Faculty of Electrical Engineering, Mathematics & Computer Science

Detecting Fog using Machine Learning and Investigating the Possibilities of Generating Synthetic Data

by

J.F.J. Blom

Master Thesis

Student Number:	4246438		
Defense Date:	17th of May 2023 at 15:00		
Thesis Committee :	prof.dr.ir. G. Jongbloed,	TU Delft,	chair
	dr. R.J. Fokkink,	TU Delft,	supervisor
	prof.dr.ir. M.B. van Gijzen,	TU Delft,	supervisor

Acknowledgements

This project would not have been possible without the help I received from my supervisors. Robbert Fokkink and Martin van Gijzen, thank you for the guidance, feedback, and support during the project. You always gave me new energy and motivation to keep going. I enjoyed the discussions about chess. I would like to thank Geurt Jongbloed for his enthusiasm and guidance. It is always a pleasure talking to you.

Thank you, Tom Viering, for your feedback and ideas. Mainly, the papers about image dehazing you provided were very useful. I used the papers to design a conceptual algorithm for adding artificial fog to clear images. Furthermore, I would like to thank Marco Loog for his expertise and feedback. I enjoyed all our meetings and discussions. I would like to thank Juan Juan Cai for her guidance and brainstorming sessions to find a proper research subject.

Finally, I would like to thank Jan Willem Noteboom and Andrea Pagani from KNMI for the research subject and for providing the dataset. I learned a lot during the internship at KNMI.

J.F.J. Blom,
Leiden, May 2023

Abstract

Fog plays a major role in chain collisions. Proper fog detection is essential for the Dutch road authority to anticipate foggy weather conditions. Dozens of stations in the Netherlands can measure fog. However, fog can be a very local phenomenon. Therefore, more local measurements are needed. There are about 5,000 traffic cameras in the Netherlands. Several studies on detecting fog on traffic cameras have been done. The most successful studies used machine learning classification models to detect fog. The biggest challenge they face is the extreme imbalance, limited diversity, and limited accuracy of the dataset. Obtaining adequate precision is one of the primary challenges since the extreme imbalance of the dataset significantly impacts precision. The main objective of this research is to improve the dataset and investigate many machine learning configurations. Another objective is to examine the possibilities of generating synthetic data.

This thesis uses a clever (re)labeling method, significantly improving the dataset's quality. However, it turned out that the dataset still has its limitations. A large portion of false positives are caused by labeling errors. After comparing several machine learning models, it follows that a 9-layer ResNET model is optimal. Adding more layers will not result in better performance. Unexpectedly, initializing ResNET with pre-trained weights actually decreases performance. In addition, the effect of oversampling and/or using a weighted binary cross-entropy loss is investigated. Just oversampling leads to overfitting, but using a weighted binary cross-entropy loss isn't ideal either. The best performance is achieved by combining weighted binary cross-entropy loss with oversampling. Decision threshold optimization substantially improved the results. The experiments allowed for selecting the ideal configuration, which substantially increased performance. The best-performing configuration achieved a strong correlation in the Matthews correlation coefficient.

Finally, the possibilities of generating synthetic data are investigated. ADASYN and SMOTe seem attractive at first sight, but from a recent study, it follows that they don't work better than random oversampling. One of the most promising ideas for generating synthetic data is to add fog to clear images. In this thesis, a conceptual algorithm is designed to add artificial fog to clear images. Most generated images look convincing, but there is much room for improvement.

Contents

List of abbreviations

ADASYN	Adaptive synthetic sampling approach for imbalanced learning (Haibo He, Yang Bai, Garcia, & Shutao Li, 2008)
CNN	Convolutional Neural Network (Explained in Section 3.5)
FFT	Fast Fourier Transform
FN	False Negatives
FP	False Positives
KNMI	The Royal Netherlands Meteorological Institute
MCC	Matthews Correlation Coefficient (Equation (5.5) or Equation (5.18))
MOR	Meteorological Optical Range (Explained in Chapter 1)
NPR	Negative prediciton rate (Table 5.2)
PPR	Positive prediciton rate or precision (Table 5.2)
ResNET	Residual Neural Network ((He, Zhang, Ren, & Sun, 2015) or Section 3.7)
SMOTe	Synthetic Minority Over-sampling Technique (Chawla, Bowyer, Hall, & Kegelmeyer, 2002)
T1	Decision threshold optimization according to Equation (6.3)
T2	Decision threshold optimization according to Equation (6.4)
TNR	True Negative Rate or specificity (Table 5.2)
TN	True Negatives
TPR	True Positive Rate or recall (Table 5.2)
TP	True Positives

Chapter 1

Introduction

This research is in cooperation with the Royal Netherlands Meteorological Institute (KNMI) and Delft University of Technology and is written to finish a Master of Science in Applied Mathematics. KNMI is a national research institute in the Netherlands. Their main research areas are weather, climate, air quality, and seismic activity.

Fog plays a major role in chain collisions. In January 2019, for instance, more than 20 vehicles crashed due to dense fog in Texas (Maxouris, 2019). Likewise, in Sao Paulo, Brazil, a chain collision involving more than 300 vehicles occurred in 2011 due to heavy fog (Sheth, 2019).

"The definition of fog is a visible aggregate of minute water particles near the Earth's surface that reduces horizontal visibility below 1 km (5/8 mile)" (U.S. Department of Commerce & Administration, 2017). Fog is called dense when the visibility is reduced to less than 250 meters; it is called light fog when the visibility is more than 250 meters but less than 1,000 meters. KNMI is mainly interested in detecting dense fog.

Compared to crashes in clear visibility conditions, crashes in foggy weather usually involve more vehicles and serious injuries (Wang, Liang, & Evans, 2017). Accidents due to fog especially occur in the winter months, in the early morning, and rural areas (Abdel-Aty, Ekram, Huang, & Choi, 2011). In addition, (Cavallo, Colomb, & Doré, 2001) found that their study participants perceived a lead car to be 60 percent farther away in foggy weather conditions than in clear weather conditions. These crash statistics motivate further research into predicting fog.

The amount of fog can be measured by the visibility. The visibility can be expressed in MOR (Meteorological Optical Range), defined as the distance required to reduce the intensity of a light source to 5 percent of its original value (Wauben & Roth, 2016). KNMI uses forward scatter sensors to determine the atmospheric extinction σ. The visibility (MOR) can be calculated by MOR $= \ln(0.05)/\sigma$. The disadvantage of these sensors is that they are quite expensive, and the number of sensors in the Netherlands is limited.

Furthermore, fog can be a local phenomenon; therefore, more local measurements are needed to get full coverage of the country.

The Dutch road authority owns about 5,000 traffic cameras, which can be used to measure visibility. In 2016, the visibility can be estimated using the landmark discrimination and contrast reduction method (Wauben & Roth, 2016). This method is designed for fixed cameras, but the cameras of the Dutch road authority can tilt, pan, and zoom. Therefore, a more advanced approach is needed. In 2018, convolutional deep neural networks were used to improve the detection further (Molleman, 2018).

The most significant problem is the extreme imbalance, limited diversity, and limited accuracy of the dataset. Only 0.7 percent of the images contain light fog, and only 0.2 percent contain dense fog. Currently, this problem is solved using a weighted cross-entropy loss (Molleman, 2018). The camera images are labeled using the visibility sensors of six measurement stations. In total, there are 25 measurement stations. If the camera is within a radius of 7.5 km of a measurement station, it will get a label based on the visibility measurement of the measurement station. For this reason, a limited number of traffic cameras are currently being used to create the dataset. This means a decrease in the diversity of the dataset. The diversity is slightly increased by randomly cropping and flipping the images. Another problem is that the test dataset does not represent the real world. The Dutch road authority uses tilt, pan, and zoom, which makes it sometimes almost impossible to determine if fog is present. Even for a human annotator, it is hard to determine. Therefore, the test dataset used in (Molleman, 2018) mainly contains images that give a clear overview. In addition, the test dataset is balanced, which is not an accurate representation of the real world.

In another study, fully connected neural networks were used to detect fog on traffic cameras (Andrea Pagani & Wauben, 2018). This paper uses a random split to get a training, validation, and test dataset. Therefore, the test dataset has the same imbalance as the training dataset. Also, in this paper, the main problem they encounter is the extreme imbalance, limited diversity, and limited accuracy of the dataset. For this reason, the dataset will also be one of the main challenges in this thesis.

1.1 Research objectives

The main objective of this research is to improve fog detection on camera images using machine learning techniques. The research objectives are divided into three categories: the dataset, the machine learning model, and synthetic data.

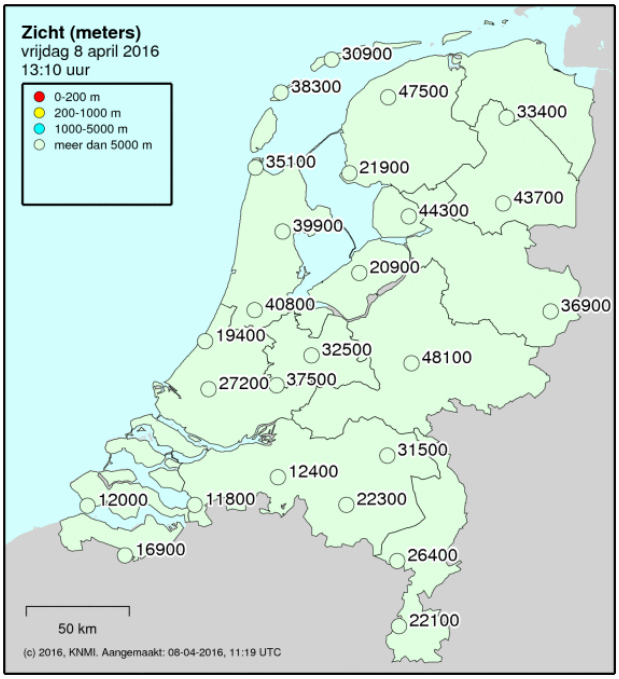


Figure 1.1: Locations of MOR-visibility sensors. The numbers on the map represent the visibility measurement in meters at that given time.

1.1.1 Dataset

The method used by (Molleman, 2018) to create the dataset isn't perfect and is prone to labeling errors. In addition, the labeled dataset created by (Molleman, 2018) isn't available anymore and therefore has to be recreated. KNMI provided images taken from the traffic cameras and measurements from MOR-visibility sensors. However, traffic cameras and the measurements from MOR-visibility sensors are not at the same location. Therefore, in order to create a labeled dataset, the measurements and the images taken from the traffic cameras must be combined somehow. In addition, the quality of the labeled dataset has to be evaluated. The research objectives concerning the dataset can be summarized in the following two research questions:

- **Research Question 1:** How can we create a properly labeled dataset?
- **Research Question 2:** What is the quality of this dataset?

1.1.2 Machine Learning Model

In order to get a proper understanding of the performance of several machine learning models on the dataset, several machine learning models will be tested in several configurations. Is a sophisticated model needed, or is a simpler model sufficient or even better? In

addition, some research on how to validate the models has to be done. A standard method to validate machine learning models does not exist. The research objectives concerning the machine learning models can be summarized in the following two research questions:

- **Research Question 3:** How do we evaluate the performance of a machine learning model?
- **Research Question 4:** How do several machine learning models and configurations perform on this dataset?

1.1.3 Synthetic Data

One of the main interests of KNMI is to investigate the possibility of improving the dataset by augmenting it with synthetic data. This is because the current dataset has limitations. Firstly, there is a large imbalance in the dataset. Secondly, the labeling method used by (Molleman, 2018) is prone to errors. Can we use synthetic data to create a better dataset? The research objectives concerning synthetic data can be summarized in the following two research questions:

- **Research Question 5:** What are the possibilities for generating synthetic data?
- **Research Question 6:** Is it possible to improve the performance of the neural network by augmenting the dataset with synthetic data?

1.2 Outline of Report

In this section, the overall structure of the report is described. In Chapter 2, the literature study is described. The literature study looks for other studies about imbalanced datasets and synthetic data. The fundamentals of neural networks are explained in Chapter 3. In addition, the neural network ResNET is explained and reviewed. In Chapter 4, the method for creating a labeled dataset is explained. In addition, the quality of the resulting dataset is reviewed. In Chapter 5, we explain how the experiments are done and how the models are validated. In Chapter 6, the results of the experiments are evaluated. What are the reasons that one configuration works better compared to another? In Chapter 7, the conclusion and the recommendations for future work are given. In the conclusion, we will determine if we achieved the research objectives in Section 1.1. In Appendix A, thoughts and findings on adding artificial fog to clear images are described. A conceptual method is proposed for the addition of fog to clear images.

Chapter 2

Related Work

The main focus of this literature study is finding other research about synthetic data and imbalanced datasets concerning machine learning. We focus on synthetic data because it is one of the main interests of KNMI. We focus on imbalanced datasets because the imbalanced dataset is one of the main challenges in this research.

In (Buda, Maki, & Mazurowski, 2018), several convolutional neural networks (CNN) are trained using several configurations and using several image classification datasets. The goal is to compare several methods that deal with imbalanced datasets. The following methods are compared:

- Random minority oversampling
- Random majority undersampling
- Two-phase training with pre-training on a randomly oversampled dataset
- Two-phase training with pre-training on a randomly undersampled dataset
- Optimizing the decision threshold
- Oversampling and optimizing the decision threshold
- Undersampling and optimizing the decision threshold

Two-phase training means that the neural network is first trained on a balanced dataset. A balanced dataset is achieved by oversampling the minority class or undersampling the majority class. In the second phase, the network will be trained on the unmodified, imbalanced dataset. The learning rate in the second phase is multiplied by 10^{-1}, so the learning rate in the second phase is smaller compared to the first phase. Optimizing the

decision threshold means that they choose an optimal decision threshold using the validation dataset. They show that oversampling does not result in overfitting with convolutional neural networks. In addition, they show that oversampling in combination with optimizing the decision threshold results in the best performance.

In (Chawla et al., 2002), they use synthetic data to augment an imbalanced dataset. It is one of the best-known papers about synthetic data. The paper has been cited over 23,000 times. The algorithm is called SMOTe (Synthetic Minority Over-sampling Technique). The algorithm is oversampling the minority class by generating synthetic data. The algorithm is designed to improve the performance of machine learning models on imbalanced datasets. They show in an experiment that SMOTe, in combination with undersampling performs better than plain undersampling. Undersampling means removing data points from the majority class to balance the dataset. In Algorithm 1, the complete SMOTe algorithm is given.

