, the filters in the next convolutional layers would also have a depth of 10. Since we can think of filters as feature detectors, the number of channels in a given layer corresponds to the number of features that we would like to be able to detect from the input.

Stride

The stride of a convolutional layer determines the vertical and horizontal space between the center of the receptive field of a neuron and the center the of the receptive fields of its neighbors. Above, a stride of one is assumed, meaning that the area to which the filter is applied for one neuron has an offset of one compared to adjacent nodes. If we choose a stride of two, there will be an offset of two in vertical and horizontal direction. Thus, the stride determines how densely the filter is applied to the input. Note that a stride larger than one will reduce the height and width of the output since fewer nodes are needed to cover the input.

Pooling

It is often desirable to reduce the dimensionality of the output of a convolutional layer, as features become more abstract in later layers of the network and the precise location plays a less important role. Next to strided convolutions (implying a stride that is larger than one), pooling layers are the main method for achieving this. Spatial pooling (also referred to as subsampling) is a method which reduces dimensionality while retaining important information. Max pooling achieves this by dividing each depth-slice of the input into regions of a certain shape, say 2×2, and subsequently reducing each region to a single value by retaining the maximum value. The output of a max pooling layer then has the same depth as its input, but reduced height and width. Average pooling follows the same procedure, but retains the average instead of the maximum of each region.

2.7.1 Dropout in Convolutional Neural Networks

Standard dropout works by temporarily deleting random neurons in the network, including all incoming and outgoing weights, in the forward pass that takes place during training. As we have seen above, the nodes within convolutional layers are spatially arranged such that the feature map is equal to the output of a convolution of a filter with the input to the layer. It is common that the stride of the convolution is sufficiently small relative to the size of the filter such that adjacent neurons have overlapping receptive fields with respect to the input. If this is the case, the feature maps of the convolutional layer become spatially correlated as the filter is applied to overlapping parts of the input. This, taken together with the fact that image pixels are often spatially correlated to begin with, motivated Tompson et al. to point out that standard dropout may not be the best regularization for convolutional layers because the regularizing effect of dropout may be weak since the nodes adjacent to dropped nodes are likely to preserve the spatially correlated information [62]. This amounts to a reduction in learning rate, while providing relatively little regularization. Instead, they propose *SpatialDropout* which randomly drops whole feature maps, thereby setting the activations of all neurons within a channel to zero. As with standard dropout, the probability of dropping whole feature maps is characterized by a Bernoulli distribution with dropout probability p. This approach provides stronger regularization compared to standard dropout because there is no a priori reason for filters within a layer to be redundant when no dropout or other regularization methods are used.

2.8 Modern Convolutional Neural Networks

2.8.1 AlexNet

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is one of the best-known competitions in computer vision, featuring annual object detection and image classification challenges [8]. Perhaps the most notable breakthrough in the recent history of neural networks occurred in 2012, when a ConvNet called AlexNet [36] outperformed all other competitors in the ImageNet classification competition by more than 10% in the top-5 error[4] rate. The 2012 challenge featured a training set of 1.2 million images and 1000 object categories, which makes the competition relevant for real-life applications. AlexNet is one of the first *deep* neural networks, featuring five convolutional layers, three max-pooling layers, and three fully connected layers. ReLUs were used as activation functions and dropout was applied in order to regularize the fully connected layers. AlexNet can be considered a milestone in the history of NNs because it was one for the first very large networks (having 60 million parameters and 650,000 neurons). In addition, it used sophisticated graphical-processing unit (GPU) optimization to be able to train the large network in order to fully take advantage of the size of the ImageNet dataset.

2.8.2 VGG

Another breakthrough in image classification was achieved by the VGG-16 model, which managed to halve the top-5 error of AlexNet on ImageNet [58]. The key difference in terms of architecture is that VGG-16 used 3×3 convolutional kernels, compared to AlexNet, which also uses kernels of 11×11 and 5×5 in the early layers. This decision was based on the insight that the larger receptive field of larger kernels can be imitated by stacking several layers with smaller kernels. For example, two layers with 3×3 kernels have an effective receptive field of 5×5. This makes it possible to incorporate more activation functions (since there are more layers) and also significantly decreases the number of parameters that would be required for a larger kernel with an equivalent receptive field. In total, VGG-16 has 138 million parameters, but significantly larger depth than comparable networks at the time due to the use of smaller kernels.

2.8.3 GoogLeNet

While it may seem at this point that the increase in performance may just be a result of ever-increasing size, GoogLeNet (Inception V1), the 2014 winner of the ILSVCR

[4]top-5 error rate denotes the proportion of images in the test set for which the ground truth label is not among the five labels that the model predicts to be most likely

challenge showed that this is not the case, achieving a 6.7% top-5 error with merely 6.8 million parameters [61]. This impressive performance was achieved by introducing the *inception* module, in which 1×1 convolution, 3×3 convolution, 5×5 convolution and 3×3 max pooling are applied to the same input, stacking the result to provide the input to the next layer. This configuration, albeit rather complex, enables the network to choose the most suitable path itself. The key change that allowed for this to work and also changed the total number of parameters was the extensive use of 1×1 convolutions, which substantially reduce the depth of the input to the layers throughout the network.

2.8.4 Residual Networks

In the sequence of AlexNet, VGG, and GoogLeNet, increasing depth goes hand in hand with improved performance. Residual networks (ResNets) are specifically devised to make it possible to train very deep models [26]. The deepest model presented is a 152-layer ConvNet with 60.2 million parameters and has a top-5 error of only 4.49 % on the ImageNet validation set. He et al. observed that as ConvNets become deeper than a certain threshold, performance begins to deteriorate. Since this is also observed in the training set, this is a problem of optimization rather than generalization. The authors attribute this to the fact that it is already quite challenging for deep neural networks to learn the identity function as more layers are added. As a remedy, they use residual blocks, which contain shortcut connections to later layers besides regular convolutional layers which perform feature extraction. At the end of each residual block, the input to the block is added to the output of the convolutional path by means of the shortcut connection. This design makes it straightforward for the network to learn the identity function because the model can just drive the weights for the convolutional path to zero and rely on the parameter-free skip connections. The name *residual* derives from the interpretation that the convolutional path in the residual block now only has to learn the residual between the input and the desired output. For the three larger ResNets presented in [27], special kinds of residual blocks are used. In these *bottleneck* blocks, the convolutional path consists of a 1×1 convolution reducing the input depth from 256 to 64, a 3×3 convolution, and a final 1×1 convolution which maps the output back to 256 to be compatible with the original input to the block that is provided by the shortcut connection. A graphical depiction of a residual block and bottleneck block is shown in Figure 2.1.

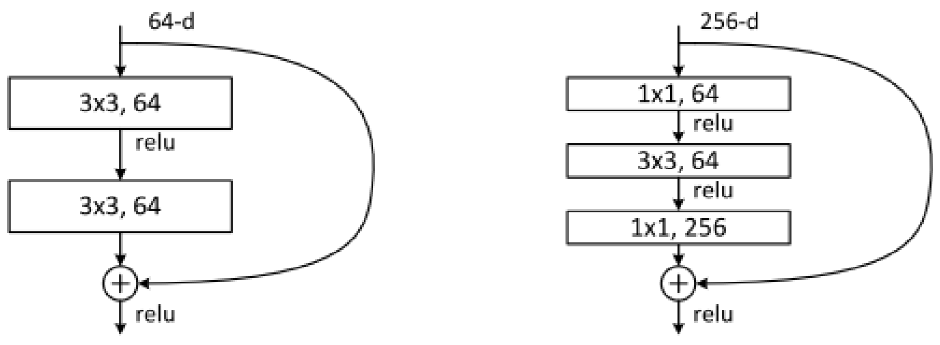


FIGURE 2.1: Adapted from [27]. Left: a residual block used for the ResNet-34 network. Right: a bottleneck block that is used for the larger ResNet-50/101/152. The + symbol stands for element-wise addition

.

Chapter 3

Uncertainty Estimation for Neural Networks

3.1 Types of Uncertainty

When evaluating the uncertainty of a modelling process, a distinction can be made between aleatoric and epistemic uncertainty [9]. In general terms, epistemic uncertainty is caused by insufficient knowledge about which model and which parameters best model the data, while aleatoric uncertainty results from noise in the data itself [33]. In contrast to aleatoric uncertainty, epistemic uncertainty can be reduced by collecting additional data.

In [33], a further distinction is made between aleatoric uncertainty that is homoscedastic and aleatoric variance which is hetereoscedastic. Homoscedastic aleatoric uncertainty refers to uncertainty that is constant for different inputs, for instance due to fundamental limitations in camera resolution. In contrast, heteroscedastic aleatoric uncertainty differs between different input patterns. Examples in the computer vision domain include darkness, fog, sensor noise, and occluded objects in images.

3.2 Uncertainty in Classification

Neural networks usually perform classification by producing an output vector of values between 0 and 1, one entry for each class and summing to 1 if the classification is mutually exclusive and exhaustive, i.e. each input belongs to exactly one ground truth class. This is frequently implemented by a softmax activation function. For multi-label classification, multiple categories can be assigned to the same input pattern. The entries can be assigned independently from each other and do not have to sum to one. This can be achieved by a layer with independent sigmoid activation functions for example. It is common practice for practitioners to interpret these values as actual predicted probabilities rather than mere confidence scores on a scale from 0 to 1. This relies on the assumption that the predictions are well calibrated. A model is well-calibrated if the predicted probabilities coincide with the actual accuracy of the model. For instance, we would expect a perfectly calibrated model to have an empirical accuracy of 70% in expectation for input patterns which the networks classifies with a softmax or sigmoid score of 0.7. Note that model accuracy and calibration are independent, since a model can have a very low accuracy while still being well-calibrated and vice-versa.

When the model is trained by using the NLL as a loss function, the loss is only minimized when the network learns to match predicted probabilities with its empirical accuracy [12]. In other words, the loss is reduced when the model improves its calibration during training. This makes the NLL a *proper scoring rule* [38][19]. Since the likelihood is also part of the loss that is used for MAP estimation (refer to Equation 2.15), good calibration is also rewarded in MAP estimation [1].

Thus, assuming that the optimization procedure found a good local minimum, we have some ground for treating the predicted values of the model as probabilities. Since the network learns to provide its own uncertainty estimate while classifying input patterns, we can categorize this as an example of aleatoric uncertainty.

However, it has been pointed out that calibration of modern NNs may have deteriorated compared to earlier, more simple models [24]. Using different models, the authors varied network depth, the number of filters per layer, and whether or not batch normalization was used. All three aspects were found to have an effect on calibration, as well as accuracy. Increasing model capacity (i.e. using deeper layers and more filters per layer) up to a certain point increases accuracy, while letting calibration deteriorate. This may partly be due to overfitting the predictive distribution to the training set [24], especially when the accuracy of the network is already very high. Using batch-normalization had a similar effect, increasing accuracy while making the model less calibrated. They further examine different approaches for manually calibrating classifiers after training and find that *temperature scaling*, an extension of Platt scaling [53], is a simple and effective way of improving the predictive distribution, provided that sufficient data is available for validation.

3.3 Uncertainty in Regression

In contrast to the classification case, standard neural network regression predicts the target value(s) without giving an indication of confidence in the outputs. The most commonly used cost functions are the MSE and MAE which, in contrast to the NLL used in classification, do not have a probabilistic interpretation because they only concern themselves with the size of the error and not with the probability of observing errors of different magnitudes. To obtain an additional confidence score for regression, we can make the additional assumption that the prediction error is characterized by a particular distribution. For example, for a regression task with one-dimensional targets and normally distributed residuals, the likelihood function is:

$$p_{model}\left(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right) = p(y^{(i)}|\hat{y}^{(i)}) \tag{3.1}$$

$$= \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y^{(i)}-\hat{y}^{(i)})^2}{2\sigma^2}\right) \tag{3.2}$$

where the mean is the predicted value $\hat{y}^{(i)}$ and σ^2 is the variance of the Gaussian distribution. By splitting the output for the regression from one to two neurons, we can train our model to predict the variance of this distribution in addition to the regression estimate itself by training the model to maximize the likelihood function. As for the classification case, in practice, it is common to minimize the NLL instead:

[1] An exception to this rule occurs when an improvement in calibration comes at the cost of a lower prior probability of the model parameters, which would amount to a penalty in the loss function.

$$-\log p(y^{(i)}|\hat{y}^{(i)}) = \frac{(y^{(i)}-\hat{y}^{(i)})^2}{2\sigma^2} - \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) \tag{3.3}$$

$$= \frac{(y^{(i)}-\hat{y}^{(i)})^2}{2\sigma^2} + \frac{1}{2}\log(\sigma^2) + \log(\sqrt{2\pi}) \tag{3.4}$$

To minimize the NLL, we can ignore the last term since it is constant with respect to the model parameters, which gives us the following loss:

$$\mathcal{J}(\boldsymbol{\theta}|y^{(i)}, \mathbf{x}^{(i)}) = \frac{(y^{(i)}-\hat{y}^{(i)})^2}{2\sigma^2} + \frac{1}{2}\log(\sigma^2) \tag{3.5}$$

We can see that the first term on the right-hand side of the equation is just the squared error, divided by $2\sigma^2$. Thus, the model can reduce the loss that it incurs for any input pattern by predicting a large variance for the Gaussian distribution of the residual that we assumed. At the same time, predicting a larger variance is penalized by the second term on the right-hand side of the equation. This is an instance of learned loss attenuation [33], in which the network learns to reduce training loss by recognizing difficult examples.

A similar result, but involving the absolute error, can be obtained by assuming a Laplace noise distribution. Recall that the probability density function of the Laplace distribution is defined as follows:

$$p(x|\mu, b) = \frac{1}{2b}\exp\left(-\frac{|x-\mu|}{b}\right) \tag{3.6}$$

Similarly to the predicted standard deviation σ of the Gaussian noise distribution, b defines the spread of the Laplace distribution, with the variance given by $\sigma^2 = 2b^2$. The likelihood for regression is then

$$p_{model}\left(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right) = \frac{1}{2b}\exp\left(-\frac{|y^{(i)}-\hat{y}^{(i)}|}{b}\right) \tag{3.7}$$

where $\hat{y}^{(i)}$ and b are predicted by the model. The negative NLL, which we wish to minimize, is then given by:

$$\mathcal{J}(\boldsymbol{\theta}|y^{(i)}, \mathbf{x}^{(i)}) = \frac{|y^{(i)}-\hat{y}^{(i)}|}{b} + log(2b) \tag{3.8}$$

As for the Gaussian distribution, the model can reduce the loss for the regression error for any input by predicting a large variance, which is also penalized in the loss.

3.4 Bayesian Neural Networks

As alluded to in subsection 2.3.3, training Bayesian NNs amounts to approximating the posterior distribution of the weights given the data. Given an approximation of the posterior distribution, we can sample different networks in order to obtain a distribution of outputs. The variance of the predictive distribution then provides us with a measure of epistemic uncertainty because we do not rely on a single set of weights for the prediction. Note that a single model may be highly confident, indicating small aleatoric uncertainty, while a Bayesian prediction using multiple draws from the posterior may still indicate high epistemic uncertainty due to large variation in the output of the different networks.

The posterior distribution of modern NNs cannot be solved analytically and is generally difficult to sample from [22].

3.4.1 Variational Inference

Variational inference addresses this problem by approximating the posterior $p(\boldsymbol{\theta}|\mathcal{D})$ with a more tractable distribution $q(\boldsymbol{\theta}|\boldsymbol{\beta})$ and optimizing the variational parameters $\boldsymbol{\beta}$ to resemble the posterior as much as possible [2]. The optimization is based on the Kullback-Leibler (KL) divergence, which in this context is given by

$$\mathrm{KL}(q\|p) = \mathrm{E}_q\left[\log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}|\mathcal{D})}\right] \tag{3.9}$$

The KL divergence is also referred to as relative entropy and measures the extent to which one distribution differs from a target distribution. Note that by Equation 3.9, the KL divergence is not symmetric and is equal to zero when the distributions are identical. Following [2], it is shown that Jensen's inequality can be used to find a lower bound:

$$\begin{aligned}\log p(\boldsymbol{x}') &= \log \int_{\boldsymbol{\theta}} p(\boldsymbol{x}', \boldsymbol{\theta}) \\ &\geq \mathrm{E}_q[\log p(\boldsymbol{x}', \boldsymbol{\theta})] - \mathrm{E}_q[\log q(\boldsymbol{\theta})]\end{aligned} \tag{3.10}$$

where $\boldsymbol{x}'$ is meant to denote a data point $\boldsymbol{x}$, including the ground truth target variable(s). This is known as the evidence lower bound (ELBO), or variational free energy. They further show that the KL divergence can be rewritten to

$$\mathrm{KL}(q(\boldsymbol{\theta})\|p(\boldsymbol{\theta}|\boldsymbol{x}')) = -\left(\mathrm{E}_q[\log p(\boldsymbol{\theta}, \boldsymbol{x}')] - \mathrm{E}_q[\log q(\boldsymbol{\theta})]\right) + \log p(\boldsymbol{x}'), \tag{3.11}$$

which is just the negative ELBO plus a constant term that does not depend on q. Thus, optimizing the variational parameters to maximize the ELBO is equivalent to minimizing the KL divergence. The choice of the approximating distribution has then to be performed in such a way that the computation of the expectations in Equation 3.10 becomes feasible.

To implement this approach in a Bayesian NN, Graves [22] proposed to replace the true posterior distribution with a variational distribution which, while still intractable to integrate, would allow for efficient sampling. These samples can then be used for stochastic optimization, like gradient descent. In this way, using a Gaussian prior distribution results in a diagonal Gaussian posterior for the NN parameters and was found to perform well on a phoneme recognition task, showing little overfitting.

Blundell et al. [3] built on this work by making gradient updates unbiased and extending the method to non-Gaussian prior distributions, calling their approach *Bayes by Backprob*. In their approach, the variational optimization fits a Gaussian mean and standard deviation for each weight in the network, thereby doubling the number of parameters compared to a standard NN. The name of their method is due to the fact that the standard gradient that is used in backpropagation reappears as part of the gradient updates for the means and standard deviations in the variational posterior distribution. Using a Bayesian NN with a mixture of two Gaussian distributions of different scales as the prior and a Gaussian as the posterior distribution, they were able to achieve performance on the MNIST digit recognition task on par with NNs that were regularized by dropout, while also providing Bayesian uncertainty estimates.

The previous approaches made the training of Bayesian NNs viable for real-world applications. However, the fact that the number of parameters is effectively doubled (for a Gaussian prior with independent means and standard deviations for each weight) makes it computationally expensive to extend these methods to larger models, which may be needed for more challenging tasks.

3.5 Dropout as Bayesian Approximation

Motivated by this fact, Gal and Ghahramani showed in [14] that using dropout during training and prediction approximates a deep Gaussian process (GP) [7], a probabilistic model with inherent Bayesian uncertainty estimates. As a machine learning model, GPs are characterized mainly by their covariance function, which measures the similarity between data points. To make predictions for new data points, the prediction is extrapolated by referring to known target values in the data set and weighing their contribution to the prediction by evaluating the covariance function between known data points and the new data point. The resulting prediction is a mean and variance, the sufficient statistics for a one-dimensional Gaussian distribution.

The deep GP is a hierarchical extension of standard GPs, in a very similar fashion to the way in which a MLP is an extension of perceptron-like single layer NNs. That is to say, each layer in a deep GP is a Gaussian process latent variable model (GP-LVM), taking as input the output of the previous layer and passing its output to the next layer in the hierarchy.

Below, a summary of the derivation is presented. For the (extensive) full derivation please refer to the appendix of [14]. Consider the following covariance function:

$$\mathbf{K}(\mathbf{x}_1, \mathbf{x}_2) = \int p(\mathbf{w}) p(b) \sigma \left(\mathbf{w}^T \mathbf{x}_1 + b \right) \sigma \left(\mathbf{w}^T \mathbf{x}_2 + b \right) \mathrm{d}\mathbf{w} \mathrm{d}b \tag{3.12}$$

for any two data points $\mathbf{x}_1$ and $\mathbf{x}_2$, and given the parameters of a covariance function $\mathbf{w}$ and b and an element-wise non-linearity $\sigma(\cdot)$, the output of a deep GP consisting of multiple layers characterized by this covariance function can be approximated by an NN in which the hidden units explicitly represent the activation term $\sigma\left(\mathbf{w}^T \mathbf{x} + b\right)$ above. This is partly based on an earlier result in which it was shown that a hidden layer with an infinite number of neurons converges to a GP [52] when a distribution is set over the weights. Above, some prior distribution $p(b)$ is assumed, in addition to a prior distribution $p(\mathbf{w}) = \mathcal{N}\left(\mathbf{w}; 0, l^{-2} I\right)$. In this context, l is called the prior length-scale.

The interpretation of dropout training as variational inference will now be further explained. For the L layers of the deep GP, let $\mathbf{W}_i \in \mathbb{R}^{K_i \times K_{i-1}}$ be a matrix for layer i and $\boldsymbol{\omega} = \{\mathbf{W}_i\}_{i=1}^{L}$ the set of all these matrices. In addition, let $\mathbf{m}_i \in \mathbb{R}^{K_i}$. Integrating over the parameters $\boldsymbol{\omega}$, Gal and Ghahramani parameterize the posterior predictive probability as

$$\begin{aligned} p(\mathbf{y}|\mathbf{x},\mathcal{D}) &= \int p(\mathbf{y}|\mathbf{x},\boldsymbol{\omega})p(\boldsymbol{\omega}|\mathcal{D})\mathrm{d}\boldsymbol{\omega} \\ p(\mathbf{y}|\mathbf{x},\boldsymbol{\omega}) &= \mathcal{N}\left(\mathbf{y};\widehat{\mathbf{y}}(\mathbf{x},\boldsymbol{\omega}),\tau^{-1}\mathbf{I}\right) \\ \widehat{\mathbf{y}}\left(\mathbf{x},\boldsymbol{\omega}\right) &= \sqrt{\frac{1}{K_L}}\mathbf{W}_L\sigma\left(\ldots\sqrt{\frac{1}{K_1}}\mathbf{W}_2\sigma\left(\mathbf{W}_1\mathbf{x}+\mathbf{m}_1\right)\ldots\right) \end{aligned} \tag{3.13}$$

Above, τ refers to the model precision, which represents a prior belief about how much noise is present in the observations. As discussed earlier, it is not computationally feasible to compute the posterior distribution $p(\boldsymbol{\omega}|\mathcal{D})$ for the parameters of the covariance function. Instead, the authors use a variational distribution $q(\boldsymbol{\omega})$ to approximate the posterior.

$$\begin{aligned} &\mathbf{W}_i = \mathbf{M}_i \cdot \operatorname{diag}\left(\left[\mathbf{z}_{i,j}\right]_{j=1}^{K_i}\right) \\ &\mathbf{z}_{i,j} \sim \text{ Bernoulli } (p_i)\,, \;\; i=1,...,L, \;\; j=1,...,K_{i-1} \end{aligned} \tag{3.14}$$

Thus, the matrix $\mathbf{W}_i$ is equal to a matrix $\mathbf{M}_i$, but each column j is set to zero when the corresponding random variable $\mathbf{z}_{i,j}$, drawn from a Bernoulli distribution with probability p_i, is zero. The parameters of the variational distribution are therefore $\{\mathbf{M}_i\}_{i=1}^{L}$. To optimize these parameters, we again choose to minimize the negative log ELBO instead of working directly with $\mathrm{KL}(q(\boldsymbol{\omega})\|p(\boldsymbol{\omega}|\mathcal{D}))$:

$$\mathcal{J}_{ELBO} = -\int q(\boldsymbol{\omega})\log p(\mathbf{Y}|\mathbf{X},\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} + \mathrm{KL}(q(\boldsymbol{\omega})\|p(\boldsymbol{\omega})) \tag{3.15}$$

$$p(\mathbf{Y}|\mathbf{X},\boldsymbol{\omega}) = -\sum_{n=1}^{N}\int q(\boldsymbol{\omega})\log p\left(\mathbf{y}_n|\mathbf{x}_n,\boldsymbol{\omega}\right)\mathrm{d}\boldsymbol{\omega} \tag{3.16}$$

In the appendix of [14], it is shown that the KL divergence between the variational posterior and the prior distribution can be approximated as follows:

$$\mathrm{KL}(q(\boldsymbol{\omega})\|p(\boldsymbol{\omega})) \approx \sum_{i=1}^{L}\frac{p_i l^2}{2N}\left\|\mathbf{M}_i\right\|_2^2 + \frac{l^2}{2N}\left\|\mathbf{m}_i\right\|_2^2 \tag{3.17}$$

with l as the prior length-scale mentioned before. To optimize the objective in equation 3.15 stochastically, the integral in 3.16 is approximated with Monte Carlo (MC) integration by drawing a different set of variational parameters from the variational distribution for each prediction during training: $\widehat{\omega}_n \sim q(\boldsymbol{\omega})$. Finally, the loss is scaled by $1/\tau N$, which yields the final objective:

$$\mathcal{J}_{\mathrm{GP-MC}} \propto \frac{1}{N}\sum_{n=1}^{N}\frac{-\log p\left(\mathbf{y}_n|\mathbf{x}_n,\widehat{\boldsymbol{\omega}}_n\right)}{\tau} + \sum_{i=1}^{L}\left(\frac{p_i l^2}{2\tau N}\left\|\mathbf{M}_i\right\|_2^2 + \frac{l^2}{2\tau N}\left\|\mathbf{m}_i\right\|_2^2\right) \tag{3.18}$$

This loss function can be optimized with any gradient optimization technique, such as SGD. Note that the loss is equivalent to the training objective that is used when training NNs using weight-decay and dropout:

$$\mathcal{J}_{\text{dropout}} := \frac{1}{N}\sum_{i=1}^{N}E\left(\mathbf{y}_i,\widehat{\mathbf{y}}_i\right) + \lambda\sum_{i=1}^{L}\left(\left\|\mathbf{W}_i\right\|_2^2 + \left\|\mathbf{b}_i\right\|_2^2\right) \tag{3.19}$$

Above, $E\left(\mathbf{y}_i, \widehat{\mathbf{y}}_i\right)$ refers to the standard NLL loss as a function of the ground truth $\mathbf{y}_i$ and the prediction $\widehat{\mathbf{y}}_i$. In summary, the well-established training procedure of using gradient optimization with dropout for training NNs amounts to finding the parameters $\{\mathbf{M}_i\}_{i=1}^{L}$ for the variational distribution which approximates the posterior.

After training, predictions can be made by sampling from the variational distribution. In practice, that means performing several stochastic forward passes through the network using dropout. This is referred to as Monte-Carlo (MC) dropout, to distinguish it from standard dropout, in which dropout is performed only during training and all units are used during test time.

3.6 Domain-Shift Detection

Recall from the introduction that covariate shift is a special case of dataset shift in which only the input distribution changes. Supervised learning generally assumes that any data that a model might encounter is i.i.d. with the training set. It would therefore be useful to be able to determine when such a distributional shift takes place because generalization to a new input distribution is not guaranteed. One benchmark task for detecting covariate shift involves training a classifier on the MNIST dataset [39] and subsequently applying it to notMNSIT [5], a similar dataset which consists of symbols of different fonts, instead of the handwritten digits in MNIST. On these image classification datasets, it was shown that out-of-distribution (OOD) detection could be performed using the predictive uncertainty derived from ensembles [38] and Bayesian NNs [47] since uncertainty was larger for the OOD data points, on average.

Similarly, dropout uncertainty has been found to increase when an NN classifier, trained on medical images of eyes to detect diabetic retinopathy, is applied to a dataset with different semantics (ImageNet) [41]. The authors also investigated whether an autoencoder, attached to the penultimate layer of the NN, could be used for the same purpose. Autoencoders are NNs which are designed to learn an identity mapping, that is to reconstruct the inputs [21]. These networks are usually subject to some constraint which makes perfect reconstruction difficult in order to force the autoencoder to learn meaningful representations of the data.

Conceptually, autoencoders consist of an encoder $\boldsymbol{c} = e(\boldsymbol{x})$, which transforms the input to a code $\boldsymbol{c}$. The code is a representation in a hidden layer that is then passed to the decoder $\hat{\boldsymbol{x}} = d(\boldsymbol{c})$, which attempts to reconstruct the input from the code. In NNs, the encoder and decoder may consist of multiple layers each. One of the most common constraints that is placed on the autoencoder is that the dimensionality of the code is much smaller compared to that of the input, which is often implemented in a shape that resembles an hour glass, with layers that decrease in width for the encoder and for the decoder layers that increase in width in order to recover the original input dimensions. Autoencoders can be applied to detect covariate shift based on the assumption that due to the constraints that are placed on the hidden representation, the autoencoder is forced to learn transformations that are specific to the distribution of the training set. In other words, the fact that the autoencoder might overfit to the training set may be used to detect unfamiliar data points, which should be associated with a higher reconstruction error.

There are other approaches which are specially designed to address this problem. For neural network classifiers, the authors of [28] suggest the softmax probability of the predicted class as a simple baseline for detecting OOD examples. In addition, they propose an *abnormality module* which combines the softmax predictions with a reconstruction error from a reconstruction module, specifically attached to the deep features of the network for this purpose. Another approach modifies the loss function such that the Kullback-Leibler divergence between the output distribution for out-of-data examples generated by generative adversarial networks (GANs) and the uniform distribution is minimized [40]. A third approach learns an extra confidence output which, during training, determines the extent to which the softmax output is interpolated with the ground truth to reduce the loss. This amounts to the network being able to pay for hints when it is uncertain and is somewhat similar to the learned loss attenuation proposed in [33].

Chapter 4

Object Detection

Viola and Jones presented one of the earliest successful real-time object detection algorithms in 2001 [64]. It was primarily designed for face detection in images. The algorithm adopts a sliding window approach by partitioning the image into patches (which is repeated for different scales) and then performing binary classification on each patch to determine whether a face is present in that region. This classification relies on a large set of Haar-like features that were designed manually based on prior knowledge. The final detector is then trained by applying the AdaBoost [11] boosting algorithm to iteratively select features based on the reduction in error rate that is associated with the inclusion of the feature. To achieve real-time performance, a cascade of classifiers is used in which later classifiers are only used for prediction if no previous classifier rejected the patch by predicting the absence of a face.

With advances in NN research and state-of-the art performance in image classification, performing object detection with NNs was a natural next step. The main obstacle that was needed to overcome was the dual nature of the task, since both localization and multi-class classification are required, early NN approaches were essentially hybrid approaches that relied on computer vision algorithms to obtain regions of interest (RoIs), which are likely to contain an object.

4.1 Performance Metrics for Object Detection

When evaluating object detection models, we need a way to match predictions with the ground truth boxes. The criterion that is used for this is the intersection over union (IoU), also known as the Jaccard similarity coefficient. The IoU is defined as the size of the intersection of two sets divided by the size of the union:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{4.1}$$

When applied to images, A and B are the areas of the predicted bounding box and the ground truth box and the IoU is thus obtained by dividing the area of the overlap of the two boxes by the total area covered by the two boxes (counting the overlap only once). The most commonly used threshold for counting a predicted box as a match with respect to a ground truth box is IoU = 0.5, though other values can be used and sometimes evaluation is repeated for different IoU thresholds.

Precision is a performance metric that originates form information retrieval and can be defined as follows

$$\text{precision} = \frac{\#TP}{\#TP + \#FP}$$

Here, $\#TP$ refers to the number of true positives, i.e. target objects that were correctly classified and $\#FP$ refers to the number of false positives, i.e. instances where the model predicts an object but no object is present at that location.
Similarly, recall is defined as

$$\text{recall} = \frac{\#TP}{\#TP + \#FN}$$

$\#FN$ denotes the number of false negatives, that is to say the number of objects in the data which were not detected by the model.