Algorithm 1 SMOTe

Set k and N according to preference
Let $\mathbf{D}_{minority}^{synthetic}$ be the empty set
Let $\mathbf{D}_{minority}$ the set of all data points from the minority class
Take a random sample $\mathbf{D}_{minority}^{sample} \subset \mathbf{D}_{minority}$ of size N
for $d \in \mathbf{D}_{minority}^{sample}$ **do**
  Calculate the k-nearest neighbours of d
  Let $\mathbf{V}$ be the set of k vectors from d to every nearest neighbor
  for $v \in \mathbf{V}$ **do**
    Take $k \in (0, 1)$ randomly and add the synthetic data point $d + k \cdot v$ to $\mathbf{D}_{minority}^{synthetic}$
  end for
end for
The set $\mathbf{D}_{minority}^{synthetic}$ now contains all the synthetic data points generated by the SMOTe algorithm

Later, another algorithm is proposed for generating synthetic images for the minority class. The algorithm is called ADASYN (Adaptive Synthetic Sampling Approach for Imbalanced Learning) (Haibo He et al., 2008). This algorithm is inspired by the SMOTe algorithm. The idea is that ADASYN will differentiate between minority samples that are easy to learn and those that are difficult to learn. If a minority sample is difficult to learn, more synthetic data points are generated around that minority data point. A minority sample is considered difficult to learn if the k-nearest neighbors of that minority sample contain many majority samples. Intuitively, this means that the minority sample is close to the decision boundary, which makes it difficult to differentiate between a majority and a minority sample. In Algorithm 2, the complete algorithm for ADASYN is given.

In some sense, r_i is the factor that determines how difficult it is to learn the minority

Algorithm 2 ADASYN

Choose $\beta \in (0, 1]$, which is the balance level after generating and augmenting the dataset with synthetic data. $\beta = 1$ means the dataset will be balanced completely.
Let $\mathbf{D}_{minority}^{synthetic}$ be the empty set
Let $\mathbf{D}_{training}$, $\mathbf{D}_{minority}$ and $\mathbf{D}_{majority}$ be all the data points of the training dataset, all the data points in the minority class and all the data points in the majority class.
Let m be the number of data points in $\mathbf{D}_{training}$ and let $m_{minority}$ and $m_{majority}$ be the number of minority data points and the number of majority data points.
Define $G = (m_{majority} - m_{minority})\beta$, which is the number of synthetic data points that have to be generated.
for $x_{minority} \in \mathbf{D}_{minority}$ **do**
  Calculate the k-nearest neighbors of $x_{minority}$
  Define $r_i = \Delta_i / k$, where $i \in \{1, \cdots m_{minority}\}$, Δ_i is the number of neighbors that are from the majority class.
  Define $\hat{r}_i = \sum_{i=1}^{m_{minority}} r_i$ (normalization)
  Define $g_i = \hat{r}_i \cdot G$, where g_i is the number of synthetic data points that have to be generated for that minority instance
  for $i \in \{1, \cdots, g_i\}$ **do**
    Randomly pick a minority data point $x_{minority}^{neighbor}$ from the k-nearest neighbors of $x_{minority}$
    Randomly pick $\lambda \in (0, 1)$
    Add $x_{minority} + (x_{minority} - x_{minority}^{neighbor})\lambda$ to $\mathbf{D}_{minority}^{synthetic}$
  end for
end for
$\mathbf{D}_{minority}^{synthetic}$ now contains all generated synthetic minority data points

data point. From the experiment in (Haibo He et al., 2008), it follows that the ADASYN algorithm is a slight improvement over the SMOTe algorithm. However, from a more recent paper, it becomes clear that SMOTe and ADASYN don't do better than random oversampling (Elor & Averbuch-Elor, 2022). They even argue that not oversampling in combination with threshold optimization is preferred because it performs similarly to oversampling in most cases. However, for two models, no oversampling with threshold optimization actually results in worse performance. They argue that this only happens if the model is a "weak classifier". In all other cases, oversampling, SMOTe, ADASYN, and no oversampling with threshold optimization perform similarly. No oversampling without threshold optimization performs the worst in all cases. This implies that just oversampling should be sufficient since it did well in all cases in their experiment. In this report, we will use oversampling as one of the methods to compensate for the class imbalance of the dataset.

An idea to create synthetic data is to add fog to non-foggy images. This would be

convenient since foggy images are very rare, but we have many clear images. There is one paper where this is investigated. In (Sakaridis, Dai, & Van Gool, 2018), the possibility of adding artificial fog to clear images is investigated. Their motivation is primarily to improve semantic segmentation in foggy circumstances. They are able to improve semantic segmentation using synthetic data. However, in this paper, stereoscopic images are used. In other words, the images contain a depth map. The images provided by KNMI don't contain a depth map. They use something called the image formation model, which is also used in many papers about image dehazing ((Berman, Treibitz, & Avidan, 2016), (Cai, Xu, Jia, Qing, & Tao, 2016), (Fattal, 2015), (He, Sun, & Tang, 2011), (Omer & Werman, 2004), (Ren, Pan, Zhang, Cao, & Yang, 2020), (Schechner, Narasimhan, & Nayar, 2001), (Schechner, Narasimhan, & Nayar, 2003), (Swami & Das, 2018), (Xu, Guo, Liu, & Ye, 2012)). The image formation model is given in Equation (2.1).

$$\mathbf{I}(\mathbf{x}) = t(\mathbf{x}) \cdot \mathbf{J}(\mathbf{x}) + [1 - t(\mathbf{x})] \cdot \mathbf{A} \tag{2.1}$$

$\mathbf{x}$ is the pixel coordinate, $\mathbf{I}(\mathbf{x})$ is the hazy image, $\mathbf{J}(\mathbf{x})$ is the clear image, $\mathbf{A}$ is a color representing the so-called atmospheric light, which is basically the color of the fog, and $t(\mathbf{x})$ is the transmission map. The transmission map is given in Equation (2.2).

$$t(\mathbf{x}) = e^{-\beta d(\mathbf{x})} \tag{2.2}$$

β is the attenuation coefficient of the atmosphere and $d(\mathbf{x})$ is the depth map. In the image formation model, β depends on the wavelength. Usually, this dependency is neglected to reduce the number of unknowns.

This image formation model could also be used to add fog to clear images if we find a way to estimate the parameters. Many papers about image dehazing have already developed methods to extract the parameters from foggy images. Although we are interested in doing the opposite (adding fog to images), there might be some interesting ideas we could use in those papers. In Appendix A, the possibility of adding fog to clear images using the image formation model is investigated.

Another interesting research area is domain adaptation ((Ben-David et al., 2010), (Mansour, Mohri, & Rostamizadeh, 2009), (Cortes & Mohri, 2014), (Cortes, Mohri, & Medina, 2019), (Germain, Habrard, Laviolette, & Morvant, 2015), (Zhang, Liu, Long, & Jordan, 2019), (Ganin et al., 2015), (Ben-David & Urner, 2014), (David, Lu, Luu, & Pal, 2010), (Ben-David & Urner, 2012)). Usually, it is assumed that the source and target datasets are drawn from the same probability distribution. However, this is not always the case. The data from the training dataset might be collected in different circumstances. Is it still possible to learn something if the source and the target distribution are (slightly) different? These are questions that researchers within the research area of domain adaptation try to answer. In (Johnson-Roberson, Barto, Mehta, Sridhar, & Vasudevan, 2016),

a neural network is trained on images of the game Grand Theft Auto 5 to detect cars. He tested the network on real-world images. The accuracy is not high $< 70\%$, but he showed that it is definitely possible to use 3D-rendered images for this application. This is an example where the source and target datasets are drawn from different probability distributions. In our case, if we generate synthetic data points, the synthetic data points will be drawn from a different probability distribution. Therefore, domain adaptation seems like an interesting research area for this thesis. However, the research area is very theoretical, and most papers don't have any practical applications.

Chapter 3

Neural Networks

In this chapter, the fundamentals of neural networks are explained. In addition, the neural network ResNET is explained and reviewed. A neural network in machine learning is a model inspired by the biological brain. The brain consists of neurons that can communicate with other neurons through synapses. Neurons are cells in the brain that can receive and send electrical signals. Synapses are the connections between neurons that allow a neuron to send signals to another neuron. The first computational neuron was introduced by Warren McCulloch and Walter Pitts (McCulloch & Pitts, 1943). This type of computational neuron only works with binary inputs. In (Rosenblatt, 1957), a new type of computational neuron is introduced called a perceptron. A block diagram of the perceptron model is depicted in Figure 3.1. In Equation (3.1), the corresponding mathematical expression is given. $a_1, \cdots, a_n$ are the inputs. $w_1, \cdots, w_n$ are the weights that must be determined. b is the bias that has to be determined.

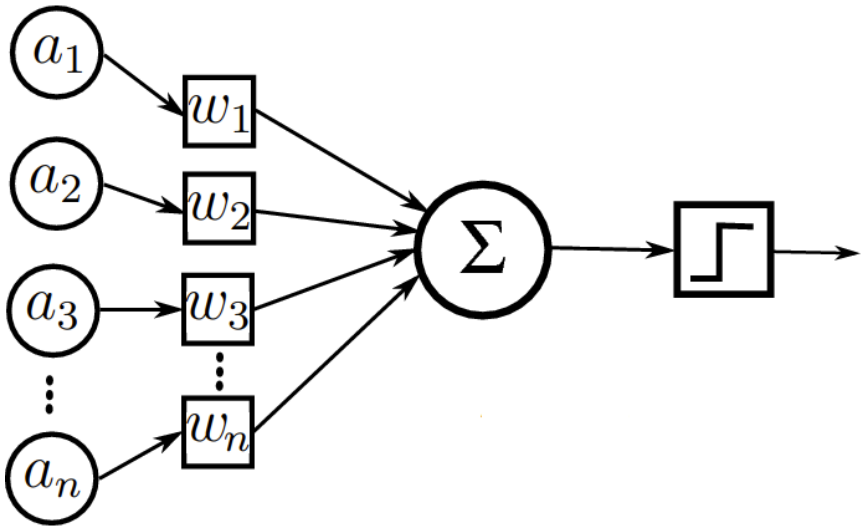


Figure 3.1: Perceptron model of Frank Rosenblatt

$$\text{output} = \begin{cases} 0 & \left(\sum_{i=1}^{n} a_i w_i\right) + b < 0 \\ 1 & \text{otherwise} \end{cases} \tag{3.1}$$

The perceptron is still the basic building block of modern artificial neural networks. However, there are some minor modifications. For example, in modern artificial neural networks, the step function is swapped for a different increasing function. In addition, in modern neural networks, multiple layers of multiple perceptrons are used. The details of more modern neural networks are explained in the following sections.

3.1 Fully Connected Layers

Usually, a fully connected neural network is illustrated as (artificial) neurons with connections between them. Every neuron is basically the perceptron from Equation (3.1), but the step function might be swapped for a different function. In Figure 3.2, an example of such a model of a neural network is depicted. A neural network consists of an input layer, hidden layers, and an output layer. A hidden layer is a layer in between the input layer and the output layer. This neural network has only one hidden layer. However, in practice, a neural network can have more than 100 hidden layers.

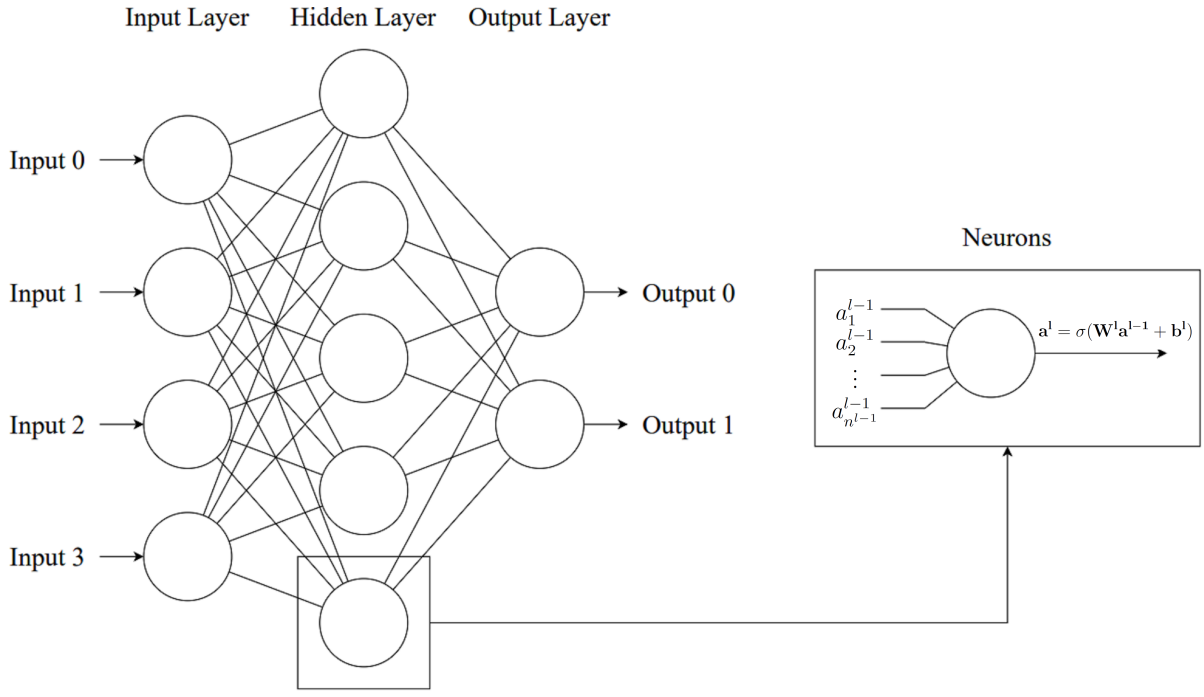


Figure 3.2: Illustration of a fully connected neural network

Let $L \in \mathbb{N}$ be the number of layers of the neural network. For $l \in \{0, ..., L\}$, we have the following equation:

$$\mathbf{a^l} = \sigma(\mathbf{W^l a^{l-1}} + \mathbf{b^l}) \tag{3.2}$$

$\mathbf{a^l}$ is a vector representing the input or output of the layer, so $\mathbf{a}^0$ is the input vector of the neural network, and $\mathbf{a}^L$ is the output of the neural network. Let $n^l \in \mathbb{N}$ be the number of neurons in layer $l \in \{0, ..., L\}$. $\mathbf{W^l}$ is a $n^l \times n^{l-1}$ matrix, and the values in the matrix are called weights. $\mathbf{b^l}$ is a vector of length n^l, and the values in this vector are called biases. The weights and biases will be estimated during the training phase of the neural network. The starting values for the weights and biases are randomly selected or copied from an existing trained neural network. $\sigma()$ is called the activation function. The activation function is a non-linear function with the purpose of creating a non-linearity in the system. Without this non-linearity, a multi-layer neural network would be linear and equivalent to a one-layer neural network. In order to stack layers and create more complexity in the network, non-linearities are needed. Usually, a rectified linear unit (ReLU) is used for the activation function. In Equation (3.3), the definition of a ReLU from $\mathbb{R} \to \mathbb{R}$ is given.

$$\sigma(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{3.3}$$

For convenience, we assume that the derivative of the ReLU at 0 is equal to 0. In Equation (3.4), the definition of the ReLU from $\mathbb{R}^n \to \mathbb{R}^n$ is given.

$$\sigma(\mathbf{x}) = \begin{bmatrix} \sigma(x_1) \\ \vdots \\ \sigma(x_n) \end{bmatrix} \tag{3.4}$$

Another popular activation function is the sigmoid function. Contrary to the ReLU, the output of the sigmoid function is a number between 0 and 1. For this reason, a sigmoid function is usually used in the last layer of the neural network to ensure the outputs are numbers between 0 and 1. In Equation (3.5), an expression for the sigmoid is given.

$$\sigma_i = \frac{1}{1 + e^{-x_i}} \tag{3.5}$$

Another interesting activation function is the softmax function. The output of the softmax function will be a non-negative vector that sums to 1, so the output can be seen as a probability distribution. For this reason, a softmax function is usually used in the last layer to ensure the neural network's output can be seen as a probability distribution. In

Equation (3.6), the expression for the softmax function is given.

$$\sigma_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{3.6}$$

$i \in \mathbb{N}$ is the class index.