Since neural networks for classification generally output a confidence score between 0 and 1 for each class, we have to choose a threshold value above which we predict the presence of the class. Since the number of TP, FP, and FN depends on how high or low we set the classification threshold, each threshold value is associated with a pair of precision and recall values. We can obtain the precision-recall curve assigning precision to the y-axis and recall to the x-axis and plot the precision/recall points for the whole range of classification thresholds. The average precision (AP) for a class is then defined as the area under the precision-recall curve:

$$AP = \int_0^1 p(r)dr$$

As the name suggests, it can be interpreted as the the average precision for the different recall values. AP is the main metric that is used in the literature to evaluate object detection performance for a given class. The mean average precision (mAP) for all classes can be calculated to evaluate the performance across multiple classes.

4.2 Object Detection Neural Networks

4.2.1 R-CNN

In 2014, Girshek et al. proposed the Region-based Convolutional Network (R-CNN) [17], which relies on the Selective Search algorithm [63] to propose RoIs, which are then resized to a fixed size and passed as input to the AlexNet ConvNet. Here, the ConvNet is only used for extracting (4096) features from the RoIs. To perform the final detection, an SVM is trained for each class to perform binary classification (whether or not the class is present). In addition, the features are used to train a linear regression model in order to more accurately predict the bounding box for the object. The combined approach was able to achieve a leap in performance, an increase in mAP of more than 30% on the PASCAL VOC 2012 data set [10], a frequently used benchmark, compared to the next best model at the time. However, since the ConvNet has to process a large number of RoIs per image, the model is much too slow for real-time prediction.

4.2.2 Fast R-CNN

The fast R-CNN [16] is an adaptation of the R-CNN architecture, in which the whole image is passed to the ConvNet (instead of selected RoIs) and the Selective Search is performed on the feature maps of the ConvNet. After identifying RoIs in the feature map, these regions are resized to a fixed size by a RoI pooling layer. Next, the resized features are passed to a fully connected layer which performs additional processing before passing the output to a softmax layer for multi-class classification and a regression layer that refines the shape and location of the bounding box. These changes made training the fast R-CNN about 10 times faster compared to the R-CNN and predictions about 20 times faster. While the bottleneck in terms of performance for the R-CNN was that the ConvNet had to evaluate all RoIs separately, for the fast R-CNN, the bottleneck is the region proposal itself.

4.2.3 Faster R-CNN

To adress this bottleneck, a region-proposal network (RPN) was devised, which obviates the costly Selective Search for RoIs [55]. First, the RPN performs a 3×3 convolution, followed by a 1×1 convolution to extract a feature vector at each location in the feature map of the last convolutional layer of the ConvNet. To propose the actual RoIs, each of the feature vectors is passed to (the same) two fully connected layers. The first fully connected layer uses regression to predict the offset with respect to k predefined anchors, which are default bounding boxes of different size and aspect ratios. The second fully connected layer uses the same features and predicts the *objectness* score corresponding to each of the anchors. In order to avoid multiple detections of the same object when there are multiple anchors which receive high objectness scores, non-maximum suppression (NMS) is performed by only retaining the boxes associated with the highest scores for boxes whose overlap is larger than a given IoU threshold. The final classification after NMS takes place by first performing RoI pooling on the last convolutional layer of the ConvNet and then passing the features to a softmax layer, which predicts a probability score for each class (background also being counted as a class). Since the RPN uses the same convolutional features as the final classifier, the region proposal is virtually for free during runtime. One drawback of this architecture is that the RPN network has to be trained first and then the proposed RoIs can be used to train the fast R-CNN part of the model. This is repeated iteratively since both parts of the model share the same ConvNet.

4.2.4 SSD

The Single Shot Multibox detector (SSD, [46]) takes a similar approach to the faster R-CNN, but simplifies the architecture and training procedure. One of the main changes is that feature maps from several parts of the ConvNet are taken for predicting object location and class. Four new convolutional layers are added to the two feature maps from the VGG-16 model [58]. This has the purpose of accounting for the large variation in the size of objects because the height and width of the feature maps become increasingly smaller, meaning that nodes in the deeper layers have a much larger receptive field with respect to the input image. The architecture is shown in Figure 4.1. The early layers are therefore used for predicting small objects and the later layers for predicting large objects.

To be more precise, the network attaches a 3×3 convolutional layer with $(c + 4) \cdot k$ filters to each of these six feature maps, where c is the number of classes and k is the number of anchor boxes per cell in the feature map. The network therefore predicts at each location in those feature maps and for each of the k anchors the corresponding softmax class probabilities in addition to four regression offsets with respect to the anchor box. By using several feature maps of different sizes and receptive fields with anchors of different sizes, the SSD discretizes the output space and performs classification for each box before applying NMS. The lack of an explicit region proposal step makes the network easier to train compared to the approaches discussed above and when first published, the network achieved state of the art performance while being faster than most existing models. A difficulty that arises from the architecture described above is that during training, the large majority of boxes will be classified as negatives (i.e. as background), which is an issue because these negative examples dominate the loss during training. To deal with this particular problem, hard negative mining (HNM) is used for training the SSD. HNM selects only a subset of these negatives during training by only taking those with the highest classification loss and ensuring that at most three times as many negative boxes as positive boxes are used for training per image.

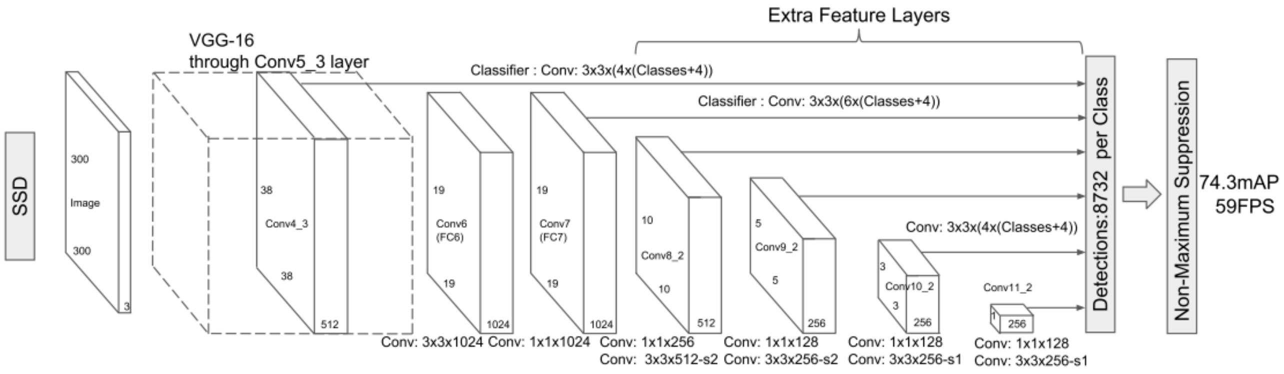


FIGURE 4.1: Adapted from [46]. Single Shot Multibox Detector (SSD). Additional layers of decreasing height and width are appended to the VGG-16 ConvNet. Predictions are made by sliding detection filters across the feature maps at multiple scales with respect to the input image.

4.3 RetinaNet

RetinaNet [44] is a relatively recent state of the art object-detection network that is similar to SSD, but is improved by a new loss function and richer convolutional features at multiple scales with respect to the input image. These changes increased performance from an AP of 31.2 and 34.9 for the faster R-CNN+++ [26] and an improved version of SSD [13] respectively to an AP of 39.1 for RetinaNet on the challenging MS COCO [45] data set. This is a significant improvement, considering that all three models used the same ResNet-101 backbone as feature extractor.

4.3.1 Focal Loss

The training of the network is characterized by the focal loss function, which addresses the problem that the ground truth for the vast majority of output boxes corresponds to background which can dominate the overall loss for an image. This is the same problem that SSD tries to solve by hard negative mining. To detect objects, for each box, RetinaNet performs a binary classification for each class using a sigmoid activation function which predicts whether the given class is present in the box or not. Defining p_t as the probability that is assigned to the ground truth,

$$p_{\mathrm{t}} = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{otherwise} \end{cases} \tag{4.2}$$

the focal loss for the classification of a box is given by

$$\mathrm{FL}\left(p_{\mathrm{t}}\right) = -\left(1-p_{\mathrm{t}}\right)^{\gamma} \log\left(p_{\mathrm{t}}\right) \tag{4.3}$$

This is just the cross entropy with an additional factor $(1-p_{\mathrm{t}})^{\gamma}$, with hyperparameter $\gamma \in [0,\infty)$. Adding this factor has the effect of significantly reducing the contribution of easy predictions (high p_t), especially for high values of γ. The authors found that $\gamma = 2$ worked best in their experiments. By using focal loss, the network can be trained using all outputs and without extra steps such as HNM.

For the bounding box regression, a smooth L1 loss is used. This loss was also used by SSD and fast(er) R-CNN and is very similar to the standard L1 loss, but smoothed around zero.

4.3.2 Feature Pyramid Network

The second defining feature of RetinaNet is that it uses a feature pyramid network (FPN) [43] which can be obtained from the majority of recent ConvNets, such as residual networks [26]. Extending standard ConvNets to FPNs is motivated by the fact that ConvNets often have difficulties detecting small objects [46]. In contrast, FPNs are particularly designed to facilitate object detection at multiple scales. This is motivated by the success of traditional multi-scale approaches in computer vision, such as the scale-invariant feature transform (SIFT) [48] and was indeed found to provide a significant increase in performance when applied to object detection.

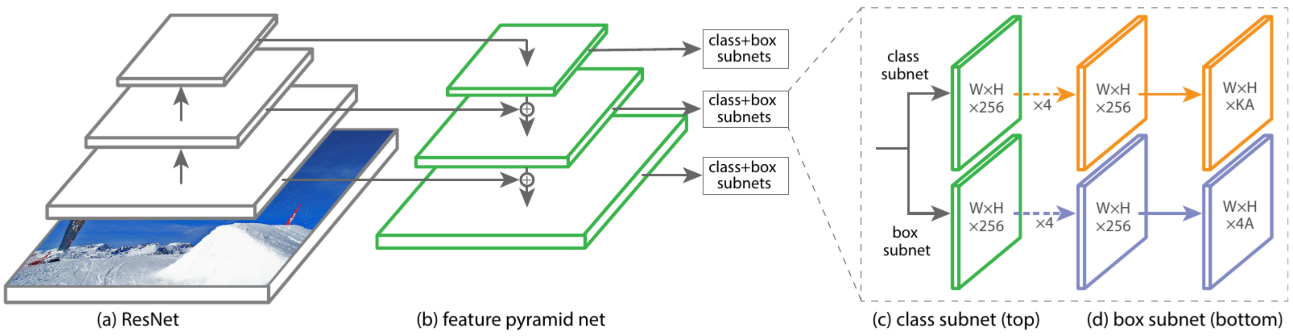


FIGURE 4.2: RetinaNet. Figure adapted from [44]. A ResNet is used as feature extractor (a). Pyramid layers are connected in a feedforward manner by upsampling from small to large feature maps. In addition, they receive lateral connections from the corresponding layers in the backbone. This feature pyramid network (b) creates features at multiple scales. Weight-sharing classification subnetworks and box regression subnetworks are attached to each pyramid layer.

The main idea of FPNs is to append new layers that are increasing in height and width to the base network, which usually *decreases* in height and width as the network becomes deeper. This architecture results in an hour-glass shape, such as shown in Figure 4.2. Each new pyramid layer receives input from the previously added smaller pyramid layer in addition to the deepest convolutional layer of the base-network with the same dimensions as the new layer. To be more precise, the relevant layers from the base network undergo a 1×1 convolution to reduce dimensionality. These feature maps are then added element-wise to the layers which were upsampled by a factor of 2 (using nearest neighbor upsampling) from the previous pyramid layer. Finally, a 3×3 convolution is used to generate the final pyramid layers and reduce alias that may occur due to the upsampling. In this manner, it is possible to obtain abstract semantic information by upsampling from small to large layers while also retaining precise spatial information by lateral connections from the original layers of the base network.

4.3.3 Classification and Regression Subnetworks

In order to perform object detection, two fully convolutional subnetworks are attached to each level of the feature pyramid, with the weights of these subnetworks being shared between pyramid layers. See Figure 4.2 for a visual reference. The first subnetwork is the classification subnetwork and consists of 4 convolutions with 256 channels per layer, followed by a convolutional layer with output depth $c \times k$, where c again denotes the number of classes and k denotes the number of anchor boxes per location in the spatial grid that is given by the width and height of the convolutional feature map. By default, 9 anchors (3 sizes with 3 aspect ratios) are used. Since the network is fully convolutional, each grid cell corresponds to a region of the original image. This means that RetinaNet provides detection outputs for each spatial location as defined by the width $\times$ height grids of the feature maps of the feature-pyramid layers and for each anchor bounding box in each grid cell. The last layer uses sigmoid activation to predict the presence of each class, with scores close to 1 indicating complete confidence in the presence of an instance of a particular object category and values close to 0 indicating strong confidence that no object of that class is present in the region corresponding to the anchor.

The regression subnetwork has a very similar architecture and is used to adjust the anchor bounding boxes to better fit objects that are detected by regressing the offsets of the predicted box with respect to the anchor. Like the classification subnetwork, the regression subnet is attached to each layer of the FPN and consists of 4 convolutions with 256 channels per layer. The last layer is a convolutional layer with output depth $4 \times k$. Thus, the output of the regression subnet represents the predicted offset of the x_1, y_1, x_2, and y_2 coordinates of the sides of the bounding box with respect to the anchor per spatial location. In practice, the offsets are predicted relative to the width and height of the anchor boxes.

In conclusion, the classification and regression subnetworks output sigmoid classification scores (for each class) and class-agnostic bounding box regression to predict the offset relative to the height/width of the anchor dimensions. To obtain the final predictions, non-maximum suppression (NMS) is performed, in which only the predictions with the highest classification scores are selected when there are several detections with overlapping bounding boxes, as measured by the intersection over union (IoU) criterion.

Chapter 5

Experiments

In this chapter, we will discuss the experiments that we performed in order to answer the research questions that were introduced in section 1.4. To recapitulate, these can be summarized as follows: In the NN object-detection setting,

1. how does dropout affect the calibration of the model?
2. does the dropout uncertainty provide value when used as a measure that is used for decision referral?
3. can dropout uncertainty be used to detect covariate shift?
4. can we model bounding box regression uncertainty by training the model to predict the regression variance? How does this compare to the dropout uncertainty for regression?

5.1 Dataset

The Berkeley Deep Drive dataset [65] is a recent data set that contains object detection annotations for 100,000 images taken from independent clips of traffic situations. The videos are captured by cameras that are mounted on cars that drive to a number of cities and highways in the United States. Images have a resolution of 1280x720 pixels. The object categories that are annotated are: Bus, Light, Sign, Person, Bike, Truck, Motor, Car, Train, Rider. Here, Rider refers to the person who is riding a bicycle or motorbike. Bike and Motor refer to the bicycles and motorbikes respectively. As can be expected from real-world scenes, there is a substantial class-imbalance in the data set, as illustrated in Table 5.1.

The dataset is quite challenging since many images were taken from behind the windshield of a car and are therefore blurry or subject to flare on the windshield or occlusion due to rain drops.

For the main part of the experiment, we removed the annotations for traffic lights and traffic signs in order to focus on vehicles and people as participants of traffic. The data set also contains annotations with respect to the time when the videos were captured (Dawn/Dusk, Daytime, and Night). We split all of the Night images from the rest of the images in order to obtain an additional test set that could be used to evaluate how this distributional shift affects performance when testing on a split that differs from the training distribution. This split is henceforth referred to as DAY (Daytime, Dawn/Dusk) and NIGHT respectively.