3.2 Cost Function

The cost function is a measure of the error the neural network makes. In regression, the sum of squared errors can be used as a cost function. Let K be the number of output variables, let $\hat{\mathbf{y}} = [\hat{y_1}, ..., \hat{y_K}]^T$ be the output vector of the neural network, and let $\mathbf{Y} = [y_1, ..., y_K]^T$ be the actual value. The sum of squared errors is then defined as:

$$C(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} (y_k - \hat{y}_k)^2 \tag{3.7}$$

Another common loss function is the cross-entropy loss. This loss function is usually used for classification problems. In (Simard, Steinkraus, & Platt, 2003), they show that for classification problems, a cross-entropy loss leads to faster training and improved generalization compared to a sum-of-squared error loss.

A categorical cross-entropy loss is usually used in categorical classification problems, where only one class can be true simultaneously. For instance, take the MNIST dataset (Deng, 2012), which contains images of single-digit handwritten numbers. Every image is part of a particular class $(0, 1, \cdots, 9)$, but an image is never part of two classes because every image only contains one number. For the cross-entropy loss, it is necessary that the sum of $y_1, ..., y_k$ and $\hat{y_1}, ..., \hat{y_k}$ is equal to one and that they are positive real numbers. In this case, $\mathbf{y}$ will be one-hot encoded. The activation function for the last layer is usually a softmax function to ensure that the output is non-negative and that the sum of $\hat{y_1}, ..., \hat{y_k}$ equals one. The expression for the cross-entropy loss is given in Equation (3.8).

$$C(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} y_k \log \hat{y_k} \tag{3.8}$$

A binary cross-entropy loss can be used for binary classification problems where more than one class can be true simultaneously or if there is only one class. For instance, a dataset might contain pictures of cats and dogs, and the goal is to determine if a cat and/or a dog is in the image. Some images might not have any cats or dogs, and others might have a cat and a dog. Therefore, an image can be part of two classes simultaneously

or not be part of any class. In Equation (3.9), an expression for the binary cross entropy loss is given. In this case, the activation function for the last layer is normally a sigmoid function to ensure the neural network outputs are between 0 and 1. Unlike the regular cross-entropy loss in Equation (3.8), the sum of $y_1, ..., y_k \in \{0, 1\}$ or $\hat{y_1}, ..., \hat{y_k} \in (0, 1)$ doesn't have to be equal to one. This makes sense because multiple classes can be true simultaneously. For instance, an image could contain a dog and a cat. Every $\hat{y_i}$ could be seen as a separate probability for that given class to be true.

$$C(\hat{\mathbf{y}}, \mathbf{y}) = -\left(\sum_{k=1}^{K} y_k \log \hat{y_k} + (1 - y_k) \log(1 - \hat{y_k}) \right) \tag{3.9}$$

3.3 Backpropagation

Backpropagation was proposed in 1974 (Werbos & John, 1974). It speeds up the training phase of neural networks. Backpropagation is an efficient way of calculating all the gradients of the cost function with respect to the weights and biases. In other words, we want to know $\frac{\partial C}{\partial w_{ij}^l}$ and $\frac{\partial C}{\partial b_j^l}$ for every bias and weight. We can just calculate $\frac{\partial C}{\partial w_{ij}^l}$ and $\frac{\partial C}{\partial b_j^l}$ for each weight and bias separately, but this will result in a lot of repetitive calculations, which is inefficient. Backpropagation is an algorithm to calculate all the weights and biases without repetitive calculations. For backpropagation, the cost function has to fulfill two assumptions:

1. The cost function can be written in the following form: $C(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{k=1}^{K} C(\hat{y}_k, y_k)$
2. The cost function should be a function of the outputs of the neural network and the labels only.

This explanation of backpropagation is taken from (Nielsen, 2019). Define:

$$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l \tag{3.10}$$

Define for neuron $j \in \mathbb{N}$ and layer $l \in \{1, \cdots, L\}$:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{3.11}$$

Calculating δ_j^L yields using $\frac{\partial a_k^L}{\partial z_j^L} = 0$ for $j \neq k$:

$$\delta_j^L = \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) \tag{3.12}$$

Therefore:

$$\boldsymbol{\delta}^L = \begin{bmatrix} \frac{\partial C}{\partial a_1^L}\sigma'(z_1^L) \\ \vdots \\ \frac{\partial C}{\partial a_{n^L}^L}\sigma'(z_{n^L}^L) \end{bmatrix} = \nabla_{\mathbf{a}^L} C \odot \sigma'(\mathbf{z}^L) \tag{3.13}$$

Where $\odot$ is the Hadamard product (elementwise multiplication). Writing $\boldsymbol{\delta}^l$ in terms of $\boldsymbol{\delta}^{l-1}$ using the chain rule yields:

$$\begin{aligned} \delta_j^l = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \\ \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \\ \sum_k \delta_k^{l+1} \frac{\partial}{\partial z_j^l} \big(\sum_i w_{ki}^{l+1} a_i^{l+1} + b_k^{l+1} \big) = \\ \sum_k \delta_k^{l+1} \frac{\partial}{z_j^l} \big(\sum_i w_{ki}^{l+1} \sigma(z_i^l) + b_k^{l+1} \big) = \\ \sum_k \delta_k^{l+1} w_{kj}^{l+1} \sigma'(z_j^l) \end{aligned} \tag{3.14}$$

Therefore:

$$\boldsymbol{\delta}^l = \begin{bmatrix} \delta_1^l \\ \vdots \\ \delta_{n^l}^l \end{bmatrix} = ((\mathbf{W}^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot \sigma'(\mathbf{z}^l) \tag{3.15}$$

With Equation (3.13) and Equation (3.15) we can calculate δ^l for all $l \in \{0, \cdots, L\}$ of the neural network. Writing $\frac{\partial C}{\partial w_{ij}^l}$ and $\frac{\partial C}{\partial b_j^l}$ in terms of δ_j^l using the chain rule yields:

$$\frac{\partial C}{\partial w_{ij}^l} = \sum_k \frac{\partial C}{\partial z_k^l} \frac{\partial z_k^l}{\partial w_{ij}^l} = \frac{\partial C}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l} = \delta_i^l a_j^{l-1} \tag{3.16}$$

$$\frac{\partial C}{\partial b_j^l} = \sum_k \frac{\partial C}{\partial z_k^l} \frac{\partial z_k^l}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l \tag{3.17}$$

Using Equation (3.16) and Equation (3.17), we can calculate all the gradients of the cost function with respect to the weights and biases. The complete backpropagation algorithm is given in Algorithm 3.

Algorithm 3 Backpropagation

for $l = 1 \cdots L$ **do**
  Calculate $\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l$ and $\mathbf{a}^l = \sigma(\mathbf{z}^l)$
end for
Calculate $\delta^L = \nabla_{\mathbf{a}^L} C \odot \sigma'(\mathbf{z}^L)$
for $l = L-1, L-2 \cdots 1$ **do**
  Calculate $\delta^l = ((\mathbf{W^{l+1}})^T \delta^{\mathbf{l+1}}) \odot \sigma'(\mathbf{z^l})$
end for
The gradients can now be calculated with $\frac{\partial C}{\partial w^l_{ij}} = \delta^l_i a^{l-1}_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$

It is called backpropagation because $\delta^1 \cdots \delta^L$ are calculated backwards (first δ^L then δ^{L-1} and so on).

3.4 Optimization algorithm

The goal of the optimization algorithm is to minimize the cost function. In this paragraph, we will explain the most common optimization algorithms. Let $\hat{\mathbf{y}}_\mathbf{1}, \cdots, \hat{\mathbf{y}}_\mathbf{N}$ be the outputs of the neural network. Let $\mathbf{y_1}, \cdots, \mathbf{y_N}$ be the corresponding labels. The neural network should learn to predict the labels. N is the number of samples in the dataset.

3.4.1 Gradient Descent

One iteration of gradient descent is defined by:

$$\mathbf{W} = \mathbf{W} - \frac{\eta}{N} \sum_{n=1}^{N} \nabla_{\mathbf{W}} C(\hat{\mathbf{y}}_\mathbf{n}, \mathbf{y_n}) \tag{3.18}$$

$$\mathbf{b} = \mathbf{b} - \frac{\eta}{N} \sum_{n=1}^{N} \nabla_{\mathbf{b}} C(\hat{\mathbf{y}}_\mathbf{n}, \mathbf{y_n}) \tag{3.19}$$

The downside of plain gradient descent is that the gradient of the cost function has to be calculated for every sample in the entire dataset for each iteration. This is computationally very expensive and will use a lot of memory. Therefore, plain gradient descent is not feasible in large deep learning projects.

Stochastic Gradient Descent and Mini-batch Gradient Descent

Stochastic gradient descent aims to be computationally less expensive by calculating the gradient of the cost function only for a single sample of the dataset. The algorithm is

defined in Algorithm 4. One epoch means iterating one time over the whole dataset, so epochs in Algorithm 4 is the number of iterations over the entire dataset.

Algorithm 4 Stochastic Gradient Descent

Initialize the weights, biases, and learning rate η
for j = 1 **to** j = epochs **do**
  Shuffle the dataset
  for i = 1 **to** i = N **do**
    $\mathbf{W} = \mathbf{W} - \eta \nabla_{\mathbf{W}} C(\hat{\mathbf{y}}_\mathbf{i}, \mathbf{y}_\mathbf{i})$
    $\mathbf{b} := \mathbf{b} - \eta \nabla_{\mathbf{b}} C(\hat{\mathbf{y}}_\mathbf{i}, \mathbf{y}_\mathbf{i})$
  end for
end for

A compromise between plain gradient descent and stochastic gradient descent would be mini-batch gradient descent. Mini-batch gradient descent will calculate the gradient based on a batch of the dataset instead of one sample. The algorithm is defined in algorithm 5. In most papers, when they mention stochastic gradient descent, they actually mean mini-batch gradient descent. Mostly, this is easy to understand because when they define a batch size, it becomes clear they mean mini-batch gradient descent.

Algorithm 5 Mini-batch Gradient Descent

Initialize the weights, biases, the learning rate η and the batch size p.
for j = 1 **to** j = epochs **do**
  Shuffle the dataset
  Divide the dataset into M batches of data.
  for i = 1 **to** i = M **do**
    $\mathbf{W} = \mathbf{W} - \frac{\eta}{p} \sum_{k=1}^{k=p} \nabla_{\mathbf{W}} C(\hat{\mathbf{y}}_\mathbf{i}^\mathbf{k}, \mathbf{y}_\mathbf{i}^\mathbf{k})$
    $\mathbf{b} = \mathbf{b} - \frac{\eta}{p} \sum_{k=1}^{k=p} \nabla_{\mathbf{b}} C(\hat{\mathbf{y}}_\mathbf{i}^\mathbf{k}, \mathbf{y}_\mathbf{i}^\mathbf{k})$
  end for
end for

3.4.2 Momentum

Momentum is used to speed up convergence by adding a fraction of the previous update vector. If the previous update was in the same direction as the current update, momentum will cause acceleration in that direction. In Equation (3.20), the expression for one update of mini-batch gradient descent with momentum is given. $\gamma \in [0, 1]$ is a parameter that has

to be set depending on the amount of preferred momentum. The default value for γ is 0.9.

$$\begin{cases} \mathbf{v_t} = \gamma \mathbf{v_{t-1}} - \frac{\eta}{p} \sum_{k=1}^{k=p} \nabla_{\mathbf{W}} C(\hat{\mathbf{y}}_\mathbf{i}^\mathbf{k}, \mathbf{y}_\mathbf{i}^\mathbf{k}) \\ \mathbf{W} = \mathbf{W} + \mathbf{v_t} \end{cases} \tag{3.20}$$

Momentum will make sure the step size will accelerate if the directions of the gradients of the previous and current steps agree.

3.4.3 RMSprop

RMSProp is an unpublished optimizer that Geoffrey Hinton has introduced in his lecture slides (Hinton, 2012). With RMSprop, the gradient is divided by the square root of the moving averages of the squared gradients. In Equation (3.21), the expression for one update of mini-batch gradient descent with RMSprop is given. The hyperparameter $\beta \in (0, 1)$ has to be set. The standard value for β is 0.9. The idea of RMSprop is to decrease the step size in directions where the gradient oscillates too much.

$$\begin{cases} g_t = \frac{1}{p} \sum_{k=1}^{k=p} \nabla_{\mathbf{W}} C(\hat{\mathbf{y}}_\mathbf{i}^\mathbf{k}, \mathbf{y}_\mathbf{i}^\mathbf{k}) \\ v_t = \beta v_{t-1} + (1 - \beta) g_t^2 \\ \mathbf{W} = \mathbf{W} - \frac{\eta}{\sqrt{v_t}} g_t \end{cases} \tag{3.21}$$

3.4.4 Adam Optimizer

Adam optimizer is inspired by momentum and RMSprop (Kingma & Ba, 2014), but it is not a generalization of the two optimizers. The name Adam is derived from adaptive moments. In (Kingma & Ba, 2014), they show that Adam converges faster than other optimizers like RMSprop. For this reason, this optimizer will be used for all the experiments in this report (see Section 5.4 for the full motivation). In Equation (3.24), the algorithm Adam is given. p is the batch size. m_t is the term that is inspired by the momentum optimizer. v_t is the term that is inspired by RMSprop. The default setting for β_1 is 0.9, and the default setting for β_2 is 0.999. $\hat{m}_t$ and $\hat{v}_t$ are m_t and v_t with bias correction applied. A bias correction is needed since $m_0 = v_0 = 0$. In Equation (3.22) and Equation (3.23), the bias correction terms are derived.

$$
\begin{aligned}
\mathbb{E}[m_t] = & \\
\beta_1 \mathbb{E}[m_{t-1}] + (1-\beta_1)\mathbb{E}[g_t] = & \\
\beta_1^2 \mathbb{E}[m_{t-2}] + \beta_1(1-\beta_1)\mathbb{E}[g_t] + (1-\beta_1)\mathbb{E}[g_t] = & \\
\beta_1^2 \mathbb{E}[m_{t-2}] + (1-\beta_1^2)\mathbb{E}[g_t] = & \\
\beta_1^3 \mathbb{E}[m_{t-3}] + (1-\beta_1^3)\mathbb{E}[g_t] = & \\
\beta_1^t \mathbb{E}[m_0] + (1-\beta_1^t)\mathbb{E}[g_t] = & \\
(1-\beta_1^t)\mathbb{E}[g_t] & \\
\implies \hat{m}_t = \frac{m_t}{1-\beta_1^t} &
\end{aligned}
\tag{3.22}
$$

$$
\begin{aligned}
\mathbb{E}[v_t] = & \\
\beta_2 \mathbb{E}[m_{t-1}] + (1-\beta_2)\mathbb{E}[g_t^2] = & \\
\beta_2^2 \mathbb{E}[m_{t-2}] + \beta_2(1-\beta_2)\mathbb{E}[g_t^2] + (1-\beta_2)\mathbb{E}[g_t^2] = & \\
\beta_2^2 \mathbb{E}[m_{t-2}] + (1-\beta_2^2)\mathbb{E}[g_t^2] = & \\
\beta_2^3 \mathbb{E}[m_{t-3}] + (1-\beta_2^3)\mathbb{E}[g_t^2] = & \\
\beta_2^t \mathbb{E}[m_0] + (1-\beta_2^t)\mathbb{E}[g_t^2] = & \\
(1-\beta_2^t)\mathbb{E}[g_t^2] & \\
\implies \hat{v}_t = \frac{v_t}{1-\beta_2^t} &
\end{aligned}
\tag{3.23}
$$

When t becomes large, the effect of the bias correction will be negligible since $\beta_1^t \to 0$ and $\beta_2^t \to 0$ when $t \to \infty$. In the final equation, we update $\mathbf{W}_t$. η is the learning rate. The default setting for the learning rate is 0.001. ϵ is a small hyperparameter to prevent division by zero. The default setting for ϵ is 10^{-8}.

$$
\begin{cases}
g_t = \frac{1}{p} \sum_{k=1}^{k=p} \nabla_{\mathbf{W}} C(\hat{\mathbf{y}}_{\mathbf{i}}^{\mathbf{k}}, \mathbf{y}_{\mathbf{i}}^{\mathbf{k}}) \\
\mathrm{m}_0 = v_0 = 0 \\
m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t = \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t = \frac{v_t}{1-\beta_2^t} \\
\mathbf{W}_t = \mathbf{W}_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}
\end{cases}
\tag{3.24}
$$

3.5 Convolutional Layer

The first convolutional neural network was proposed in 1980 (Fukushima, 1980). Nowadays, convolutional neural networks (CNN) are really common. For convolutional layers, Equation (3.2) is replaced by Equation (3.25).

$$\mathbf{a}_c^l = \sigma\left(\sum_{\mathbf{k}\in\mathbf{K}_l^c, d\in\mathbb{N}, d\leq C_l} \mathbf{a}_d^{l-1} * \mathbf{k} + \mathbf{b}_c^l\right) \tag{3.25}$$

$\mathbf{a}_c^l \in \mathbb{R}^2$ where $c \in \mathbb{N}$ is the channel and $l \in \mathbb{N}$ is the layer. C_l is the number of channels in layer l. For instance, for RGB images, $C_0 = 3$, since there are three input channels (red, green, and blue). $\mathbf{b}_c^l \in \mathbb{R}^2$ is the bias for layer l and channel c. In most implementations of convolutional layers, there is no bias. $\mathbf{K}_l^c$ is the set of all kernels for output channel c in layer l. Kernels are sometimes also called filters. One kernel is a $n \cdot n$ matrix with $n \in \mathbb{N}$. n is usually chosen to be 3, 5 or 7. The number of kernels in layer l equals $C_{l-1} \cdot C_l$. The size of $\mathbf{a}_c^l$ is usually called the feature map size. A feature map is the output of one kernel applied to the previous layer's output.

In Equation (3.25), $*$ is a (strided) convolutional operator. In Equation (3.26), the expression for a (strided) convolution is given. s is called the stride, which is the step size; hence the word "strided" in the name. The stride is used for downsampling if needed. Downsampling means decreasing the size of the feature map.

$$\mathbf{a} * \mathbf{k}(i,j) = \sum_m \sum_n \mathbf{a}(m,n) \cdot \mathbf{k}(m - s \cdot i, n - s \cdot j) \tag{3.26}$$

In Figure 3.3, the operation of the strided convolutional operator is illustrated. The stride is two, and the kernel size is $3 \cdot 3$. During this operation, the feature map decreases from $7 \cdot 7$ to $3 \cdot 3$.

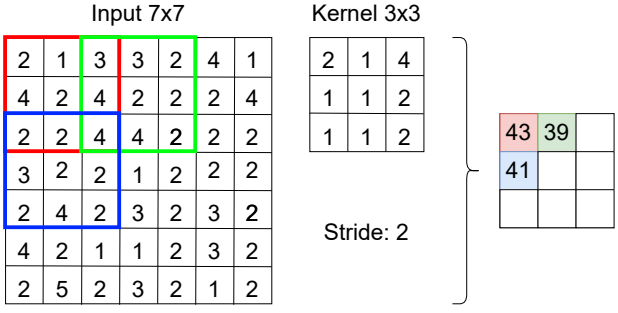
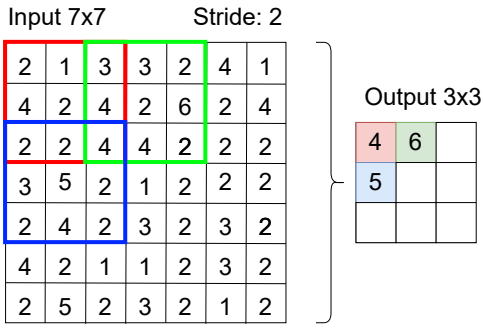


Figure 3.3: Illustration of the operation of a convolution operator in deep learning. For instance, for red, we have $2 \cdot 2 + 1 \cdot 1 + 3 \cdot 4 + 4 \cdot 1 + 2 \cdot 1 + 4 \cdot 2 + 2 \cdot 1 + 2 \cdot 1 + 4 \cdot 2 = 43$. During this operation, the feature map size decreases from $7 \cdot 7$ to $3 \cdot 3$.

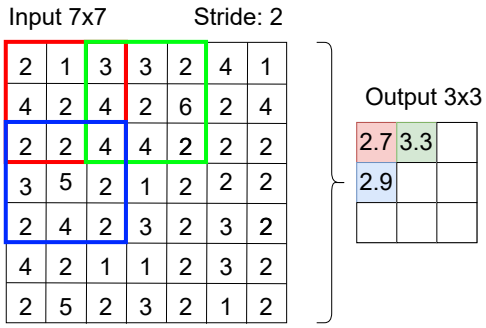
In order to prevent the image from shrinking, padding is usually used. Zero padding is the most common method. For instance, if the image is $7 \cdot 7$ and the kernel size is $3 \cdot 3$, the images are padded with zeros on each side to make the image $9 \cdot 9$. Therefore, the output becomes $7 \cdot 7$ after a convolutional layer with stride one.

3.6 Pooling Layers

Pooling layers can be used to decrease the size of the feature map. Commonly used pooling layers are average pooling and max-pooling. Pooling layers have the same sliding window approach as in Figure 3.3, but instead of elementwise multiplying with a kernel, we take the average for average pooling or the maximum for maximum pooling. For pooling, a stride and a pool size have to be specified. In Figure 3.4a, maximum pooling is illustrated, and in Figure 3.4b, average pooling is illustrated. The stride is two, and the pool size is $3 \cdot 3$. This is done for every channel of the image. Therefore, the number of channels will stay the same, but the feature map size will decrease.



(a) Illustration of max-pooling. During this operation, the feature map size decreases from $7 \cdot 7$ to $3 \cdot 3$.



(b) Illustration of Average-pooling. During this operation, the feature map size decreases from $7 \cdot 7$ to $3 \cdot 3$.

Figure 3.4: Illustration of Max-pooling and Average-pooling

3.7 Residual Neural Networks

ResNET (He et al., 2015) is a state-of-the-art neural network that won the ILSVRC 2015 in scene classification, object detection, and object localization. In addition, ResNET won the MS COCO 2015 challenge in object detection and segmentation. Therefore, ResNET has shown better performance compared to other neural networks. This is the main reason that ResNET will be used in the experiments of this report (see Section 5.1 for a full motivation). ResNET is a modified version of a convolutional neural network where they added "shortcut connections". The last layer of ResNET is fully connected. One building block of this network is illustrated in Figure 3.5.

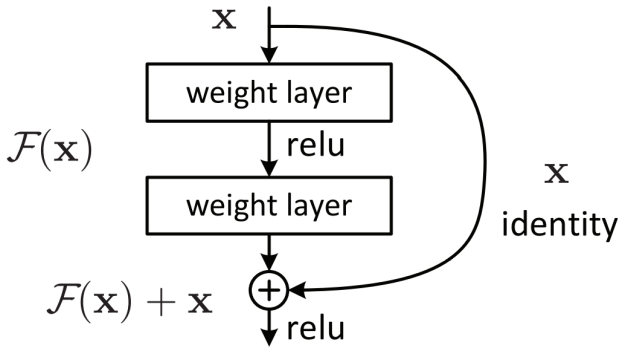


Figure 3.5: (He et al., 2015) ResNET Building Block

This neural network is designed to avoid the degradation problem, which means that stacking more layers will eventually lead to lower accuracy. In (He et al., 2015), they show in several experiments that ResNET is less prone to the degradation problem. In addition, this neural network solves the so-called vanishing gradient problem. The vanishing gradient problem means that the gradient in some nodes becomes very small, preventing the node from changing its value. The "shortcut connection" in Figure 3.5 transfers the gradient into the next layer.

Some architectures used to train on 224x224 RGB images from ImageNet are illustrated in Figure 3.6. In Figure 3.6 (left), the VGG-19 architecture is illustrated. Note that the computational complexity of this network is high because of the extra fully connected layers at the bottom and because of the higher number of channels, while the feature map size is also larger. In Figure 3.6 (middle), the "plain" neural network is illustrated, which is the architecture used as a baseline to test the effectiveness of the extra "shortcut connections". In Figure 3.6 (right), the "residual" neural network is illustrated, which has the extra "shortcut connections".

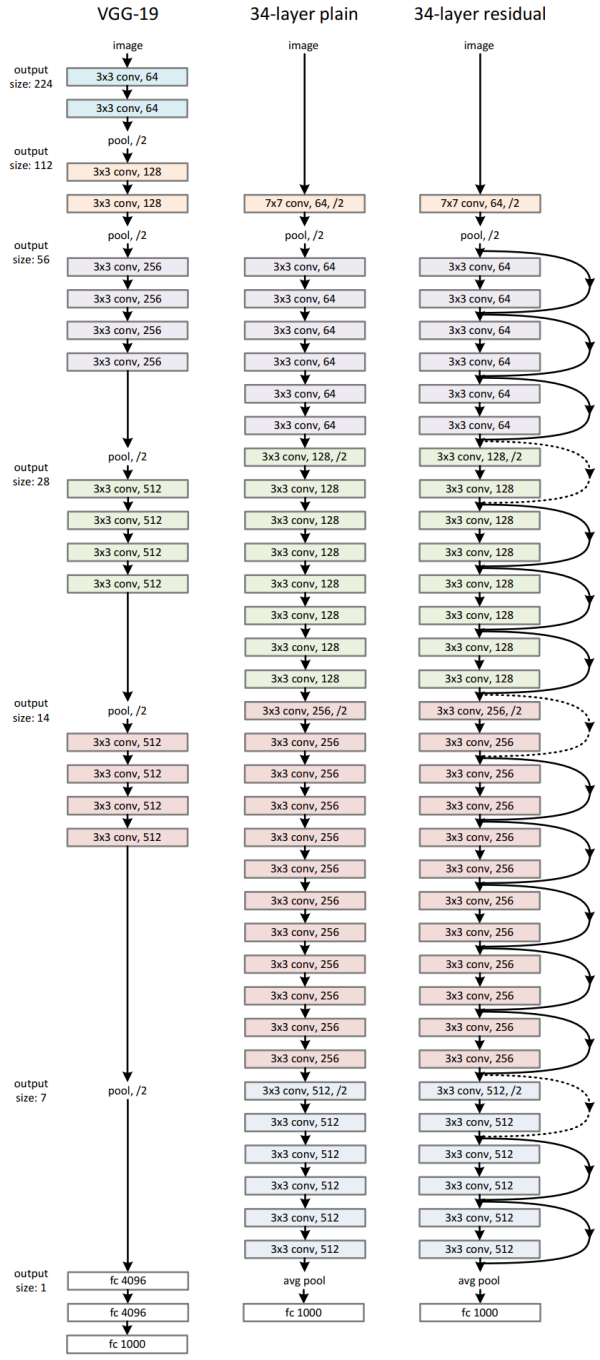


Figure 3.6: (He et al., 2015) left: VGG-19 architecture (19.6 billion FLOPS); middle: 34-layer "plain" (3.6 billion FLOPS); right: 34-layer "residual" (3.6 billion FLOPS). A block with "7x7 conv, 64" means a convolutional layer with kernel size 7x7, 64 channels, and stride 1. "/2" means that the stride is 2, which will half both dimensions of the feature map. The feature map size is the size of $\mathbf{a}_c^l$ in Equation (3.25). The dotted line indicates that both feature map dimensions are halved using a convolutional layer with stride 2 and kernel size 1x1. On the left of the figure, one dimension of the feature map size is given. For instance, "output size 28" means the feature map size is 28x28. For every convolutional layer, zero padding is used to prevent the feature map from shrinking. After every convolutional layer, batch normalization is applied.

The ResNET architecture is partly inspired by the VGG-16 architecture used in (Simonyan & Zisserman, 2014). The VGG-16 architecture and ResNET are both based on the following design rules:

- Most convolutional layers have filters of size 3x3.
- If the number of channels doubles, both feature map dimensions will be halved.
- If the number of channels remains the same, the feature map size will remain the same.

Some differences between the VGG architecture and ResNET are:

- ResNET uses batch normalization after each convolution, which VGG lacks (Ioffe & Szegedy, 2015).
- VGG-16 uses a higher channel count at each feature map size; for example, when the feature map size is 56, the number of channels for VGG-16 is 256, while the number of channels for ResNET is 64. The lower channel count for ResNET decreases the complexity of the network.
- VGG-16 uses more fully connected layers at the output of the neural network compared to ResNET. Less fully connected layers at the output will lead to lower complexity.
- ResNET uses "shortcut" connections.