Examples from both sets are shown in Figure 5.1. This leaves us with a training set of 41,972 images, a validation set of 6,071 images and 31,957 night images.

bike	bus	car	motor	person	rider	train	truck
395	638	33,059	188	5,012	272	4	1,672

TABLE 5.1: Number of instances per class in the DAY test set

Since no annotations are available for the official test set (consisting of the remaining 20,000 images), we split the validation set into a smaller validation set of 3,036 images and test set of 3,035 images.

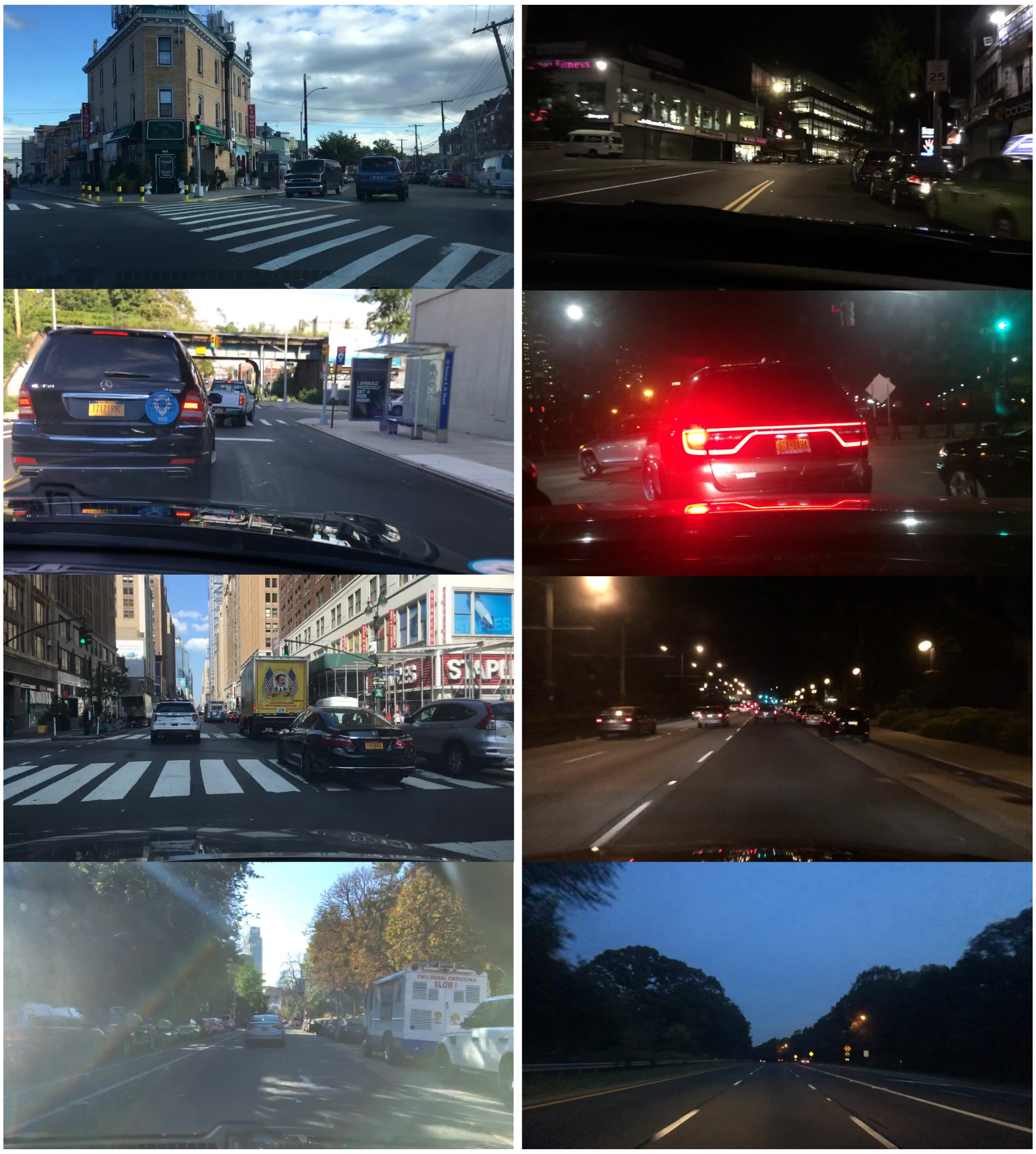


FIGURE 5.1: Example images sampled from the DAY (left column) and NIGHT (right column) dataset.

5.2 Detection Networks

All experiments were performed using a Keras [6] implementation[1] of RetinaNet [44] and we used weights that were pretrained on MSCOCO 2017 [45], achieving a mAP of 0.537 for IoU = 0.5 and a combined mAP of 0.350 for the IoU threshold ranging from 0.50 to 0.95. This network uses ResNet50 [26] as the backbone to build the feature pyramid network. To answer our research questions, we modified the network in different ways and fine-tuned all versions on the BDD dataset, as described below. The dropout rates for the networks were obtained by favoring large dropout rates in order to obtain meaningful variation in the output, while also ensuring that any decrease in detection performance remains as small as possible.

5.2.1 Standard network (noDrop)

This is the standard network without modifications.

5.2.2 Backbone Dropout 10 (BB-10)

To perform dropout in the ResNet50 backbone, we added a dropout layer with a dropout probability of 0.1 before the last convolutional layer of each bottleneck block. Dropout uncertainty can then be obtained by performing stochastic forward passes using the training model to obtain a set of outputs for each anchor in the image. The stochastic predictions can then be combined by computing the mean and variance of the output before performing NMS.

5.2.3 Backbone Dropout 20 (BB-20)

As for BB-10, but doubling the dropout rate ($p = 0.2$). We include this model because in preliminary experiments dropout in the backbone was associated with a decrease in performance, whereas the BB-10 actually improved detection performance on the main experiment a little as we will see below. For a more thorough comparison, we therefore also include a model that performs similarly to the other dropout variants below.

5.2.4 Backbone 2D Dropout 10 (BB2D-10)

Similarly to BB-10, but using SpatialDropout [62] which randomly drops convolutional filters, thereby setting whole depth-slices (channels) of the output volume to zero. The BB2D-10 configuration uses a SpatialDropout probability of 0.1 in the backbone.

5.2.5 Subnet Dropout 20 (SN-20)

For dropout in the regression and classification subnetworks, we add a standard dropout layer with dropout rate of 0.2 before each of the four convolutional layers that precede the last layer. Performing dropout in the subnetworks means that we induce noise in the semantically richer feature maps, compared to performing dropout in the backbone. This results from the finding that convolutional filters become more abstract as the network becomes deeper [66].

[1]https://github.com/fizyr/keras-retinanet

	car	person	truck	rider	motor	bike	train	bus	mAP
SN-20	0.700	0.606	0.587	0.467	0.463	0.511	0.00	0.578	0.489
SN2D-20	0.694	0.595	0.582	0.449	0.464	0.490	0.00	0.568	0.480
BB-10	0.702	0.611	0.589	0.477	0.472	0.517	0.00	0.577	**0.493**
BB-20	0.698	0.597	0.572	0.448	0.465	0.495	0.00	0.568	0.480
BB2D-10	0.700	0.606	0.576	0.454	0.469	0.496	0.00	0.575	0.484
Laplace	0.691	0.595	0.569	0.446	0.446	0.494	0.00	0.536	0.472
Gauss	0.688	0.591	0.558	0.430	0.439	0.495	0.00	0.534	0.467
NoDrop	0.702	0.614	0.588	0.464	0.453	0.498	0.00	0.583	0.488

TABLE 5.2: Detection performance (AP) of different networks on the DAY test set (at IoU = 0.5).

5.2.6 Subnet 2D Dropout 20 (SN2D-20)

In the SN2D-20 configuration, dropout is implemented as for SN-20, but using SpatialDropout instead of standard dropout.

5.2.7 Laplace Regression (Laplace)

In the Laplace configuration, the network is trained to predict (for each of the four sides) the spread parameter b (cf. Equation 3.7) of the Laplace distribution which we assume underlies the error of the bounding box regression. This assumption is based on the fact that RetinaNet is trained with a smooth version of the absolute regression error and an absolute error term is also obtained by assuming a Laplace noise distribution, as discussed in section 3.3. In practice, we predict the log of b for numerical stability and to ensure that $b > 0$. The regression error is then just the NLL.

5.2.8 Gaussian Regression (Gauss)

We do the same but using a Gaussian observation noise model. That is, we predict the log variance $\log \sigma^2$ from Equation 3.2, assuming that the regression error follows a normal distribution. We again use the NLL as the regression loss.

5.3 Training and Optimization

All networks were trained for 80,000 iterations with a batch size of 2, using the ADAM optimizer with a learning rate of 5×10^{-5}, which worked well for all reported models. Learning rates were decreased by half if the validation loss did not decrease for two subsequent epochs of 4000 iterations. If the validation loss did not decrease further after three subsequent epochs, training was stopped. Data augmentation was performed by randomly mirroring training images around the vertical axis.

5.4 Model Performance

As suggested in [14], we use MC-dropout to obtain predictions for the dropout models at test time. Specifically, we perform 30 forward passes[2], from which we compute

[2]in preliminary experiments we found no improvement when performing more than 30 forward passes.

	car	person	truck	rider	motor	bike	train	bus	mAP
SN-10	0.638	0.563	0.495	0.356	0.205	0.441	0.000	0.436	0.392
SN2D-20	0.633	0.549	0.503	0.327	0.273	0.473	0.000	0.463	0.403
BB-10	0.646	0.554	0.527	0.385	0.231	0.525	0.000	0.416	0.410
BB-20	0.633	0.541	0.515	0.377	0.251	0.467	0.000	0.452	0.404
BB2D-10	0.643	0.565	0.506	0.431	0.207	0.489	0.000	0.453	**0.412**
NoDrop	0.644	0.558	0.499	0.378	0.222	0.501	0.000	0.460	0.408

TABLE 5.3: Detection performance (AP) of different networks on the NIGHT test set (at IoU = 0.5).

the average sigmoid and regression predictions, as well as the variance around these predictions. The detection performance of the networks that were trained on the DAY training set and tested on the DAY test set is shown in Table 5.2.

A baseline for comparison is provided in [65], in which a faster R-CNN [55] is trained on all daylight images (excluding night and dawn/dusk). They report an mAP of 0.3515 for these classes. Note that we used a different split by also including dusk/dawn images in the DAY split. The relatively large difference in performance can be attributed to the fact that RetinaNet is a more recent object detection network. In addition, we fine-tuned the weights of a pretrained model, which often improves performance. The network fails to detect any trains in the test set (which is also true for the baseline reported by [65]). This is likely due to the fact that there are only 171 trains in the 100,000 images and only 4 instances in our test set that was obtained by random assignment of the images.

The performance that results from testing these models on the NIGHT split is presented in Table 5.3. The networks generalize a bit better to the NIGHT test set with a drop of about 0.08 mAP compared to [65], who found a drop of 0.093 mAP. This is probably due to the fact that our DAY split also includes dusk/dawn images, which are more similar to the night images. We deliberately decided to keep dusk/dawn images in the DAY set in order to ensure that the NIGHT split is not completely different compared to the DAY split. We consider this scenario of encountering different, but still related data distributions to be more likely to occur in practice.

5.5 Association between Dropout and Sigmoid

Considering that the derivative of the sigmoid function is largest when the input to the sigmoid is zero and close to zero when the input is a large positive or negative number, we can expect the dropout variance to be largest for predicted probabilities of 0.5 (corresponding to an input value of zero) because any change in the input due to dropout has a much larger effect on the output compared to the saturated parts of the function. This relationship is indeed found when we assign sigmoid scores to bins and measure the mean and variance of the dropout variance of the sigmoid score. This is illustrated in Figure 5.2, in which we see that the dropout variance in the subnetworks is larger compared to dropout in the backbone, for instance $\bar{\sigma}(p = 0.5) \approx 0.077$ for SN-20 and $\bar{\sigma}(p = 0.5) \approx 0.061$ for BB-20. We surmise that this is due to the fact that feature maps become more abstract and semantically richer for deep layers [66]. This would explain why dropout in layers that are very close to the NN output would induce larger variation in the output.

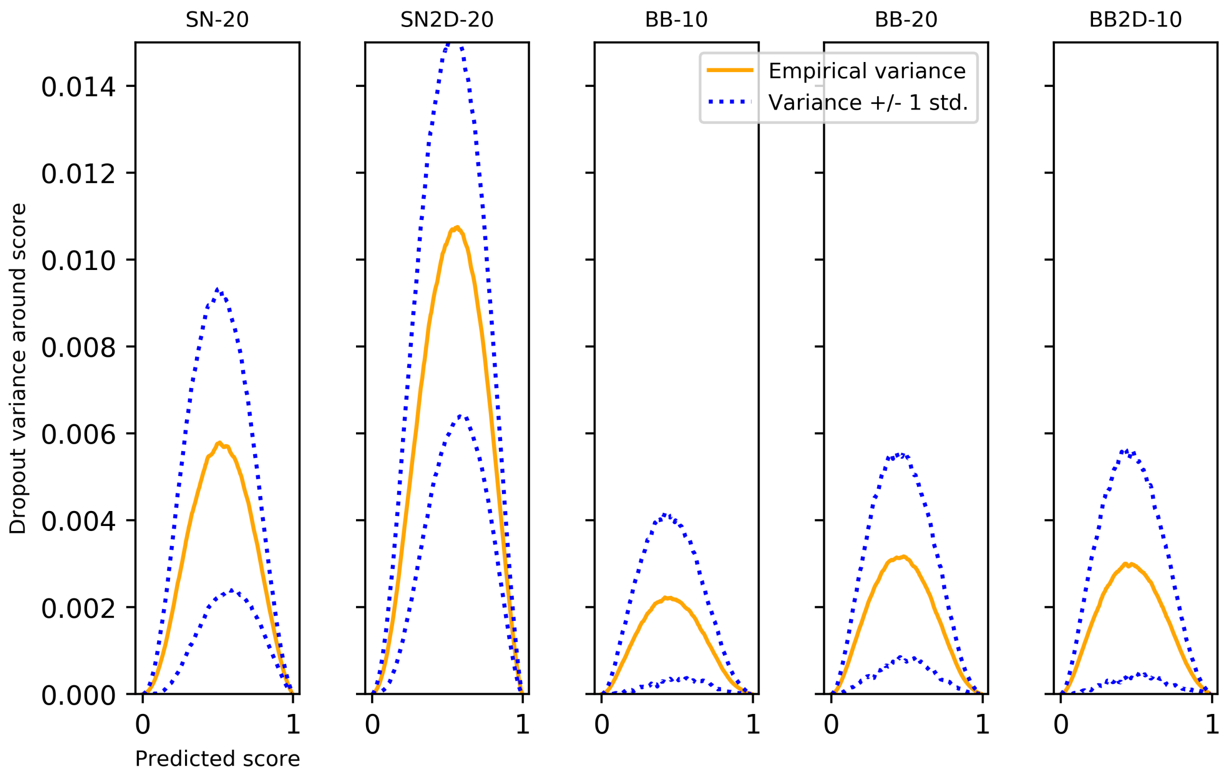


FIGURE 5.2: Empirical association between sigmoid score and corresponding dropout variance. Blue lines represent ± 1 standard deviation in dropout variance.

5.6 Calibration

The calibration of the different models can be obtained by binning predictions into confidence ranges and computing for each bin the empirical performance on the test set. We chose to divide the sigmoid range of $[0, 1]$ into 60 bins. This is presented in Figure 5.3, which shows that model predictions are overconfident for detection values below 0.4 and underconfident for values that are larger than 0.4. Looking closely, we can see that the calibration appears to be slightly worse for the BB variants in the range of $[0.4, 0.8]$, but the difference is rather small.