In (He et al., 2015), they show that ResNET performs better than the VGG architecture.

In (He et al., 2015), a 20-layer, 32-layer, 44-layer, 56-layer, 110-layer, and a 1,202-layer version of the "plain" and "residual" neural networks are trained on the CIFAR-10 dataset. The CIFAR-10 dataset consists of images of size 32x32 in different classification categories. The architecture has the same design rules as the architectures from Figure 3.6. See (He et al., 2015) for the exact architecture. In (He et al., 2015), they show that the "residual" neural network is less prone to the degradation problem since stacking more layers does lead to better accuracy in contrast to the "plain" neural network. However, the accuracy reduces slightly if the number of layers in the "residual" neural network is 1,202. The results for the CIFAR-10 dataset are illustrated in Figure 3.7.

In (He et al., 2015), they also experimented using the ImageNet classification dataset. However, ImageNet contains larger images (224x224 after cropping) compared to the images in the CIFAR dataset, which increases the computational complexity. In order to

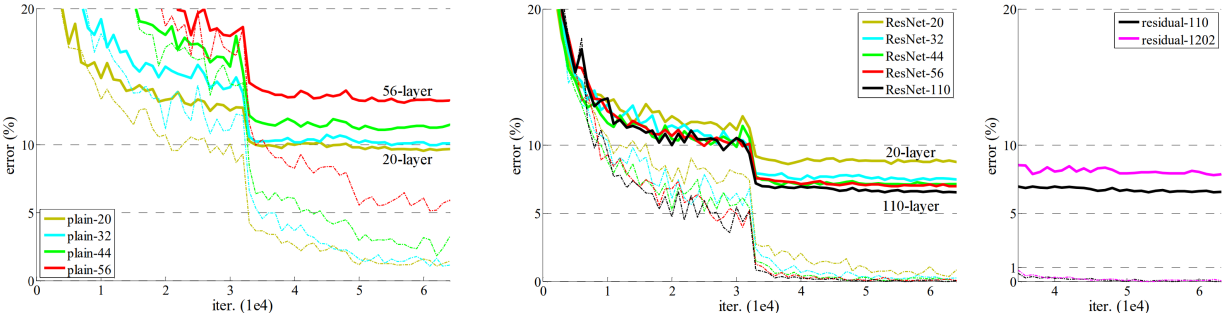


Figure 3.7: (He et al., 2015) Results CIFAR dataset

reduce the computational complexity of the experiments using ImageNet, they use a different building block for the residual neural network when the number of layers is greater than or equal to 50. They call this the bottleneck building block, which is illustrated in Figure 3.8. The idea is that the first 1x1 convolutional layer reduces the number of channels while the second 1x1 convolutional layer increases the number of channels to the original number. The experiments using ImageNet did get similar results as in Figure 3.7.

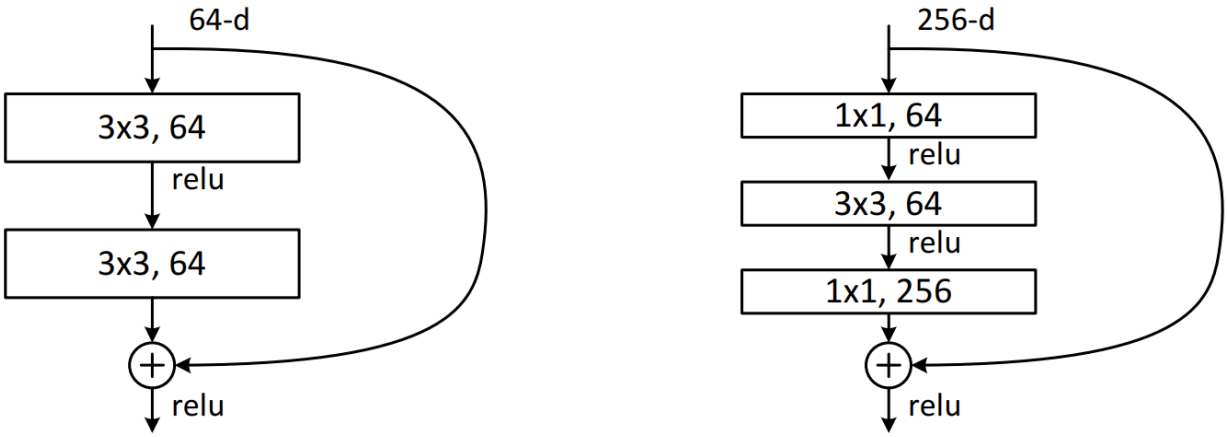


Figure 3.8: (He et al., 2015) left: example of a regular building block for ResNET; right: bottleneck building block

In (He et al., 2015), they used mini-batch gradient descent with batch size 256. They start with a learning rate of 0.1; every time the loss plateaus, the learning rate is divided by 10. They used about 60 iterations in total for each experiment.

Chapter 4

Dataset Creation and Analysis

KNMI provided images taken from the traffic cameras. Every 10 minutes, an image is taken from 319 traffic cameras from May 2017 until August 2019. In addition, KNMI provided data stored in a SQL database. The SQL database consists of the following data:

- Longitude and latitude of 319 of the traffic cameras and of the measurement stations
- Measurement data from the measurement stations, which consists of wind speed, relative humidity, temperature, dew point, and MOR-visibility. The database will return "None" if the corresponding sensor is missing.
- Data from the HARMONIE weather model of KNMI (Bengtsson et al., 2017)
- Image features like saturation, brightness, and smoothness

We are only interested in the MOR-visibility measurements because they are a measure of the amount of fog. In Figure 1.1, the locations of the MOR-visibility (meteorological optical range) sensors are depicted. There are more than 10 million images. In order to label the images efficiently, it would be convenient to label them automatically based on the measurements of the MOR-visibility sensors. When visibility is below 250 meters, it is called dense fog; when visibility is between 250 meters and 1,000 meters, it is called light fog (Wauben & Roth, 2016). KNMI is particularly interested in dense fog. Hence, for the purpose of this project, images with a visibility of less than 250 meters will be called foggy, while images with a visibility of more than 250 meters will be called clear.

4.1 Automatic Labeling Process

In order to label the images, the measurement data taken from the measurement stations and the camera images have to be somehow combined. In (Andrea Pagani & Wauben,

2018), they developed an algorithm to label the images. Let r_{meteo} be a distance in kilometers (km). In (Andrea Pagani & Wauben, 2018), they consider $r_{meteo} = 7.5$ and $r_{meteo} = 2.5$. Every image taken by a camera within a r_{meteo} km radius will be labeled "foggy" if the MOR-visibility sensor measures visibility below 250 meters, and the image will be labeled "clear" otherwise. The images taken from cameras outside the radius are removed from the dataset. In Algorithm 6, the algorithm is given for clarity. In Figure 4.1, the process is illustrated for $r_{meteo} = 7.5$. Near Deelen is a measurement station with a MOR-visibility sensor.

Algorithm 6 Automatic Labeling

```
for every image do
    Find the closest MOR-visibility sensor
    if The distance to the MOR-visibility sensor is less than r_meteo then
        if The visibility measured by the closest MOR-visibility sensor is less than 250
        meters then
            Add the label "foggy" to the image.
        else
            Add the label "clear" to the image.
        end if
    else
        Disregard image
    end if
end for
```

If $r_{meteo} = 2.5$, only 25 cameras are used for the dataset. The other cameras are disregarded since there is no MOR-visibility sensor within a 2.5-kilometer radius. If $r_{meteo} = 7.5$, 127 cameras are used for the dataset. Having only 25 camera locations for the dataset is quite limited. For this reason, $r_{meteo} = 7.5$ is chosen in this project.

The resulting dataset consists of 5,138,393 clear-labeled images and 15,086 foggy-labeled images. Therefore, only 0.29% is labeled "foggy", which means there is a significant imbalance. In addition, the automatic labeling process is prone to labeling errors because fog can be a local phenomenon.

4.2 Dataset Statistics

This section will analyze the number of faulty images and the classification errors. In Figure 4.2, two examples of those faulty images are illustrated. On the other hand, some images are mislabeled. In order to get a clear understanding of the number of images that are faulty or mislabeled, we take a random sample of 200 foggy-labeled images and

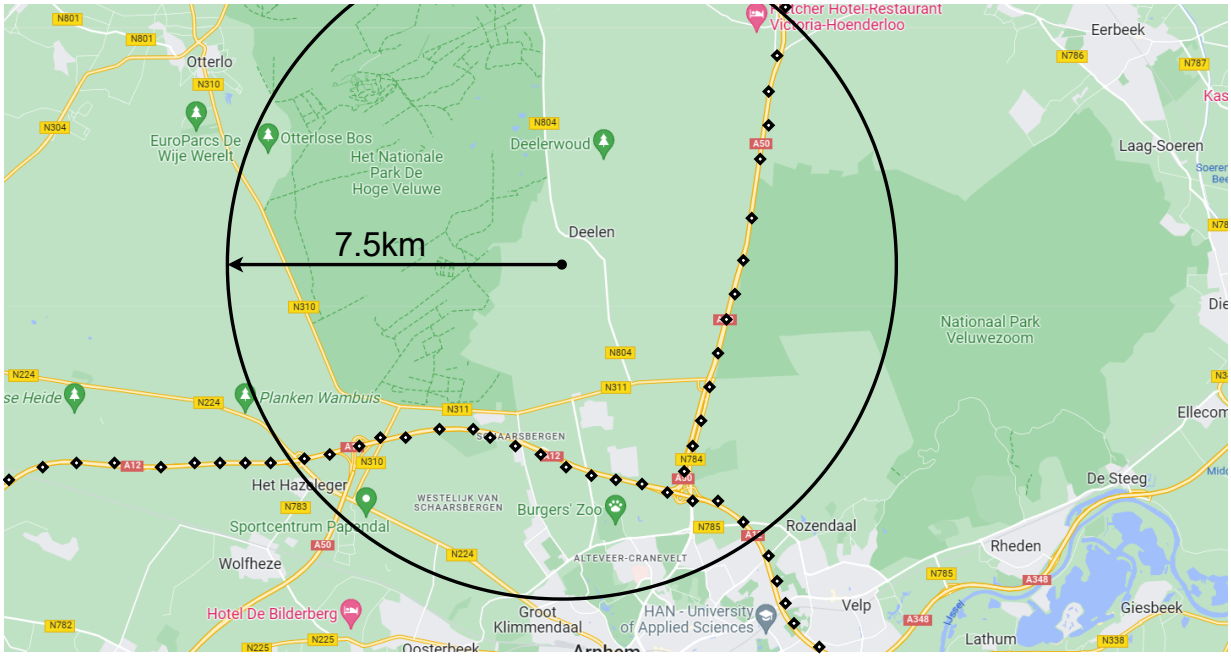
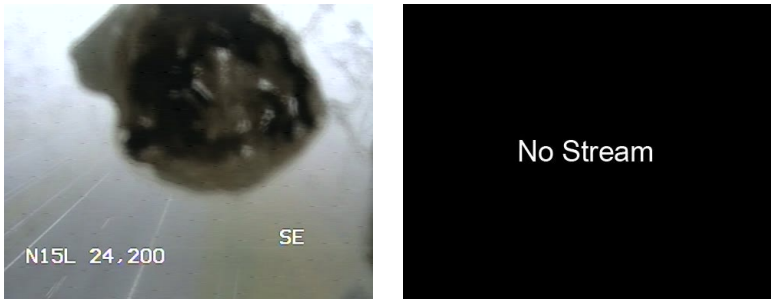


Figure 4.1: Illustration of the automatic labeling process. There is a weather station near Deelen equipped with a MOR-visibility sensor. Every image taken by a camera within a r_{meteo} km radius will be labeled "foggy" if the MOR-visibility sensor measures visibility below 250 meters, and the image will be labeled "clear" otherwise. The images taken by the cameras outside of the radius will be removed from the dataset. The locations of the cameras on this map are for illustrative purposes only. The actual locations of the cameras are different.

a random sample of 200 clear-labeled images. In Table 4.1, the dataset statistics from the random samples are given. A large percentage of the randomly selected foggy-labeled images have incorrect labels. About 4% of the images are black with the text "No Stream" in the middle. The images with the text "No Stream" will be filtered out using a Python script. There aren't any foggy images in the random sample of clear-labeled images. This is expected because the majority of the photos are clear; hence, even if every image were labeled "clear", the label accuracy for the clear-labeled images would still be quite high.

	No Stream	Dirt on lens	Wrong label
Statistics of foggy-labeled images	4%	0%	24%
Statistics of clear-labeled images	3.5%	0%	0%

Table 4.1: Dataset Statistics



(a) Dirt on lens (b) No Stream

Figure 4.2: Examples of faulty images in the Dataset

4.3 Relabeling

Table 4.1 indicates that about 24 percent of foggy-labeled images have incorrect labels. In addition, all of the clear-labeled images from the random sample are correctly labeled. There are about 15,000 foggy-labeled images. In order to increase the label accuracy, the foggy-labeled images are manually relabeled. The resulting dataset will have proper label accuracy. A convenient Python script is used for the relabeling process. The code shows the images one by one. Under the image, there are three buttons named "foggy", "clear", or "cannot say". After examining the image, the user has to choose and press one of the three buttons. The labels are stored in a list and saved to a file. In this way, the images can be relabeled efficiently. The resulting dataset has about 10,000 foggy-labeled images, so about 5,000 foggy-labeled images were wrongly labeled. In Figure 4.3, random foggy-labeled images and random clear-labeled images are shown from the resulting dataset.

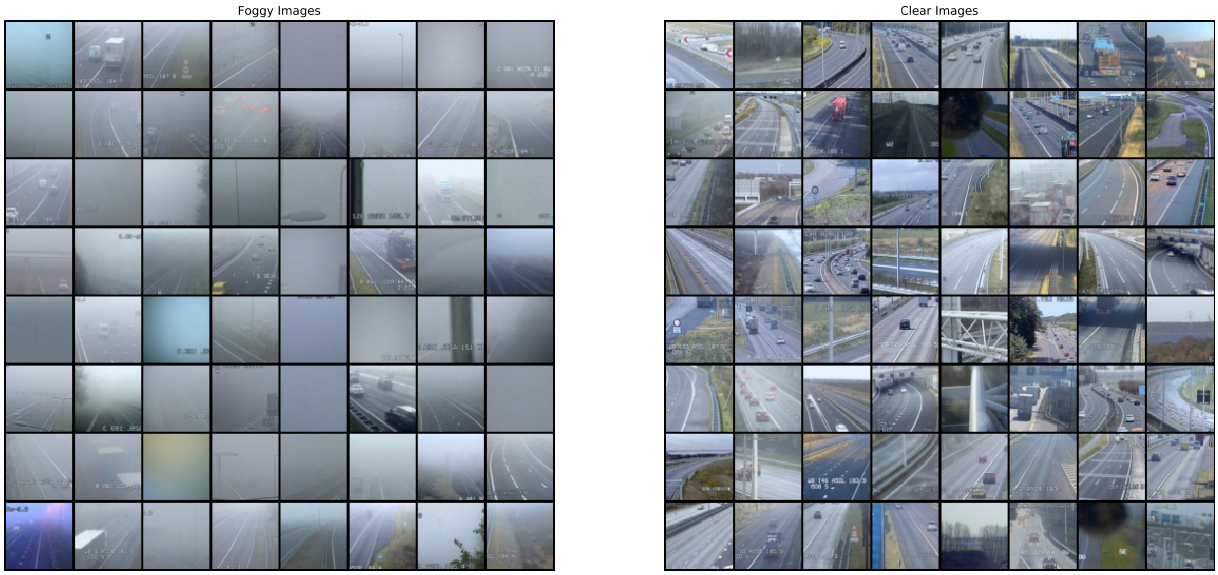


Figure 4.3: Examples of Images in the Resulting Dataset

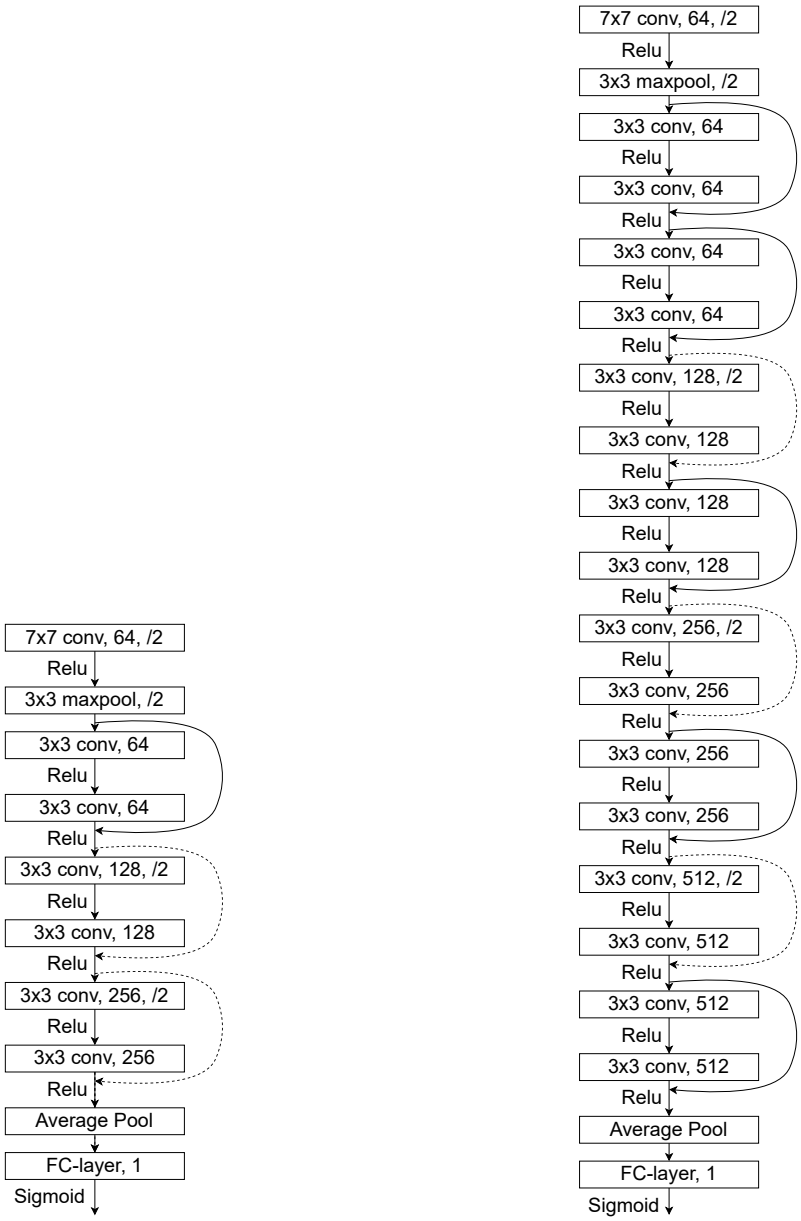
Chapter 5

Experimental Setup

5.1 Models

In the experiments, several models with different configurations will be compared. It will be interesting to compare several configurations of the neural network ResNET (He et al., 2015) because this neural network has proven to deliver the best performance in several challenges (ILSVRC 2015 in scene classification, object detection, and object localization; MS COCO 2015 challenge in object detection and segmentation). In addition, ResNET is less prone to the vanishing gradient problem and the degradation problem (see Section 3.7 or (He et al., 2015)). Compared to other deep learning challenges, the fog detection problem is more straightforward since there is only one class. For instance, ImageNET has 1,000 object classes. To limit the computational burden, the configurations used in this experiment are ResNET18 and ResNET34 (the number in the name is the number of convolutional layers the neural network has). In addition, a 9-layer version of ResNET is introduced called ResNET9 because it will be interesting to see how neural networks with fewer layers perform compared to those with more layers. In Figure 5.1, a block diagram of ResNET9 is illustrated. ResNET9 is ResNET18 with nine layers removed. ResNET34 is illustrated in Figure 3.6. Logistic regression will be implemented to measure how convolutional neural networks compare to simpler models. In short, the following models will be compared in the experiments:

- Logistic Regression
- ResNET9
- ResNET18
- ResNET34



(a) ResNET9 (b) ResNET18

Figure 5.1: Block diagram of ResNET9 and ResNET18. A block with "7x7 conv, 64" means a convolutional layer with kernel size 7x7, 64 channels, and stride 1. "/2" means that the stride is 2, which will half both dimensions of the feature map. The feature map size is the size of $\mathbf{a}_c^l$ in Equation (3.25). The dotted line indicates that both feature map dimensions are halved by using a convolutional layer with stride 2 and kernel size 1x1. For every convolutional layer, we use zero padding to prevent the feature map from shrinking. Note that the last layer is swapped from a fully connected layer with 1,000 neurons to a fully connected layer with one neuron since, in our problem, there is only one class. In addition, the softmax activation function in the last layer is swapped with a sigmoid activation function because there is only one class. After every convolutional layer, batch normalization is applied.

5.2 Preprocessing

Several preprocessing techniques will be applied to increase generalization performance and improve convergence. A good generalization performance means that the model will still perform well on images it hasn't seen before. In other words, the model has bad generalization performance if it shows good performance on the training dataset but bad performance on the test dataset. The images in the dataset are resized from 480x384 to 105x84 before doing the preprocessing operations. The following operations are applied to each image before training:

- Random crop to 80x64
- Random horizontal flip
- Adding Gaussian noise with a mean of 0 and a standard deviation of 0.03. In Figure 5.2, the effect of Gaussian noise is illustrated.

(a) Picture without Gaussian Noise (b) Picture with Gaussian Noise

Figure 5.2: Illustration of the effect of the Gaussian noise that is added

In this way, every training iteration will have slightly different data (note that one iteration processes the whole dataset once). All these operations are methods to increase generalization performance (Shorten & Khoshgoftaar, 2019). In addition, adding noise is a regularization technique (Bishop, 1995). In (Shijie, Ping, Peiyi, & Siping, 2017), it is shown that flipping and cropping increase the performance of image classification tasks.

In order to improve convergence, the data will be normalized before training (LeCun, Bottou, Orr, & Müller, 2012). The normalization method is straightforward. The mean

and the variance are calculated for the complete training dataset, and every image will be pixelwise normalized using Equation (5.1).

$$output = \frac{input - mean}{std} \tag{5.1}$$

5.3 Compensating for dataset imbalance

In order to compensate for the large class imbalance, several techniques will be compared. Firstly, a weighted binary cross-entropy loss will be used. This loss function is also used in previous research about fog detection (Molleman, 2018). In Equation (5.2), an expression is given for the weighted binary cross entropy loss. $y_n \in \{0, 1\}$ is the label, and $x_n \in (0, 1)$ is the model output. In Figure 5.3, the loss is plotted for $w_n = 1$. The weight will be set to $w_n = y_n \cdot \frac{c}{f} + (1 - y_n)$, where c is the number of clear-labeled images and f is the number of foggy-labeled images. This weight is proposed in the documentation of the loss function (Paszke et al., 2022). The loss gives the foggy-labeled images a larger weight to compensate for the class imbalance.

$$C = -\frac{1}{N}\sum_{n=1}^{N} w_n(y_n \log(x_n) + (1 - y_n)\log(1 - x_n)) \tag{5.2}$$

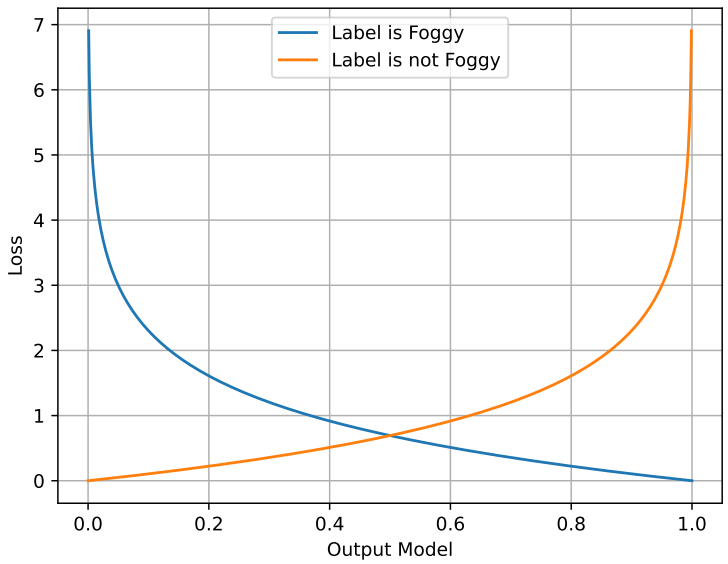


Figure 5.3: Binary Cross Entropy Loss

Secondly, a study on oversampling the foggy-labeled images will be done. The original dataset contains $0,2$ percent foggy-labeled images. The batch size is set to 256; therefore, many batches will have zero foggy-labeled images. The foggy-labeled images will be duplicated such that 5 percent is foggy-labeled. In this way, every batch will have 13 foggy-labeled images on average. Oversampling will be combined with a weighted binary cross-entropy loss to compensate for the remaining imbalance.

From (Buda et al., 2018), we know that oversampling could lead to better results. Therefore, in another experiment, the dataset will be oversampled such that 50 percent is foggy-labeled. This will be combined with a regular binary cross-entropy loss.

The following experiments will be done:

- No oversampling combined with a weighted binary cross-entropy loss
- Oversampling foggy-labeled images such that 5 percent is foggy-labeled combined with a weighted binary cross-entropy loss
- Oversampling foggy-labeled images such that 50 percent is foggy-labeled combined with a regular binary cross-entropy loss

5.4 Optimizer

Adam optimizer is used for all models because it has been demonstrated to have good performance and fast convergence (Kingma & Ba, 2014). Several studies showed that stochastic gradient descent has better generalization performance but slower convergence (Wilson, Roelofs, Stern, Srebro, & Recht, 2017). A good generalization performance means that the model will still perform well on images it hasn't seen before. In other words, the model has bad generalization performance if it shows good performance on the training dataset but bad performance on the test dataset. In (Zhou et al., 2020), they show that stochastic gradient descent will converge to flatter minima compared to adaptive optimizers like Adam. However, to reach these flatter minima, more iterations are needed, which will take a significant amount of time. In addition, since 3-fold cross-validation will be used (see Section 5.5), this effect will only be magnified. Furthermore, several models with several configurations need to be compared, which will further increase training time. For this reason, Adam optimizer is chosen for all the experiments. The learning rate is set to 0.001 and $\beta_1 = 0.9$, $\beta_2 = 0.999$ which are the default settings. The batch size is set to 256, which is the same batch size used in (He et al., 2015). A smaller batch size might lead to slower training, but a larger batch size requires more memory.

5.5 3-Fold Cross-Validation

With 3-fold cross-validation (James, Witten, Hastie, & Tibshirani, [2013]), the dataset will be divided into three equally-sized disjoint subsets. Let's call the three equally-sized disjoint subsets subset 1, subset 2, and subset 3. 3-fold Cross-Validation consists of three iterations, hence the 3-fold in the name. In the first iteration, subset 1 is used as a validation dataset; in the second iteration, subset 2 is used as a validation dataset; and in the third iteration, subset 3 is used as a validation dataset. The other two sets are used for training. In this way, the entire dataset can be used for validation. In Figure 5.4, this principle is illustrated.

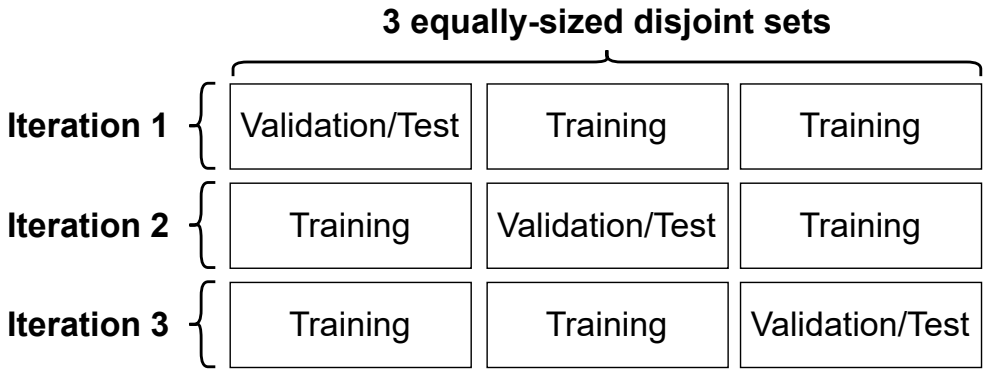


Figure 5.4: Illustration of 3-fold cross-validation

In the following experiments, a slight modification is made to this algorithm. In the validation stage, the validation dataset is split into two equally-sized disjoint sets. Firstly, the first set is used for the validation and fitting of hyperparameters, and the second set is used for testing the model with the hyperparameters. Secondly, the second set is used for the validation and fitting of hyperparameters, and the first set is used for testing the model with the hyperparameters. In this way, the entire dataset is used for testing the model after all iterations.

The dataset consists of 127 different cameras in various locations in the Netherlands. In order to make a split, the camera locations will be divided between the three subdatasets. This is to ensure the validation/testing iteration can detect if the model doesn't generalize well over different camera locations. This is significant because the Netherlands is home to roughly 5,000 traffic cameras, and this dataset only contains a small percentage of those.