5.7 Decision Referral

In order to determine whether dropout uncertainty can be used to rank which predictions are most likely to be wrong, we apply dropout uncertainty in a decision referral paradigm. In decision referral, difficult predictions are referred to an *oracle*, such as a human expert. The performance of the model is then computed on the reduced data set of predictions that have not been referred to the oracle. Specifically, we use the sigmoid score of the standard RetinaNet for the predicted class as a baseline. For this model, sigmoid scores are considered to be more uncertain the closer they are to 0.5. For dropout uncertainty, we take the dropout variance in the sigmoid score for the predicted class as the referral criterion. We can then investigate how performance increases if we remove more and more potential detections from

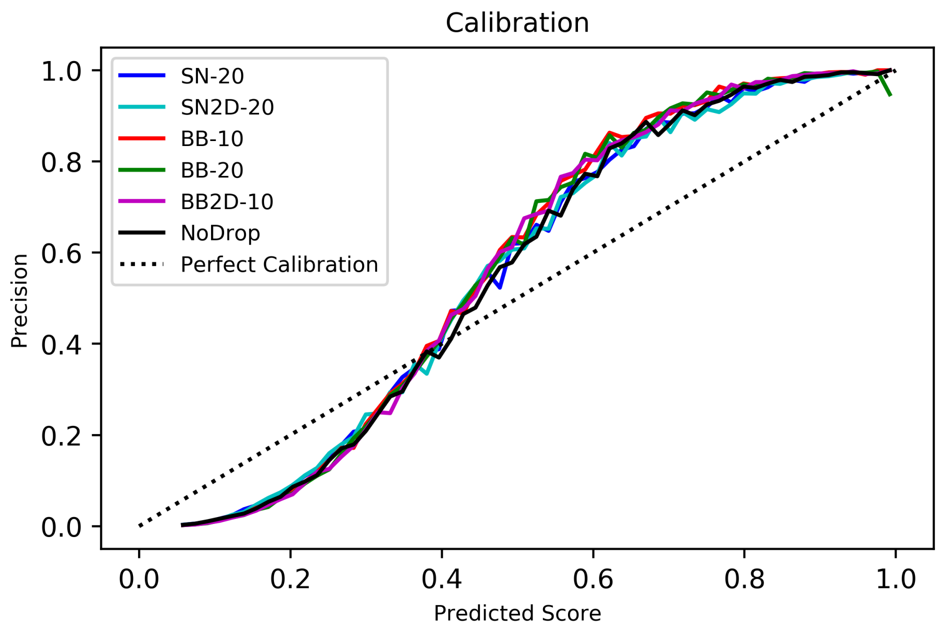


FIGURE 5.3: Calibration on the test set. The predicted score is the sigmoid confidence for the predicted class. Calibration was obtained after thresholding scores at a confidence of 0.05 and performing NMS.

the data set, starting with the most uncertain examples and removing increasingly more certain instances. In practice, we consider only referral for thresholded predictions (with a confidence of 0.1) after the NMS step because otherwise the number of predictions becomes too large to be meaningful in this paradigm. This is due to the preponderance of low-confidence predictions. Since we are interested in the change in precision and recall as data is referred, we must choose ≤ 0.5 as a fixed score threshold for predicting the presence of an object.

Performing decision referral on the DAY test set is visualized in Figure 5.4. On average, there are 43 potential detections per image after thresholding and NMS. Thus, at a referral rate of 50% (upper right end of the lines), about 21 detections are deferred per image. This means that the performance for 5% and 10% referral is of particular interest, as it is the most feasible in practice. From Figure 5.4 we can infer that the sigmoid uncertainty of the noDrop model is the most suitable referral criterion. Referring 5% or 10% of detections is associated with a much larger performance increase compared to dropout variance of the sigmoid score of the predicted class. The performance increase after 50% referral is also larger. While the improvement in precision is largest for dropout variance in the SN-20 and SN2D-20 models after 20% referral, this comes at the cost of an initial decrease in recall. Note that this phenomenon can be explained by the fact that for these networks, the association between dropout variance and sigmoid scores is shifted towards higher sigmoid scores (see Figure 5.2).

These results indicate that the standard sigmoid uncertainty is a better statistic for ranking detections by their probability of being false compared to the dropout variance of the sigmoid score. This stands in contrast to the finding of [41], in which they found that that dropout uncertainty worked well for decision referral in an image classification task, while softmax output was not able to achieve consistent improvements in accuracy. This difference in results may be due to the fact that during

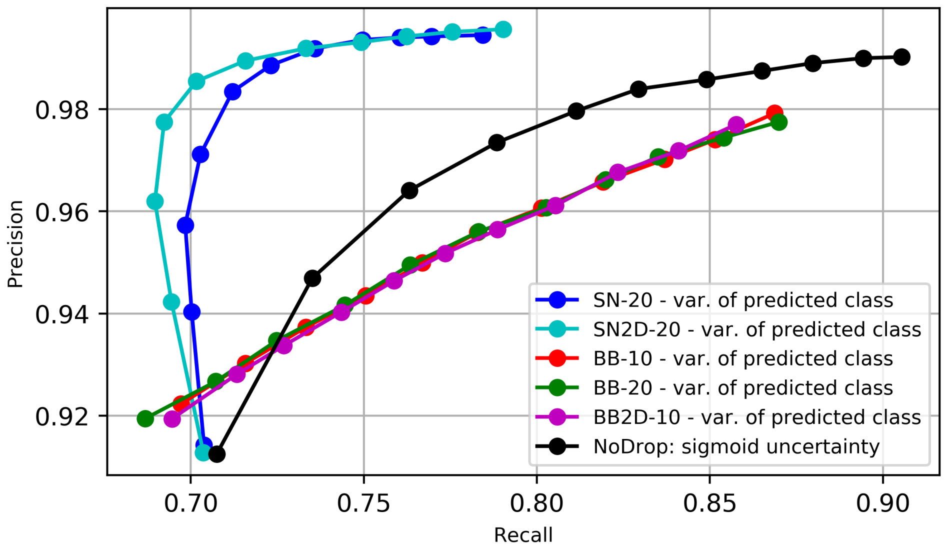


FIGURE 5.4: Decision referral. Performance on the complete test set is represented by the lower left of the lines. Each subsequent point represents the improved performance after referring an additional 5% of the detections, ranked by the uncertainty metric. At the upper right of the lines, 50% of potential detections have been referred.

training for object detection, the network encounters a significant preponderance of background patches, which may force the network to only predict objects when it is justified to do so. This may render the sigmoid uncertainty more meaningful for referral compared to the softmax uncertainty for classification.

We also looked at other ways of measuring dropout uncertainty for this task. Instead of looking only at the variance in the score for the predicted class, one could also look at the average dropout variance or standard deviation across all classes. As can be seen in Figure A.1 in Appendix A, these measures did not beat the sigmoid baseline.

5.8 Domain-shift Detection

In the next step, we set out to investigate whether dropout uncertainty can be used for detecting that distributional shift took place, meaning that the network is tested on data that differs from the training distribution. For this purpose, we applied the dropout networks, which were trained on the DAY split to the NIGHT split and compared the distribution of scores for the predicted class, the dropout variance in these scores, and an adjusted dropout variance in which we correct for the association between sigmoid scores and the dropout variance around these scores, as illustrated in Figure 5.2. In particular, we divided the sigmoid range of [0, 1] into bins and measured the dropout variance for these bins. Subsequently, to adjust a given dropout variance, we subtract the mean of its bin and divide by the standard deviation. The distribution of scores of the SN-20 network for the Day test set and 1,000 randomly selected images from the NIGHT set is presented in Figure 5.5.

To compare the distributions, we compute the two-sample Kolmogorov-Smirnov (KS) statistic, which is equal to the maximum vertical distance between the cumulative empirical distribution functions of the two sampled distributions. Large values

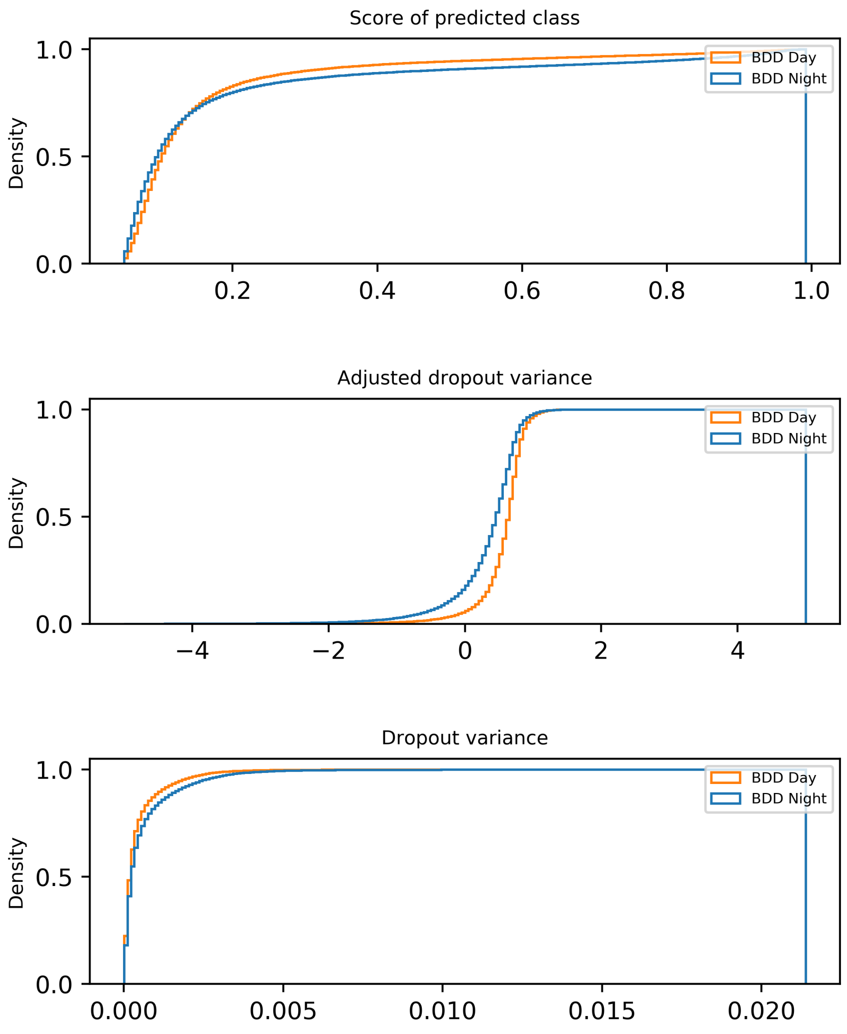


FIGURE 5.5: Cumulative density histogram of sigmoid scores, adjusted dropout variance, and dropout variance for the DAY vs. NIGHT split for **SN-20**. All measures are taken with respect to the predicted class.

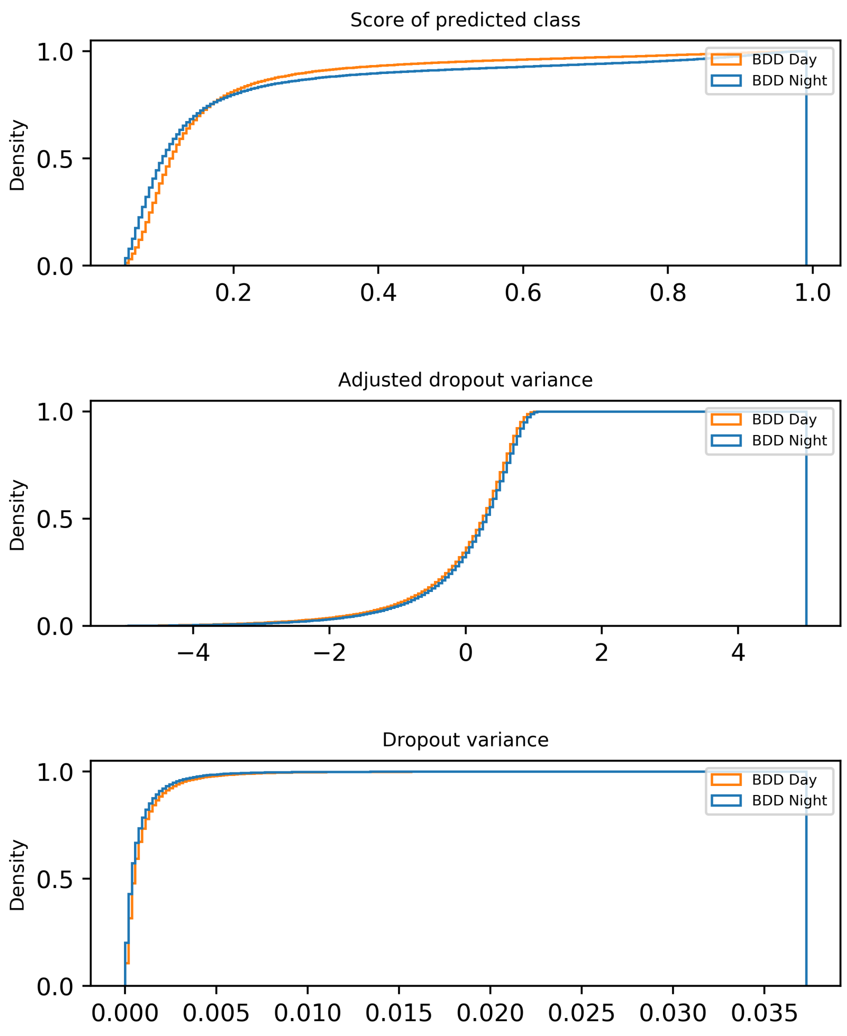


FIGURE 5.6: Cumulative density histogram of sigmoid scores, adjusted dropout variance, and dropout variance for the DAY vs. NIGHT split for **BB-20**. All measures are taken with respect to the predicted class.

of this statistic indicate that the two samples do not originate from the same underlying distribution. The results for this statistic are presented in Table 5.4. Overall, the adjusted variance for SN-20 results in the largest discriminability according to the statistic, which is approximately twice as large as the KS statistic for the sigmoid score of the BB models and the noDrop models. Note that for the BB models, the adjusted dropout variance is not a useful measure for discriminating between the two distributions, as is shown for BB-20 in Figure 5.6.

	Sigmoid score	Variance	Adj. variance
SN-20	0.183	0.001	**0.357**
SN2D-20	0.181	0.002	**0.253**
BB-10	**0.181**	0.001	0.075
BB-20	**0.181**	0.001	0.069
BB2D-10	**0.181**	0.001	0.054
NoDrop	**0.184**		

TABLE 5.4: The Kolmogorov-Smirnov statistic for the three measures, comparing the distribution of DAY images to that of NIGHT images. Large KS statistics reflect larger differences in distributions.

5.9 Regression

For the dropout models, regression variance is obtained similarly to the classification case: 30 stochastic forward passes are performed, from which we compute the mean and standard deviation of the regressed box offset. In contrast, the Laplace model does not require sampling and directly predicts the variance of a Laplace distribution. In Figure 5.7, the first column shows the regression residuals (errors) with respect to the ground truth objects from the test set. In the second column, the predicted standard deviations are visualized, which can be used to compute normalized residuals (right column) by dividing the regression errors by the predicted variance. We can see that the normalized residuals of the Laplace model fit the standard Laplace distribution rather well, which indicates that the model has successfully learned to predict a Laplace distribution for the regression task.

The Gauss model performs similarly, but since both the regression residual and the normalized residual resemble a Laplace distribution more than a Gaussian distribution, it would be more natural to opt for the Laplace regression, which is also associated with a slightly better performance [3] (see Table 5.2).

In contrast, the variance that is predicted by the dropout models is rather low. Consequently, the normalized residuals have a much higher spread compared to the Laplace model. In this case, one may still use the dropout variance for instance to compare regression uncertainty between different datasets. However, the parametric Laplace model can also be used to make probabilistic inferences about the precise location of objects.