In Figure 5.5, the number of foggy-labeled images is counted for every camera location and plotted in a histogram. Note that 46 camera locations have barely any foggy-labeled images in them. Therefore, doing a random split might result in one subdataset having a lot of foggy-labeled images while another subdataset has barely any foggy-labeled images. In order to prevent that, the camera locations are ranked by the number of foggy-labeled

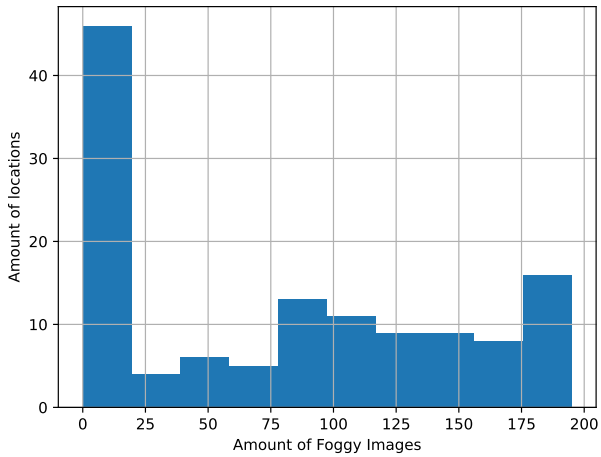
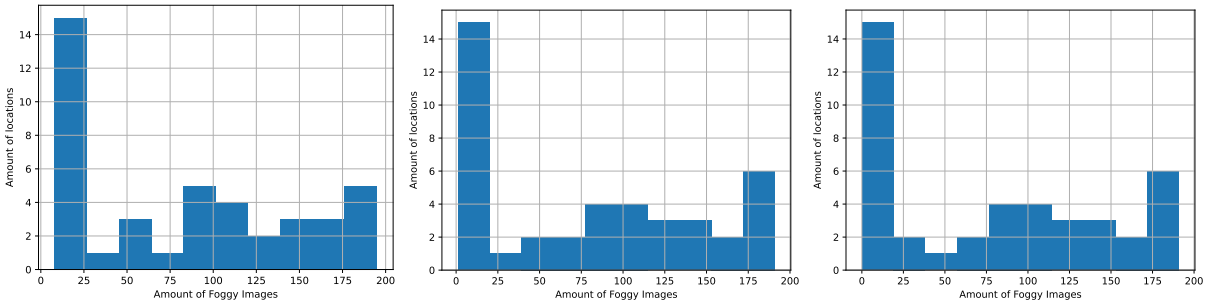


Figure 5.5: Distribution of foggy-labeled images between the different camera locations. For instance, the first bin indicates that about 46 camera locations have fewer than 19 foggy-labeled images.

images in descending order. The first camera location in this ranked list goes to Set 1, the second camera location in this ranked list goes to Set 2, the third camera location in this ranked list goes to Set 3, the fourth location in this ranked list goes to Set 1, and so on. In this way, every set will get approximately the same number of foggy-labeled images. In Figure 5.6, the number of foggy-labeled images for every camera location in these sets is plotted in histograms. Note that the histograms in Figure 5.6 look similar to the histogram in Figure 5.5, which is preferable since every set is a better representation of the entire dataset. For the test/validation stage, the subset has to be split again into a test and validation set. This split is done in the same way.



(a) Set 1 (b) Set 2 (c) Set 3

Figure 5.6: Subset distributions of foggy-labeled images between the different camera locations. For instance, in the leftmost graph, the first bin indicates that about 15 camera locations have fewer than 19 foggy-labeled images.

5.6 Verification Measures

A commonly used tool to show the performance of a model is the confusion matrix. A confusion matrix is illustrated in Table 5.1. The confusion matrix consists of four values from which many verification measures are derived. The false negatives are sometimes called the type I error, and the false positives are sometimes called the type II error. This is terminology from statistical hypothesis testing.

		Predicted Label	
		0	1
True Label	0	True Negatives (TN)	False Positives (FP)
	1	False Negatives (FN)	True Positives (TP)

Table 5.1: Confusion Matrix illustration

Accuracy is likely the most well-known performance indicator. In Equation (5.3), an expression for the accuracy is given. The accuracy isn't working well if the dataset is heavily imbalanced. For instance, in our case, labeling all images "clear" would not be preferable, but the accuracy would be 0.998.

$$\text{Accuracy} = \frac{\text{Number of correctly predicted examples}}{\text{Number of examples}} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.3}$$

Several basic classification metrics can be derived from the values in Table 5.1. The expressions of these metrics are given in Table 5.2.

Metric/Name	Expression
Recall, True Positive Rate (TPR) or Sensitivity	$\frac{TP}{TP+FN}$
Specificity or True Negative Rate (TNR)	$\frac{TN}{TN+FP}$
Precision or Positive Prediction Rate (PPR)	$\frac{TP}{TP+FP}$
Negative Prediction Rate (NPR)	$\frac{TN}{TN+FN}$

Table 5.2: Basic Classification Metrics

Recall shows the performance of the images that are foggy-labeled. Not detecting fog could result in dangerous traffic situations if traffic control decides to rely on this model. For this reason, having a low recall is essential. Specificity shows the performance of clear-labeled images. Precision shows the fraction of correctly predicted images, given that the

model predicts the image as foggy. Precision is highly dependent on the imbalance of the dataset. For instance, precision will decrease if the dataset consists mainly of negative instances and vice versa. We have mainly negative examples, which means it might be challenging to get low precision. For completion, the negative prediction rate (NPR) is included in Table 5.2. The NPR shows the fraction of correctly predicted images, given that the model predicts the image as clear. The NPR isn't used often in literature. It is, just like precision, highly dependent on the imbalance of the dataset. If the dataset consists mainly of negative instances, the NPR will increase, and vice versa.

The F_1 score is the harmonic mean of precision and recall (Chinchor, 1992). In Equation (5.4), the F_1 score is defined. Optimizing recall often leads to low precision, but optimizing precision often leads to low recall. The downside of this measure is that it completely ignores the true negatives. In addition, it is not invariant to class swapping.

$$F_1 = 2\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.4}$$

Another interesting metric is the Mathews correlation coefficient (MCC) (Matthews, 1975). This metric uses all the values in the confusion matrix in Table 5.1. In Equation (5.5), an expression for MCC is given.

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}} \tag{5.5}$$

MCC can be derived by calculating the sample Pearson's correlation coefficient (Pearson, 1895) between the labels and the predictions. Let $y_1, \cdots y_N \in \{0, 1\}$ be all the predictions. Let $\hat{y}_1, \cdots \hat{y}_N \in \{0, 1\}$ be the corresponding labels. In Equation (5.6), the sample Pearson's correlation coefficient is given.

$$r = \frac{\sum_{n=1}^{N}(y_n - \bar{y})(\hat{y}_n - \bar{\hat{y}})}{\sqrt{\left(\sum_{n=1}^{N}(y_n - \bar{y})^2\right)\left(\sum_{n=1}^{N}(\hat{y}_n - \bar{\hat{y}})^2\right)}} \tag{5.6}$$

Where $\bar{y} = \frac{1}{N}\sum_{n=1}^{N} y_n$ and $\bar{\hat{y}} = \frac{1}{N}\sum_{n=1}^{N} \hat{y}_n$. Note that:

$$\sum_{n=1}^{N} y_n \cdot \hat{y}_n = TP \tag{5.7}$$

$$\sum_{n=1}^{N} y_n = TP + FP \tag{5.8}$$

$$\sum_{n=1}^{N} \hat{y}_n = TP + FN \tag{5.9}$$

$$\bar{y} = \frac{1}{N}(TP + FP) \tag{5.10}$$

$$\bar{\hat{y}} = \frac{1}{N}(TP + FN) \tag{5.11}$$

$$y_n^2 = y_n \tag{5.12}$$

$$\hat{y}_n^2 = \hat{y}_n \tag{5.13}$$

$$N = TP + FP + TN + FN \tag{5.14}$$

Rewriting the numerator of Equation (5.6) using Equations (5.7) to (5.14) yields:

$$\sum_{n=1}^{N}(y_n - \bar{y})(\hat{y}_n - \bar{\hat{y}}) = \tag{5.15}$$

$$\sum_{n=1}^{N} y_n\hat{y}_n - y_n\bar{\hat{y}} - \bar{y}\hat{y}_n + \bar{y}\bar{\hat{y}} =$$

$$TP - (TP + FP)\frac{1}{N}(TP + FN) - (TP + FN)\frac{1}{N}(TP + FP) +$$

$$\frac{1}{N}(TP + FP)(TP + FN) =$$

$$TP - \frac{1}{N}(TP + FP)(TP + FN) =$$

$$\frac{(TP + FP + TN + FN) \cdot TP - (TP + FP)(TP + FN)}{TP + FP + TN + FN} =$$

$$\frac{TP^2 + TP \cdot FP + TP \cdot TN + TP \cdot FN - TP^2 - TP \cdot FN - TP \cdot FP - FP \cdot FN}{TP + FP + TN + FN} =$$

$$\frac{TP \cdot TN - FP \cdot FN}{N}$$

Rewriting the number under the square root in Equation (5.6) using Equations (5.7) to (5.14) yields:

$$\begin{aligned}
\Big(\sum_{n=1}^{N}(y_n-\bar{y})^2\Big)\Big(\sum_{n=1}^{N}(\hat{y}_n-\bar{\hat{y}})^2\Big) = \\
\Big(\sum_{n=1}^{N}(y_n-2y_n\bar{y}+\bar{y}^2)\Big)\Big(\sum_{n=1}^{N}(\hat{y}_n-2\hat{y}_n\bar{\hat{y}}+\bar{\hat{y}}^2)\Big) = \\
\Big(TP+FP-2(TP+FP)^2\frac{1}{N}+\frac{1}{N}(TP+FP)^2\Big)\cdot \\
\Big(TP+FN-2\frac{1}{N}(TP+FN)^2+\frac{1}{N}(TP+FN)^2\Big) = \\
\Big(TP+FP-\frac{1}{N}(TP+FP)^2\Big)\Big(TP+FN-\frac{1}{N}(TP+FN)^2\Big) = \\
\frac{1}{N^2}\Big((TP+FP)(TP+FP+TN+FN)-(TP+FP)^2\Big)\cdot \\
\Big((TP+FN)(TP+FP+TN+FN)-(TP+FN)^2\Big) = \\
\frac{1}{N^2}(TP+FP)(TN+FN)(TP+FN)(TP+FP)
\end{aligned} \tag{5.16}$$

Substituting Equation (5.15) and Equation (5.16) in Equation (5.6) will give Equation (5.5). MCC will return a value between -1 and 1. 1 means that the model didn't make any errors. -1 means the model does the exact opposite, so it miss-classifies every image. 0 is the same as random guessing.

In (Chicco & Jurman, 2020), MCC is compared to the F_1 score, and the conclusion is that the MCC is superior. They argue that the F_1 score can lead to misleading results in cases of class imbalance. According to this research, MCC solves this issue since it is invariant to class swapping. They argue that MCC is independent of the ratio of positive and negative cases in the overall dataset. However, this seems to be incorrect. In Equation (5.17), we multiply all positive cases in Equation (5.5) with some $m \in \mathbb{N}$. However, the end result is still dependent on m; therefore, MCC is dependent on the ratio of the positive and negative cases.

$$\begin{aligned}
\frac{m\cdot TP\cdot TN - FP\cdot m\cdot FN}{\sqrt{(m\cdot TP+FP)\cdot(m\cdot TP+m\cdot FN)\cdot(TN+FP)\cdot(TN+m\cdot FN)}} = \\
\sqrt{m}\frac{TP\cdot TN - FP\cdot FN}{\sqrt{(m\cdot TP+FP)\cdot(TP+FN)\cdot(TN+FP)\cdot(TN+m\cdot FN)}}
\end{aligned} \tag{5.17}$$

In (Chicco, Tötsch, & Jurman, 2021), they compare MCC to balanced accuracy, bookmaker informedness, and markedness. They conclude that MCC is more reliable. In

addition, they conclude MCC is more informative and truthful if the positive and negative classes of the dataset have the same importance in the analysis and if correctly classifying the existing ground truth data instances has the same importance as making correct predictions in the analysis. They show that MCC can be written as Equation (5.18). This shows that MCC only shows high values if all metrics in Table 5.2 are high at the same time.

$$\mathrm{MCC} = \sqrt{\mathrm{TPR} \cdot \mathrm{TNR} \cdot \mathrm{PPR} \cdot \mathrm{NPR}} - \sqrt{(1-\mathrm{TPR}) \cdot (1-\mathrm{TNR}) \cdot (1-\mathrm{PPR}) \cdot (1-\mathrm{NPR})} \tag{5.18}$$

If one or more values in Table 5.2 are close to one, the value under the second square root is (almost) negligible in Equation (5.18). In our case, the NPR is most likely already close to one since it benefits from the class imbalance (TN>>FN in most cases because of the class imbalance). This would mean that MCC can be approximated with Equation (5.19).

$$\mathrm{MCC} \approx \sqrt{\mathrm{TPR} \cdot \mathrm{TNR} \cdot \mathrm{PPR}} \tag{5.19}$$

In (Redondo et al., 2020), they compare MCC to other measures, including the F1-measure. In order to compare the measures, they generate several confusion matrices with different properties (positively imbalanced, negatively imbalanced, balanced, high FN, etc.). They show that MCC performs well in all cases.

A common tool to test the performance is the ROC curve (receiver operating characteristic) (Fawcett, 2006). The area under the ROC curve is called AUC (area under the curve). The method was initially developed for military radar technology, hence its name. Later, the ROC curve was used in many other sciences like psychology, medical science (Swets, Dawes, & Monahan, 2000), and machine learning. The ROC curve is a graph with recall and $1 -$ specificity for every possible threshold. In Figure 5.7, an example of a ROC curve is illustrated. AUC is the area under the orange curve, which is 0.93 in this case. A perfect ROC curve would touch the upper left corner of the graph (TPR $= 1$ and FPR $= 0$). The dotted blue line is the same as random guessing.

AUC and the ROC curve can measure the separability between the two classes. In other words, if there exists a threshold that separates the classes properly, AUC will be high. The downside of the ROC curve is that it is more difficult to interpret compared to, for instance, MCC, which will give a number between -1 and 1. For this reason, AUC is often used to compare models. AUC is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. This is equivalent to the Wilcoxon-Mann-Whitney statistic (Mann & Whitney, 1947). However, AUC has received some criticism (Lobo, Jiménez-Valverde, & Real, 2008). AUC is designed as a decision threshold invariant measure. However, a decision threshold will be chosen, which will be part of the model and influence the performance.