By inspecting the Laplace regression uncertainty overlayed on the predicted bounding boxes, we generally find that the model seems to have learned meaningful regression uncertainty. For instance, regression uncertainty is usually larger when objects are partially occluded. Two examples are shown in Figure 5.8 and Figure 5.9.

[3]the difference in performance was more pronounced on preliminary experiments on the similar KITTI dataset [15].

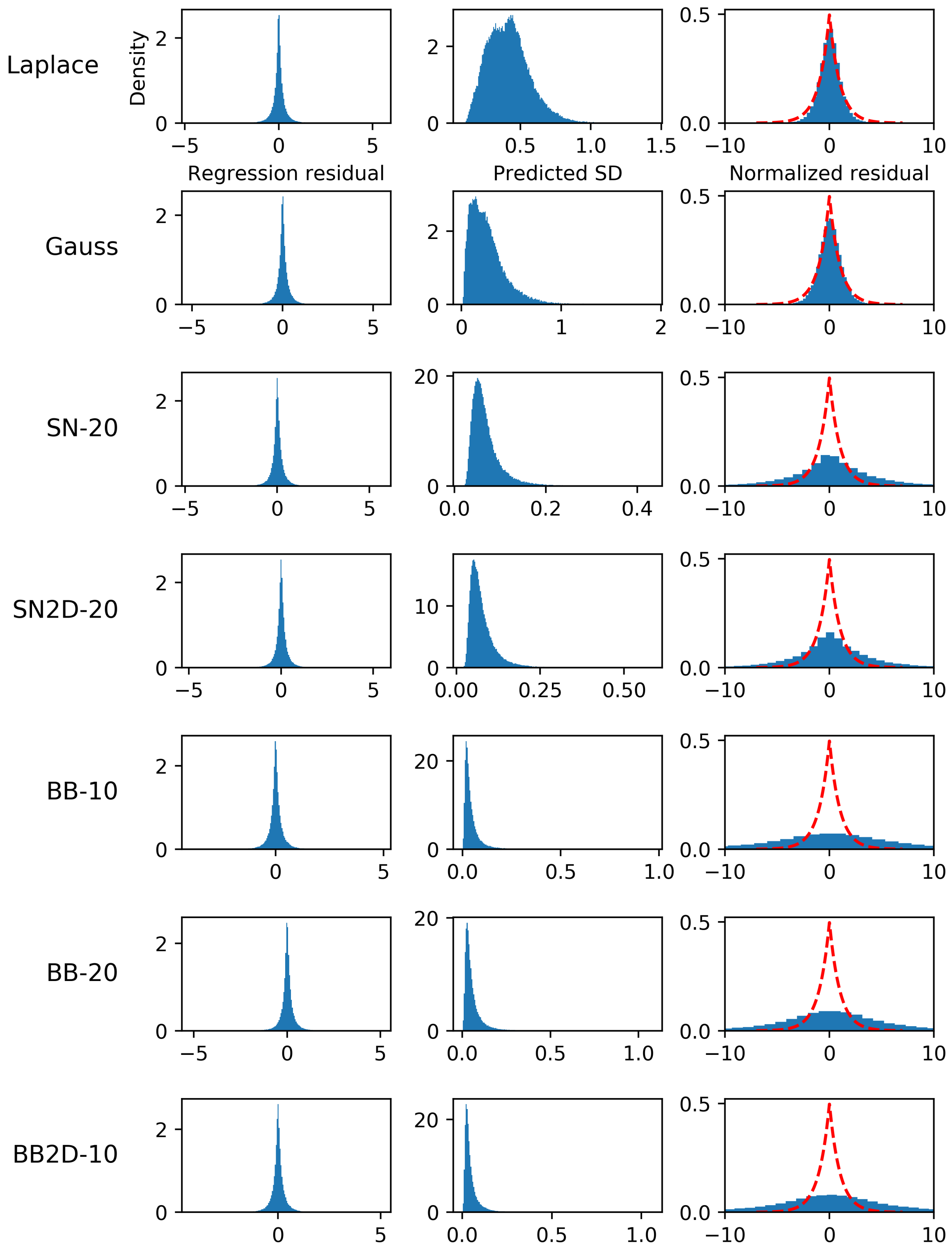


FIGURE 5.7: Bounding box regression: Regression error (left column), predicted STD (middle column), and normalized residuals (right column) of seven trained models. Normalized residuals are residuals that are divided by the predicted standard deviation. The predicted standard deviation is a learned model output for the first two rows and dropout uncertainty for the rows below that. The red line in the right column represents a standard Laplace distribution.

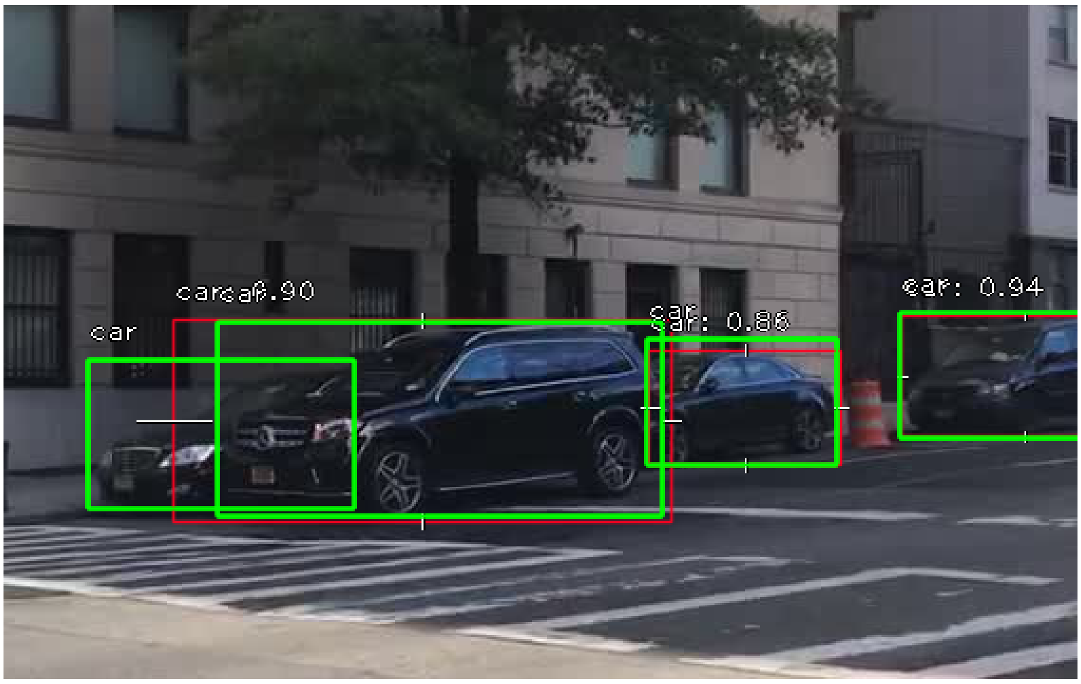


FIGURE 5.8: Predicted bounding boxes (red) and ground truth boxes (green). Predicted Laplace uncertainty is shown as white bars, indicating ±1 standard deviation. This image is a crop from the original image.

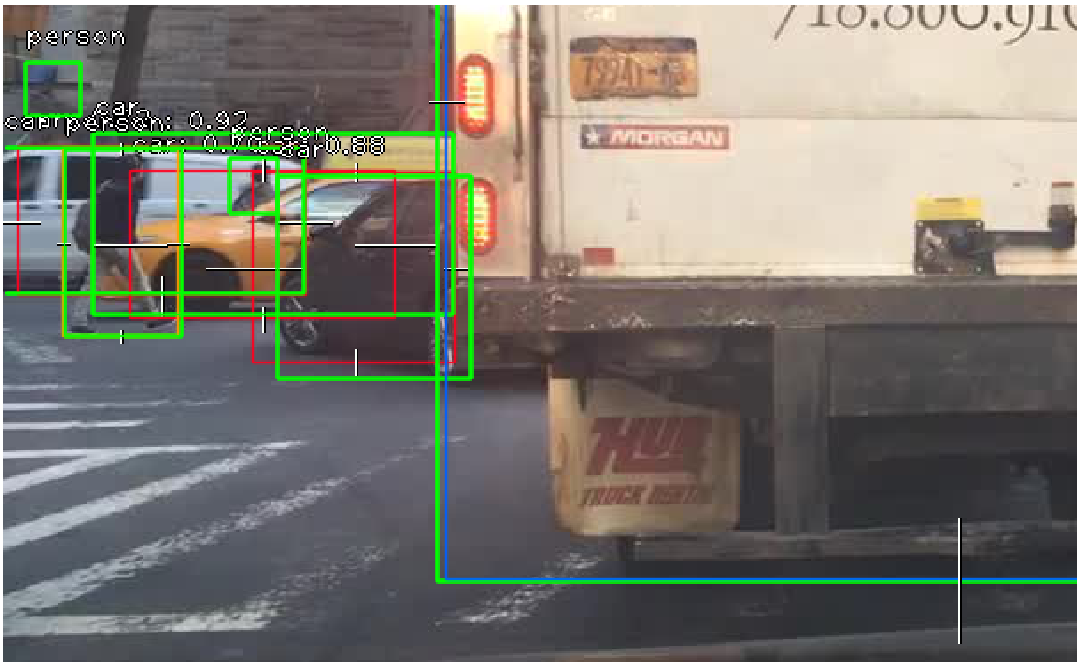


FIGURE 5.9: Ground truth boxes (green) and predicted boxes (red for cars, yellow for the person, blue for the truck). Predicted Laplace uncertainty is shown as white bars, indicating ±1 standard deviation. This image is a crop from the original image.

5.10 Novelty Detection

Novelty detection, also referred to as anomaly detection, is closely related to domain-shift detection and deals with the detection of inputs to the system that have not been encountered during training. Given that with the Laplace and Gauss models we provide over learned regression uncertainty, we surmised that this could be used in order to detect novel objects of interest. Since everything that is visible can be considered an object in some sense (i.e. there is no real background), we are especially interested in novel objects that have some similarity with objects categories from the dataset. We hypothesized that compared to instances of known classes, these objects will more frequently have a combination of high classification uncertainty, coupled with low regression uncertainty. This relies on the assumption that the class-agnostic bounding box regression generalizes to novel, but similar classes.

5.10.1 Dataset

To investigate this, we created an additional dataset split, removing all images in which at least one bus or truck is present from the DAY set (we further refer to this set as the DAY-Car set). The images that were removed together form the DAY-TB set. This results in test and validation sets of 1,686 images each, a training set comprising 22,773 images, a bus/truck subset of 21,898 images containing at least one bus or truck and 31,957 night images (which were not used during further analysis).

A model variant of the Laplace model was trained on the DAY-Car dataset, using the same training configuration that was described above, but now for 6 instead of the 8 classes that were used previously.

5.10.2 Regression Uncertainty

We test this hypothesis by finding predictions for DAY-TB that have at least a sigmoid confidence of 0.25 for the car class and match a ground truth bounding box of a car, bus, or truck by at least IoU = 0.5. Bounding boxes that are smaller on one side than 100 pixels are excluded because we want to focus on objects that are not too far away from the camera. We plot the sigmoid confidence and the average regression uncertainty of the four box sides in Figure 5.10. While it seems that there is a preponderance of buses and trucks in the region of high regression confidence and low sigmoid confidence, at closer inspection we can see that the regression uncertainty does not seem to facilitate the separation of cars from trucks and buses. In other words, the sigmoid uncertainty is much more discriminative for the three classes.

5.10.3 Autoencoder Reconstruction Error

As an additional step, we want to see whether the reconstruction error of an autoencoder can serve as an anomaly score for the same task, as discussed in section 3.6. We follow the previously discussed configuration that was applied to the disease detection task, [41], but adapt the setup for object detection. To train the autoencoder, we only consider detections with a sigmoid confidence of at least 0.15 because the large majority of the outputs corresponds to less confident predictions. Recall that the regression subnet and classification subnet have output dimensions height $\times$ width $\times$ #anchors $\times$ #classes, and height $\times$ width $\times$ #anchors $\times$ 4 respectively. For each prediction, the input for the autoencoder is the concatenation of the two 256 dimensional (depthwise) feature vectors in the corresponding (row, column)

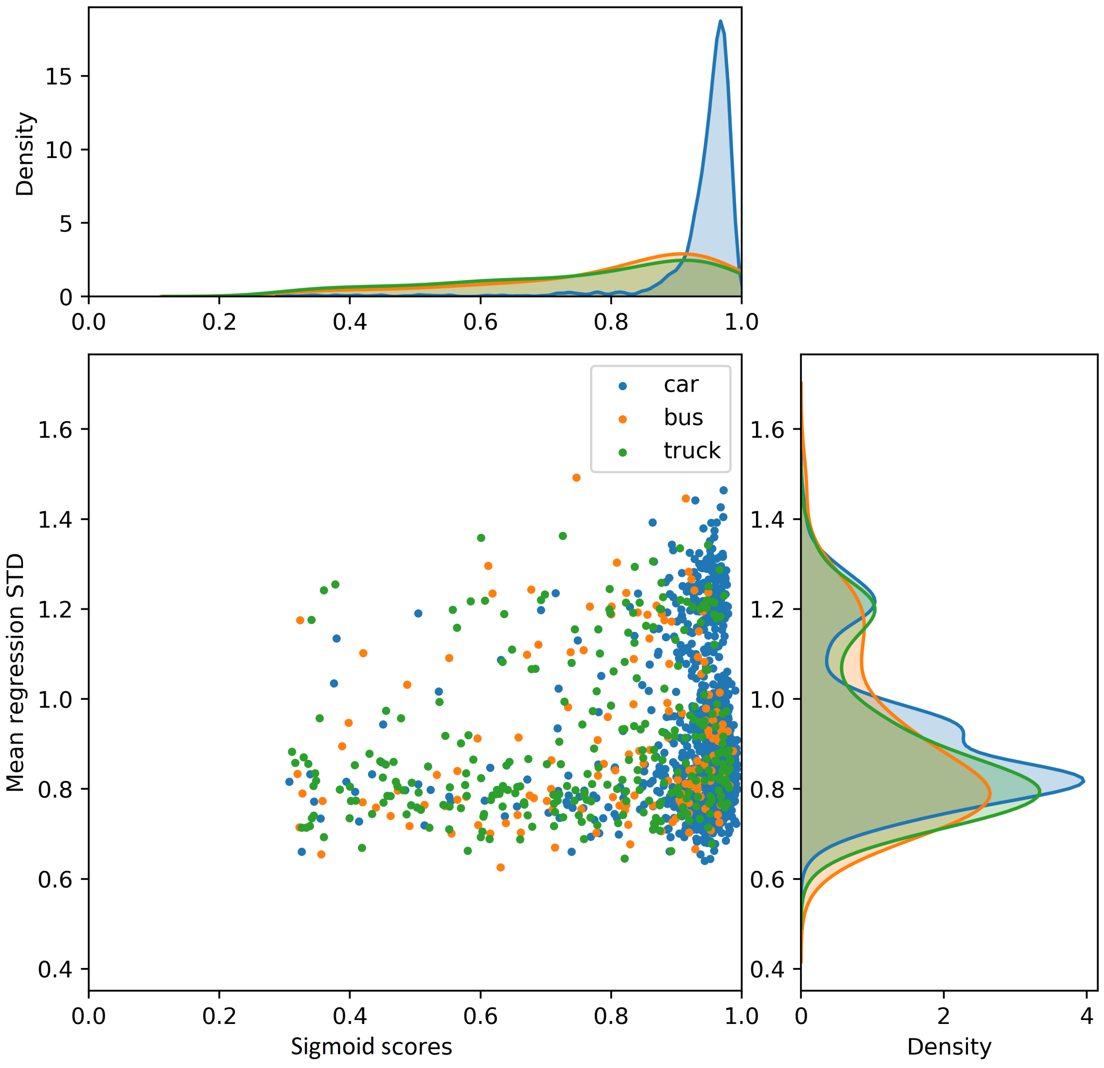


FIGURE 5.10: Scatterplot for instances of the classes car, bus, and truck, found in a test set of 1000 images. The x-axis of the scatterplot shows the sigmoid score for the class car. The y-axis shows the predicted standard deviation of the Laplace distribution, taking the average of the four sides of the bounding box. The network was trained with a Laplace regression loss on a training set which contains no trucks or buses. The density plots show marginal densities for the corresponding axes. Densities are estimated with a Gaussian kernel.

location in the penultimate layer of the classification and regression subnetworks. The input is encoded to 128 and then 32 features. The decoder reconstructs the code back to to 128 and then 512 features. Both encoder and decoder consist of two fully-connected layers with ReLU activation functions. This is visualized schematically in Figure 5.11. In practice, the actual detection output for the predicted box is also concatenated to the input to the autoencoder.

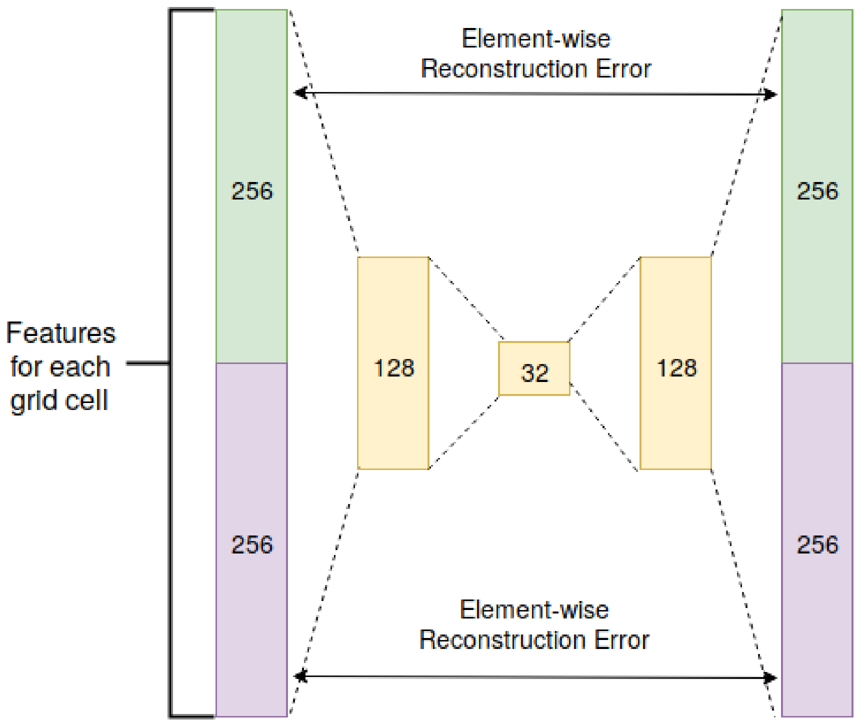


FIGURE 5.11: A schematic visualization of the autoencoder. For each prediction to be considered, the 256 dimensional feature vectors of the corresponding location (row, column) in the penultimate layer of the classification (green) and regression (purple) submodels are concatenated and encoded to 128 and then 32 features. The decoder reconstructs the code back to 128 and then 512 features. Both encoder and decoder consist of two fully-connected layers with a ReLU activation function.

The autoencoder is trained using ADAM with a learning rate of 5×10^{-5} and a batch size of 4 for 20,000 iterations (20 epochs with 1000 steps each), while freezing the parameters of the previously trained base network. As for the standard networks, the learning rate was decreased by half if the validation loss did not decrease for two subsequent epochs of 4000 iterations. If the validation loss did not decrease after three subsequent epochs, training was stopped.

Since we previously saw that the sigmoid score is much more discriminative compared to the regression uncertainty, we now look at the association between the sigmoid score and the reconstruction error. This is shown in Figure 5.12 for an autoencoder that is trained using the MSE of the reconstruction. Contrary to our expectations, buses and trucks are generally associated with a *lower* reconstruction error compared to the cars. Furthermore, there is a non-linear association between the anomaly score and the sigmoid uncertainty in that large anomaly scores are only observed for confident predictions (> 0.9) which predominantly correspond to cars. We first assumed that this may be related to the fact that the majority of predictions that are used for training the autoencoder have low sigmoid confidence, even after thresholding at 0.15. However, we repeated the procedure with sigmoid thresholds of 0.25 and 0.75 using both the MSE and the MAE as a loss function and did not find that that the novel classes are associated with higher reconstruction errors. We also looked at these configurations without using an activation function, thereby mapping the autoencoder to a linear mapping. In all of these cases, the reconstruction

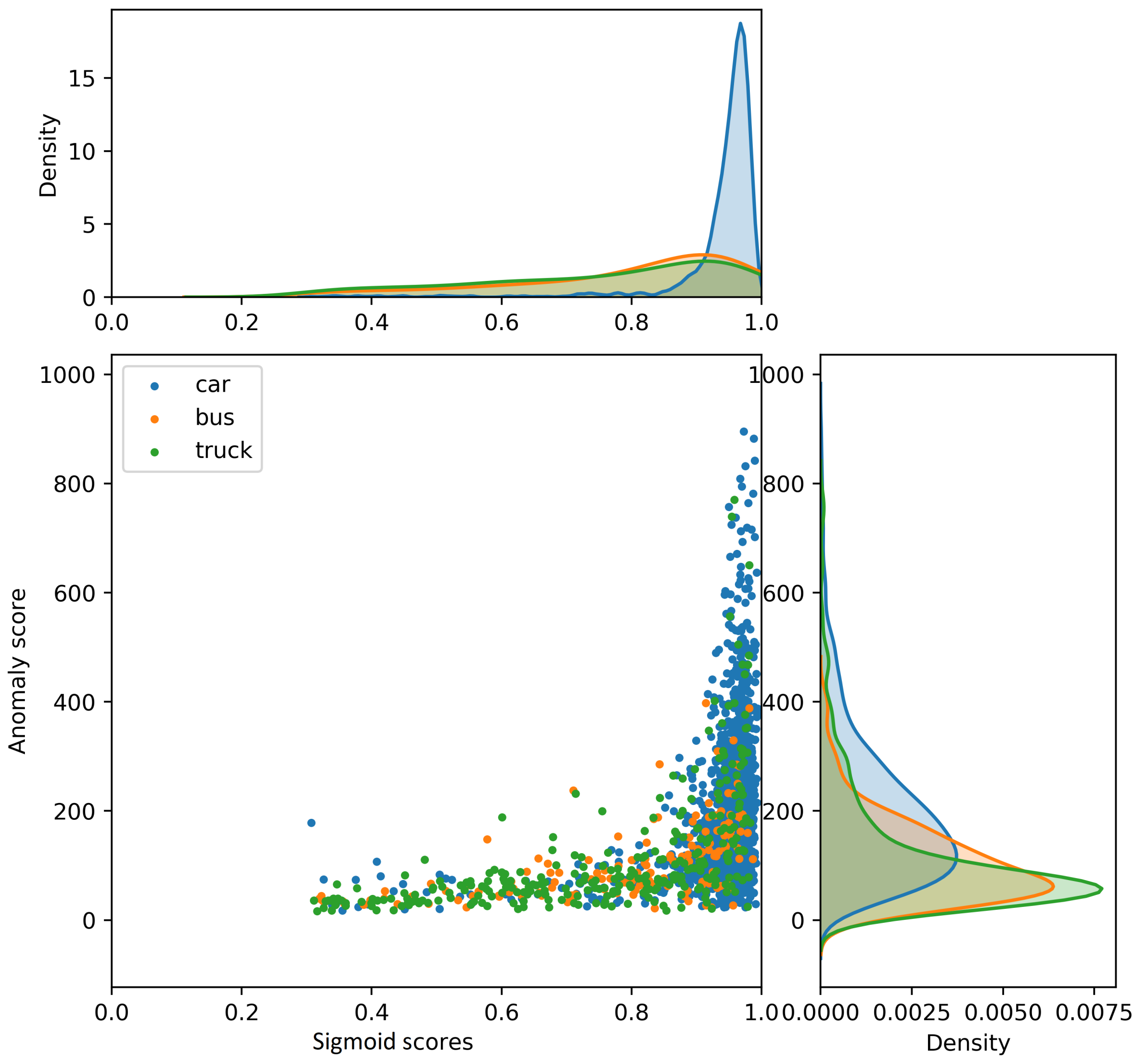


FIGURE 5.12: Scatterplot for instances of the classes car, bus, and truck, found in a test set of 1000 images. The x-axis of the scatterplot shows the sigmoid score for the class car. The y-axis shows the anomaly score (reconstruction error) that is obtained by the autoencoder that is applied to the feature vector associated with the individual predictions. The network was trained with a Laplace regression loss on a training set which contains no trucks or buses. The autoencoder was trained with MSE (cf. Figure B.1 in Appendix B for the MAE). The density plots show marginal densities for the corresponding axes. Densities are estimated with a Gaussian kernel.

error does appear to improve the separation between classes compared to using the sigmoid uncertainty alone.

5.11 Discussion

The main purpose of the experiments above was to see whether dropout uncertainty can be of additional value when performing object detection with a state-of-the-art detection NN. In order to maintain the model performance as measured by mAP, we opted for relatively small dropout probabilities because providing an additional uncertainty measure at the cost of detection performance is unlikely to be useful in practice.

We found that there are only small differences in model calibration when dropout is used. When looking at the decision referral task, dropout uncertainty did not result in an increase in performance when compared to the entropy of the sigmoid output of the standard network. This suggests that for object detection, the sigmoid detection output is a better predictor for which examples are most difficult.

For domain-shift detection, it initially appears that the distribution of dropout uncertainty is less discriminative compared to the sigmoid output for detecting a difference between the DAY and the NIGHT test sets. However, by adjusting the dropout variance for the non-linear association with the sigmoid confidence, a relatively large increase in discriminability as measured by the KS statistic is observed for the SN dropout models.

When looking at regression uncertainty, training the model to predict the variance of a Laplace noise distribution results in normalized regression residuals that fall within the expected standard Laplace distribution, indicating a good fit between predicted variance and observed regression residuals.

In contrast, regression variance that is obtained by dropout tends to underestimate the residual in our experiments, resulting in a wider distribution when we divide the observed residual by the predicted variance. For individual predictions, this means that regression variance obtained by dropout is less effective at narrowing down a range of values that the error is likely to fall into compared to the Laplace regression, which provides more reliable, probabilistic estimates.

We further found that using regression uncertainty for detecting novel objects does not seem to constitute an improvement compared to just using the sigmoid score by itself. Based on the work presented in [41] on disease detection, we expected the reconstruction error of the autoencoder, when applied to the hidden features, to increase for novel objects. Counter to our expectations, the reconstruction error was smaller, not larger for novel objects. Again, the sigmoid score by itself seems most discriminative.

Chapter 6

Conclusion

6.1 Summary of Results

In this thesis, we examined a number of scenarios in which additional uncertainty estimates may be of value when performing object detection with an NN. Overall, we found mixed evidence for the hypothesis that adding dropout uncertainty to the NN provides added value beyond the standard output. We now revisit our research questions in light of the experimental results shown in Chapter 5.

1. How does dropout affect the calibration of the model? We found in our experiment that variants of RetinaNet which perform predictions with MC-dropout showed very similar calibration when compared to the standard version of RetinaNet. At the outset, we did not have particular reasons to assume that calibration would be improved by dropout other than perhaps that predictions would be more robust when they are the result of several stochastic forward passes instead of a single deterministic forward pass. However, with respect to model calibration, this intuition does not seem to hold. If good calibration is required, dedicated approaches which calibrate trained models [24] should be considered.

2. Does the dropout uncertainty provide value when used as a measure that is used for decision referral? When performing decision referral in the BDD dataset, we found that dropout uncertainty did not result in an increase in performance when compared to the entropy of the sigmoid output of the standard network. This is surprising, considering the research in [41], where it was found that only dropout uncertainty and not the softmax classification score was associated with consistent increases in performance. This hints at the fact that the sigmoid detection output is a better predictor for which examples are most difficult, perhaps because the large number of background patches that the model encounters during training make the output more robust compared to classification tasks. Note that we do not attribute this difference in findings to the different output layer types because the softmax function can be considered a generalization of the sigmoid function to multiple object categories when classification is mutually exclusive[1]. Rather, we attribute this to the large difference in the number of instances corresponding to a non-target (background) class that is present in object detection, but often not in classification tasks.

3. Can dropout uncertainty be used to detect covariate shift? We trained different versions of RetinaNet on the DAY split and tested these models on the

[1]Some object-detection NNs such as SSD do in fact use a softmax layer instead of independent sigmoids for each class

NIGHT dataset to assess how this covariate shift would be reflected in the distribution of output and dropout uncertainty scores. In this setting, we found that standard dropout uncertainty was less discriminative compared to the sigmoid output for detecting a difference in distribution between the DAY and the NIGHT test sets. However, after adjusting the dropout variance for the non-linear association with the sigmoid confidence, the Kolmogorov–Smirnov statistic indicates that such dropout variance is much more predictive compared to the standard output for the subnetwork dropout models. This suggests that such a type of adjusted dropout variance can be used to detect covariate shift when the shift in distribution results in a new data distribution which is different, but still shares a minimum level of similarity with the original distribution. If the shift is so significant that the network has to process completely different inputs (such as only completely white or black images in the extreme case), we would predict that the KS statistic would be largest for the standard sigmoid scores because we would expect the model to predict only background and with high confidence.

4. Can we model bounding box regression uncertainty by training the model to predict the regression variance? How does this compare to the dropout uncertainty for regression? Our experiments showed that assuming a Laplace noise distribution and training the model to predict the variance for the regression errors results in normalized residuals that coincide with the expected standard Laplace distribution. While learning to predict regression variance has been investigated before (e.g. [33]), we found that this is also possible for object-detection networks like RetinaNet, which normally use the smooth L1 loss for regression. This allows us to make probabilistic predictions with respect to the bounding box dimensions. In contrast, the dropout regression variance tends to underestimate the residual and therefore has much fatter tails compared to the Laplace distribution. Considering the distinction between aleatoric and epistemic uncertainty, one may conceive of scenarios in which it would be useful to provide both types of uncertainty by using the Laplace and dropout uncertainties respectively. For most practical purposes, however, it seems that the Laplace regression suffices to estimate regression uncertainty and by visual inspection we did not find that dropout uncertainty and Laplace uncertainty reflected different underlying causes such as visual noise versus visual semantic ambiguity.

To answer these research questions, we compared models that used different types of dropout (standard dropout and SpatialDropout [62]) in different parts of the network (backbone and subnetworks). For comparable mAP scores, dropout in the subnetworks resulted in larger output variance, which we concluded to be due to the observation that feature maps in deep layers are semantically richer [66] compared to early layers in the network. Since adjusted dropout variance to detect covariate shift was only most predictive for dropout in the subnetworks but not in the backbone, we hypothesize that perhaps dropout in layers that are close to the output layers results in more meaningful variation with respect to the prediction task.

It is further worth mentioning that SpatialDropout resulted in larger output variation compared to standard dropout for equal dropout rates. However, whether dropout was performed in the backbone or in the subnetworks had a much larger effect on performance in all of our experiments.

In summary, Monte-Carlo dropout did not improve calibration and did not improve the performance of the model when used as a decision-referral criterion. We found dropout uncertainty most promising for detecting covariate shift, but in order to make definite claims about the effect of adjusted dropout variance for this purpose, further research is required, using different datasets and object-detection networks. Research on the subject of covariate shift and novelty detection would further require a thorough comparison with approaches that are dedicated to this purpose.

One conclusion that can be drawn from our results is that prior research on image classification does not necessarily generalize well to the object detection domain. This becomes clear by contrasting our results with the results in [41], in which dropout uncertainty was a useful criterion for decision referral and in which the reconstruction error that was obtained by an autoencoder was larger for novel data, which we did not replicate in our experiments. It can be argued that this is due to the output of object-detection networks being more robust compared to the output of classification models if these do not encounter a diverse set of non-target inputs during training.

6.2 Recommendations for Future Research

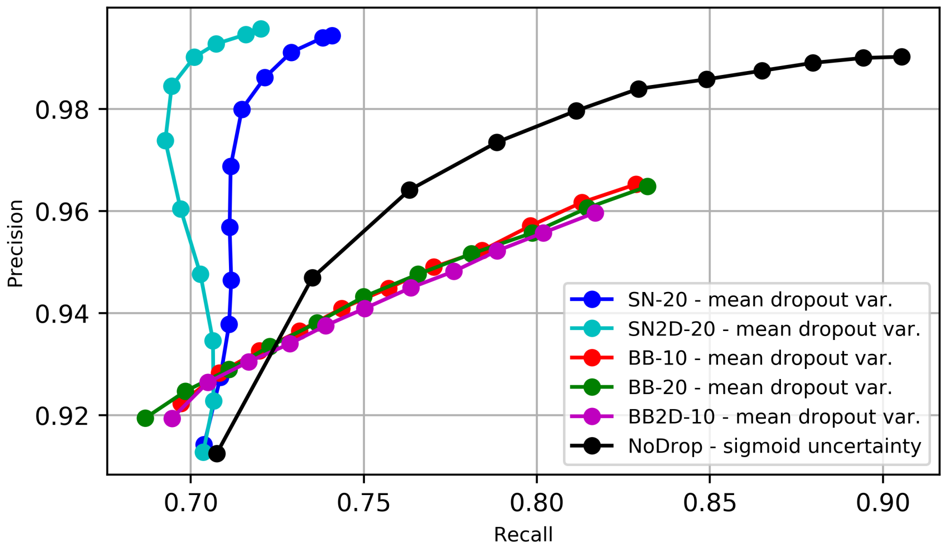
Previous research [14] showed that dropout uncertainty could be used for identifying inputs that the model is uncertain about and that would be most valuable to be trained on in an active learning paradigm. Given the results that we obtained in the decision-referral task, we suspect that this would not work well in our object-detection set-up as the sigmoid output was the best criterion for identifying difficult examples. Although we did not pursue this question further, active learning would still be an interesting topic for future research.

There is a large number of stochastic regularization techniques which could be investigated in a similar manner compared to our investigation of dropout. For example, we could replace Bernoulli dropout with multiplicative Gaussian noise [60] or distort the input images and observe the variation in network output.

Another avenue of investigation would be to extend the feasibility of Bayesian NNs such as the one proposed in [3] to large object-detection models like RetinaNet. It can be argued that this will become increasingly interesting as the performance and memory of GPUs increases over time.

Appendix A

Decision Referral - Other Metrics



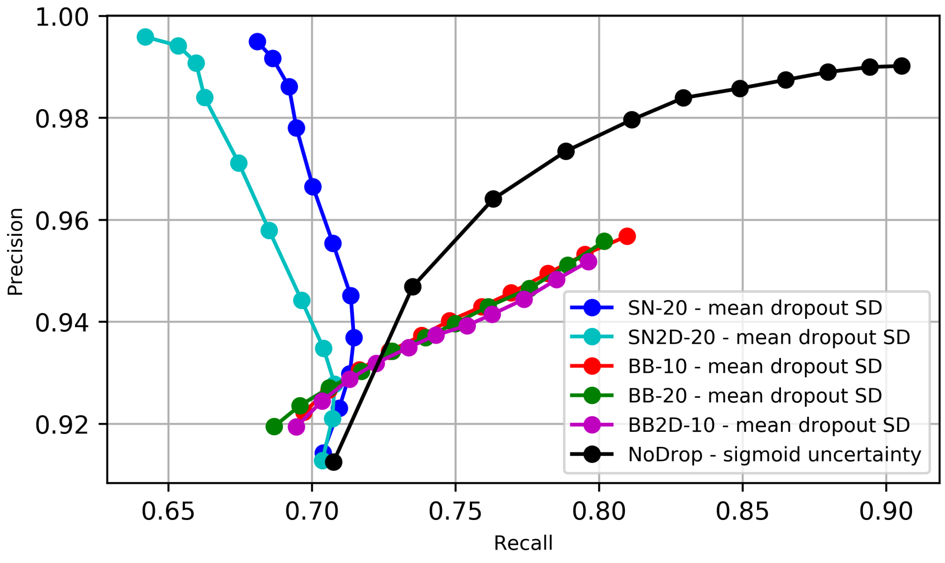


FIGURE A.1: Same as Figure 5.4 on page 43, but using the mean dropout variance (top) and the mean standard deviation (bottom) in the sigmoid output across classes instead of the predicted class as the referral criterion.

Appendix B

Novelty Detection with MAE

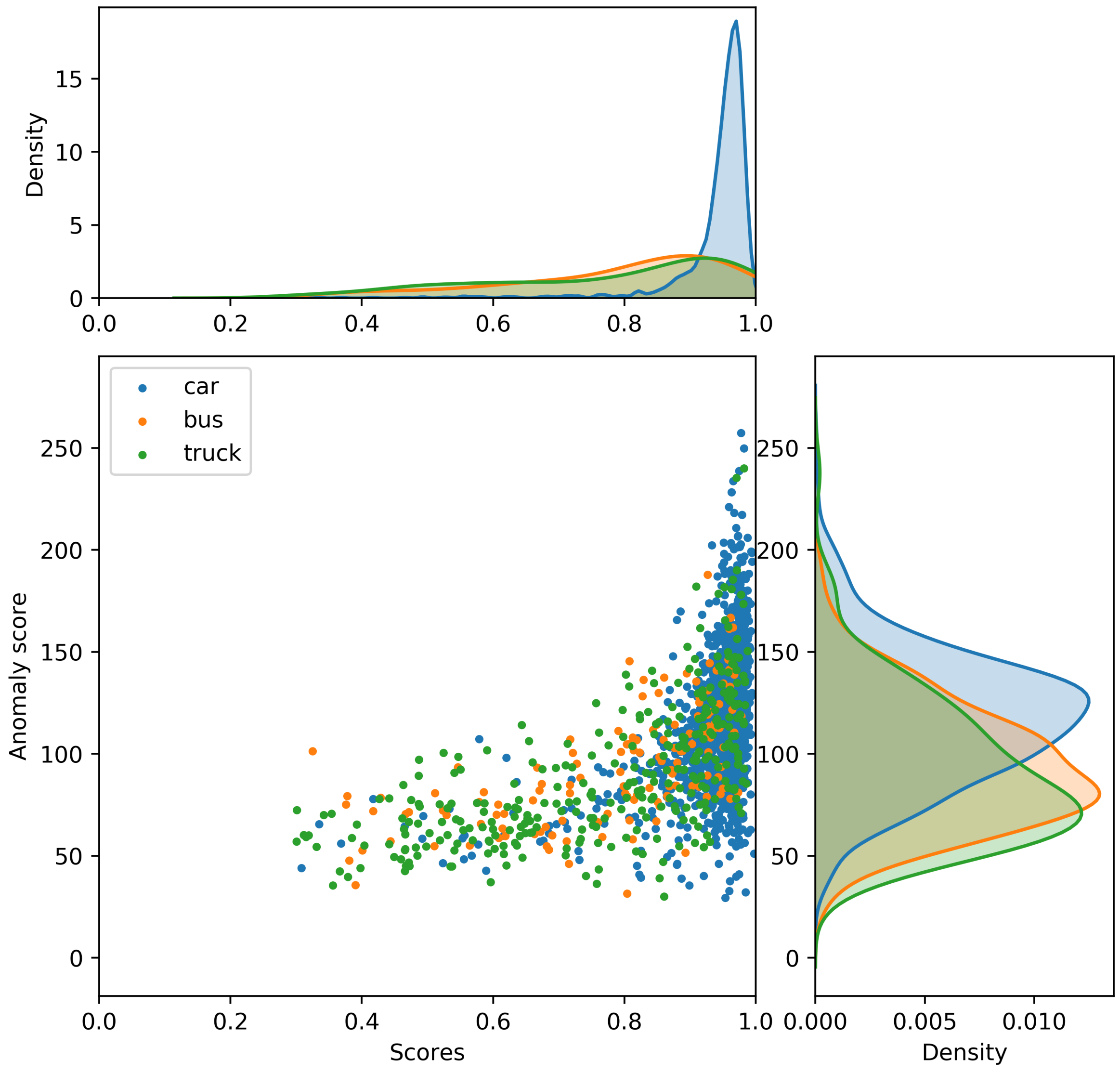


FIGURE B.1: Same as Figure 5.12 on page 52, but for an autoencoder trained using the MAE instead of the MSE.

Bibliography

[1] Defense Advanced Research Projects Agency (DARPA). *Training AI to Win a Dogfight*. 2019. URL: `https://www.darpa.mil/news-events/2019-05-08` (visited on 05/13/2019).

[2] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. “Variational inference: A review for statisticians”. In: *Journal of the American Statistical Association* 112.518 (2017), pp. 859–877.

[3] Charles Blundell et al. “Weight uncertainty in neural networks”. In: *arXiv preprint arXiv:1505.05424* (2015).

[4] Léon Bottou. “Large-scale machine learning with stochastic gradient descent”. In: *Proceedings of COMPSTAT’2010*. Springer, 2010, pp. 177–186.

[5] Yaroslav Bulatov. “NotMNIST dataset”. In: *Google (Books/OCR), Tech. Rep.[Online]. Available: http://yaroslavvb. blogspot. it/2011/09/notmnist-dataset. html* (2011).

[6] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[7] Andreas Damianou and Neil Lawrence. “Deep gaussian processes”. In: *Artificial Intelligence and Statistics*. 2013, pp. 207–215.

[8] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. IEEE. 2009, pp. 248–255.

[9] Armen Der Kiureghian and Ove Ditlevsen. “Aleatory or epistemic? Does it matter?” In: *Structural Safety* 31.2 (2009), pp. 105–112.

[10] Mark Everingham et al. “The pascal visual object classes (VOC) challenge”. In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[11] Yoav Freund and Robert E Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting”. In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.

[12] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York, 2001.

[13] Cheng-Yang Fu et al. “DSSD: Deconvolutional single shot detector”. In: *arXiv preprint arXiv:1701.06659* (2017).

[14] Yarin Gal and Zoubin Ghahramani. “Dropout as a Bayesian approximation: Representing model uncertainty in deep learning”. In: *international conference on machine learning*. 2016, pp. 1050–1059.

[15] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for autonomous driving? The KITTI vision benchmark suite”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 3354–3361.

[16] Ross Girshick. “Fast R-CNN”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[17] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[18] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep sparse rectifier neural networks”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 315–323.

[19] Tilmann Gneiting and Adrian E Raftery. “Strictly proper scoring rules, prediction, and estimation”. In: *Journal of the American Statistical Association* 102.477 (2007), pp. 359–378.

[20] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: *arXiv preprint arXiv:1412.6572* (2014).

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[22] Alex Graves. “Practical variational inference for neural networks”. In: *Advances in neural information processing systems*. 2011, pp. 2348–2356.

[23] Alex Graves and Navdeep Jaitly. “Towards end-to-end speech recognition with recurrent neural networks”. In: *International Conference on Machine Learning*. 2014, pp. 1764–1772.

[24] Chuan Guo et al. “On calibration of modern neural networks”. In: *arXiv preprint arXiv:1706.04599* (2017).

[25] Kevin Gurney. *An introduction to neural networks*. CRC press, 2014.

[26] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[27] Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on Imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[28] Dan Hendrycks and Kevin Gimpel. “A baseline for detecting misclassified and out-of-distribution examples in neural networks”. In: *arXiv preprint arXiv:1610.02136* (2016).

[29] Geoffrey E Hinton et al. “Improving neural networks by preventing co-adaptation of feature detectors”. In: *arXiv preprint arXiv:1207.0580* (2012).

[30] Geoffrey Hinton, Nitsh Srivastava, and Kevin Swersky. “Neural networks for machine learning”. In: *Coursera, video lectures* 264 (2012).

[31] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural networks* 4.2 (1991), pp. 251–257.

[32] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015).

[33] Alex Kendall and Yarin Gal. “What uncertainties do we need in bayesian deep learning for computer vision?” In: *Advances in neural information processing systems*. 2017, pp. 5574–5584.

[34] Nitish Shirish Keskar et al. “On large-batch training for deep learning: Generalization gap and sharp minima”. In: *arXiv preprint arXiv:1609.04836* (2016).

[35] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[37] Anders Krogh and John A Hertz. "A simple weight decay can improve generalization". In: *Advances in neural information processing systems*. 1992, pp. 950–957.

[38] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles". In: *arXiv preprint arXiv:1612.01474* (2016).

[39] Yann LeCun, Corinna Cortes, and Christopher JC Burges. "The MNIST database of handwritten digits, 1998". In: *URL http://yann. lecun. com/exdb/mnist* 10 (1998), p. 34.

[40] Kimin Lee et al. "Training Confidence-calibrated Classifiers for Detecting Out-of-Distribution Samples". In: *arXiv preprint arXiv:1711.09325* (2017).

[41] Christian Leibig et al. "Leveraging uncertainty information from deep neural networks for disease detection". In: *Scientific reports* 7.1 (2017), p. 17816.

[42] Sam Levin and Julia C Wong. *Self-driving Uber kills Arizona woman in first fatal crash involving pedestrian*. 2018. URL: `https://www.theguardian.com/technology/2018/mar/19/uber-self-driving-car-kills-woman-arizona-tempe` (visited on 05/13/2019).

[43] Tsung-Yi Lin et al. "Feature pyramid networks for object detection". In: *CVPR*. Vol. 1. 2. 2017, p. 4.

[44] Tsung-Yi Lin et al. "Focal loss for dense object detection". In: *arXiv preprint arXiv:1708.02002* (2017).

[45] Tsung-Yi Lin et al. "Microsoft COCO: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[46] Wei Liu et al. "SSD: Single shot multibox detector". In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[47] Christos Louizos and Max Welling. "Multiplicative normalizing flows for variational bayesian neural networks". In: *arXiv preprint arXiv:1703.01961* (2017).

[48] David G Lowe et al. "Object recognition from local scale-invariant features." In: *ICCV*. Vol. 99. 2. 1999, pp. 1150–1157.

[49] Marvin Minsky and Seymour Papert. "Perceptrons: An introduction to computational geometry". In: *MITPress, Cambridge, Massachusetts* (1969).

[50] Molly Mulshine. *A major flaw in Google's algorithm allegedly tagged two black people's faces with the word 'gorillas'*. 2015. URL: `https://www.businessinsider.com/google-tags-black-people-as-gorillas-2015-7` (visited on 05/13/2019).

[51] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[52] Radford M Neal. "Bayesian learning for neural networks". PhD thesis. University of Toronto, 1995.

[53] John Platt et al. "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods". In: *Advances in large margin classifiers* 10.3 (1999), pp. 61–74.

[54] Joaquin Quionero-Candela et al. *Dataset shift in machine learning*. The MIT Press, 2009.

[55] Shaoqing Ren et al. "Faster R-CNN: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[56] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[57] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3 (1988), p. 1.

[58] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[59] Irwin Sobel and Gary Feldman. "A 3x3 isotropic gradient operator for image processing". In: *a talk at the Stanford Artificial Project in* (1968), pp. 271–272.

[60] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[61] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[62] Jonathan Tompson et al. "Efficient object localization using convolutional networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 648–656.

[63] Jasper RR Uijlings et al. "Selective search for object recognition". In: *International journal of computer vision* 104.2 (2013), pp. 154–171.

[64] Paul Viola, Michael Jones, et al. "Rapid object detection using a boosted cascade of simple features". In: *CVPR (1)* 1 (2001), pp. 511–518.

[65] Fisher Yu et al. "BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling". In: *arXiv preprint arXiv:1805.04687* (2018).

[66] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.