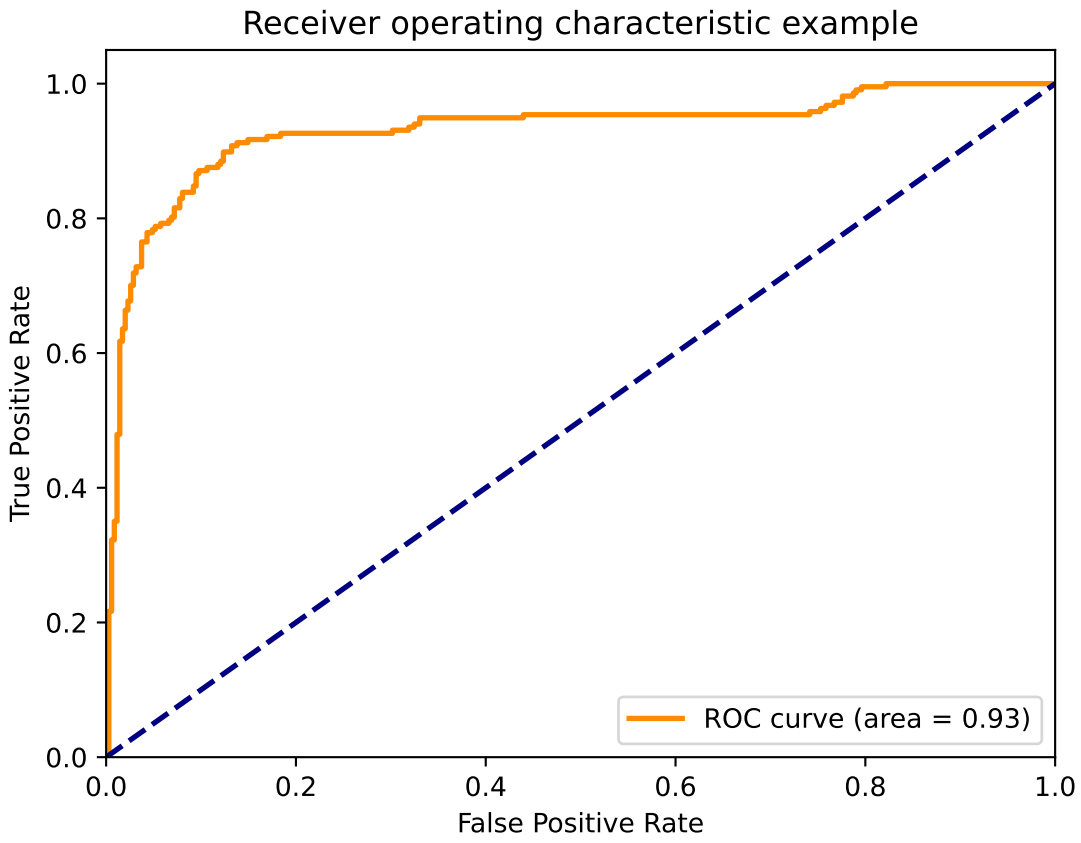


Figure 5.7: Example of a ROC curve for logistic regression. AUC is 0.93 here.

5.6.1 Conclusion

MCC has demonstrated superior performance with imbalanced datasets when compared to other measures (Chicco & Jurman, 2020) (Redondo et al., 2020) (Chicco et al., 2021). One of the main advantages of MCC is that it is unforgivable because it simultaneously requires high recall, specificity, precision, and a high negative prediction rate. A second advantage is that it is invariant to class swapping. The fact that MCC is unforgivable and invariant to class swapping sets it apart from other measures described in Section 5.6.

For this reason, MCC will be used to compare the models. In addition, recall, specificity, and precision will be included to provide more clarity. For instance, if MCC shows a weak correlation, these basic measures can provide more insight into the reason for the weak correlation.

5.7 Decision Threshold Optimization

The model's output is a number between 0 and 1. Larger values reflect the model's increased confidence in the presence of fog and vice versa. In order to make a prediction about fog, a particular decision threshold has to be set. If the model output exceeds the decision threshold, the prediction is foggy, and vice versa. Usually, the decision threshold is set to 0.5. However, it is possible to tweak the decision threshold to a desired value. In Section 5.5, we already took into account being able to set hyperparameters using the validation stage in cross-validation. From various literature, we know that optimizing the decision threshold can improve the results ((Elor & Averbuch-Elor, 2022), (Buda et al., 2018), (Schisterman, Perkins, Liu, & Bondell, 2005)). The main goal will be to maximize MCC. In addition, we don't want recall to become too low. A low recall means that fog will often not be detected. This could result in dangerous traffic situations if traffic control decides to rely on this model. We will try to improve the results by optimizing the decision threshold in Section 6.1.2.

Chapter 6

Experiments

In this chapter, several models will be compared with different configurations as described in Chapter 5. We use 3-fold cross-validation as described in Section 5.5. In the case of decision threshold optimization, the decision threshold hyperparameter is set during the validation stage of the process. The results described in this chapter are the results of the final testing stage. For reference, all confusion matrices for all experiments are given in Appendix B.

The negative prediction rate (NPR) is greater than 0.999 for all experiments in this chapter. Therefore, we can approximate Equation (5.18) with Equation (6.1).

$$\mathrm{MCC} \approx \sqrt{\mathrm{TPR} \cdot \mathrm{TNR} \cdot \mathrm{PPR}} \tag{6.1}$$

Equation (6.1) has almost the same expression as the geometric mean between recall (TPR), specificity (TNR), and precision (PPR). The only difference is that Equation (6.1) contains a square root instead of a cubic root. From Equation (6.1), it is clear that MCC will give equal weight to these three measures. However, these three measures are not of equal importance. For this reason, these three measures will also be evaluated independently in this chapter. As described in Section 5.6, recall is the most important measure, followed by precision and specificity.

If we minimize recall, then we minimize the type I error. The model makes a type I error when it is foggy, but the model predicts that it is clear. Recall can be seen as the probability that the model predicts a foggy-labeled image correctly. Recall is considered important since not detecting fog could result in dangerous traffic situations if traffic control decides to rely on this model.

Specificity and precision are both related to type II errors. In other words, if we minimize precision or specificity, we also minimize type II errors. The model makes a type II error when it is clear, but the prediction of the model is foggy. Specificity can be

seen as the probability that the model correctly predicts a clear-labeled image. Precision can be seen as the probability that the model is correct, given that the prediction of the model is foggy. For instance, if precision is 0.1 and the model prediction is foggy, the probability that the label is actually foggy is 0.1. Precision is directly dependent on the class imbalance and is, for this reason, difficult to optimize (often FP>>TP because of the class imbalance). High precision mostly implies high specificity because, for high precision, FP needs to be low. In addition, TP is a low number because of class imbalance, and TN is mostly a high number because of class imbalance.

6.1 Experiments

Firstly, the goal of the experiments is to compare the performance of different models. What is the optimal number of layers? Will adding more layers result in better performance? Secondly, we want to investigate the effect of oversampling compared to using a weighted binary cross-entropy loss. Oversampling is just duplicating the foggy-labeled images of the dataset before preprocessing is applied. The preprocessing stage is explained in Section 5.2. The preprocessing stage can be seen as a basic data augmentation technique. Oversampling is intriguing since one of the goals of this thesis is to augment the dataset with synthetic data. Therefore, we can compare basic oversampling with augmenting the dataset with synthetic data. In Appendix A, a conceptual algorithm is given to add fog to clear images. Unfortunately, training on synthetic images has not been done due to time limitations. Thirdly, the goal is to investigate the effect of decision threshold optimization. The effect of decision threshold optimization is analyzed in Section 6.1.2.

The weights of the neural networks in this section are initialized with weights from pre-trained neural networks on ImageNet. Note that no pre-trained version of ResNET9 is available because this version of ResNET has been proposed in this thesis. In Section 5.1, the architecture of ResNET9 is given. ResNET9 is basically ResNET18 with nine layers removed. For ResNET9, the pre-trained weights of ResNET18 are taken. Initializing the neural network with weights from a pre-trained neural network is also known as transfer learning (Bozinovski, 2020). In Section 6.2.1, the effect of the pre-trained weights compared to random weights is investigated by an additional experiment. Additionally, the effect of taking the fast Fourier transform (FFT) as an additional preprocessing step is investigated. The results of this experiment are analyzed in Section 6.2.2.

6.1.1 Oversampling vs Weighted Binary Cross Entropy Loss

The first goal of this experiment is to compare the performance of the different models. The second goal is to compare oversampling to a weighted binary cross-entropy loss.

No oversampling

In this experiment, the foggy-labeled images of the dataset are not oversampled. Instead, a weighted binary cross-entropy loss is used. In Equation (5.2), the weighted cross-entropy loss is defined. The weighted cross-entropy loss will compensate for the imbalance. In Table 6.1, the results of these experiments are given. LR stands for logistic regression, and MCC stands for Matthews correlation coefficient. From Table 6.1, we can conclude that logistic regression performs the worst and that ResNET9 performs the best. Adding more layers doesn't result in better performance. In fact, ResNET18 and ResNET34 are performing slightly worse than ResNET9. MCC suffers from low precision, which is why it shows a weak correlation. In Figure 6.1, the loss during training is plotted. The loss for the ResNET models is decreasing steadily. It seems that the loss didn't completely converge after ten iterations. However, the loss appears to be very close to convergence. The loss for ResNET9 is lower than the loss for ResNET18 and ResNET34 on the training dataset. This perfectly reflects the results in Table 6.1.

Setting	Recall	Specificity	Precision	MCC
LR	0.8739	0.7368	0.0066	0.0619
ResNET9	**0.9340**	0.9825	**0.0969**	**0.2978**
ResNET18	0.8676	**0.9829**	0.0923	0.2797
ResNET34	0.9097	0.9819	0.0917	0.2857

Table 6.1: Foggy-labeled images are not oversampled in this experiment. A weighted binary cross-entropy loss is used. Models are compared with basic measures from Table 5.2 and MCC. The highest score has a bold font in each column.

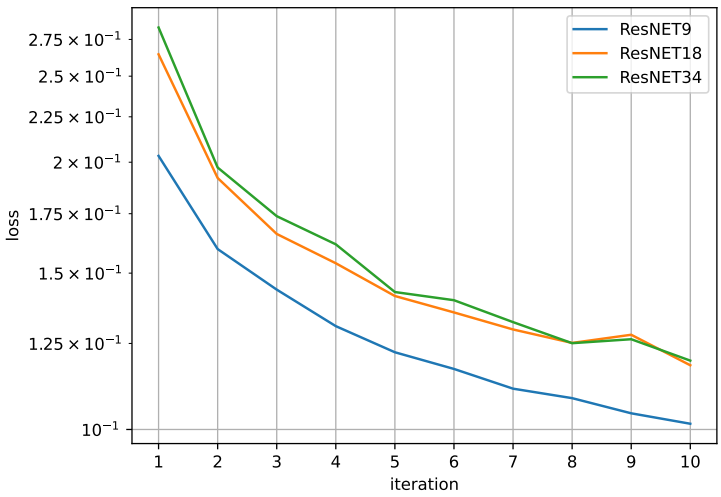


Figure 6.1: Loss during training for ResNET

Foggy-Labeled Images Oversampled to 5 Percent

In this experiment, the minority class of the dataset is oversampled such that 5 percent of the dataset is foggy-labeled. The remaining imbalance is compensated with a weighted binary cross-entropy loss. In Table 6.2, the results of these experiments are given. Logistic regression does slightly worse compared to the previous experiment. However, the ResNET models are doing better than the previous experiment. ResNET9 shows the best performance again, and precision increases to 0.23. ResNET18 and ResNET34 seem to favor recall over precision. Figure 6.2, the loss during training is illustrated. The loss decreases steadily during training. It seems that the loss did not completely converge yet. However, the loss seems to be very close to convergence.

Setting	Recall	Specificity	Precision	MCC
LR	0.8587	0.7196	0.0061	0.0575
ResNET9	0.9045	**0.9941**	**0.2338**	**0.4582**
ResNET18	0.9530	0.9800	0.0875	0.2855
ResNET34	**0.9559**	0.9861	0.1213	0.3379

Table 6.2: Foggy-labeled images are oversampled such that 5 percent is foggy-labeled. A weighted binary cross-entropy loss is used. Models are compared with basic measures from Table 5.2 and MCC. The highest score has a bold font in each column.

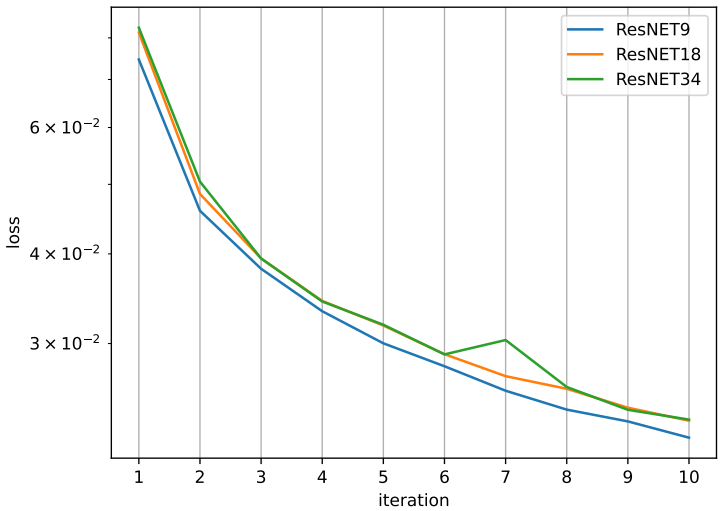


Figure 6.2: Loss during training for ResNET

Foggy-Labeled Images Oversampled to 50 Percent

In this experiment, the minority class of the dataset is oversampled such that 50 percent is foggy-labeled. Since the training dataset is already balanced, a regular binary cross-entropy loss is used in this experiment. In Table 6.3, the results of these experiments are given. Logistic regression in this experiment favors specificity and precision a bit more compared to the previous experiments. Logistic regression still performs worse compared to the models using ResNET. Compared to the previous experiments, the ResNET models favor recall over specificity and precision. In Figure 6.3, the loss during training is depicted. The loss during training for ResNET18 and ResNET34 is consistently lower than the loss for ResNET9, but ResNET18 and ResNET34 perform worse than ResNET9 on the validation dataset. This is a sign of overfitting. The loss did not completely converge yet, but it seems close to convergence.

Setting	Recall	Specificity	Precision	MCC
LR	0.8082	0.7963	0.0079	0.0670
ResNET9	**0.9890**	**0.9479**	**0.0367**	**0.1854**
ResNET18	0.9847	0.8480	0.0128	0.1032
ResNET34	0.9443	0.7461	0.0074	0.0708

Table 6.3: Foggy-labeled images are oversampled such that 50 percent is foggy-labeled. Models are compared with basic measures from Table 5.2 and MCC. The highest score has a bold font in each column.

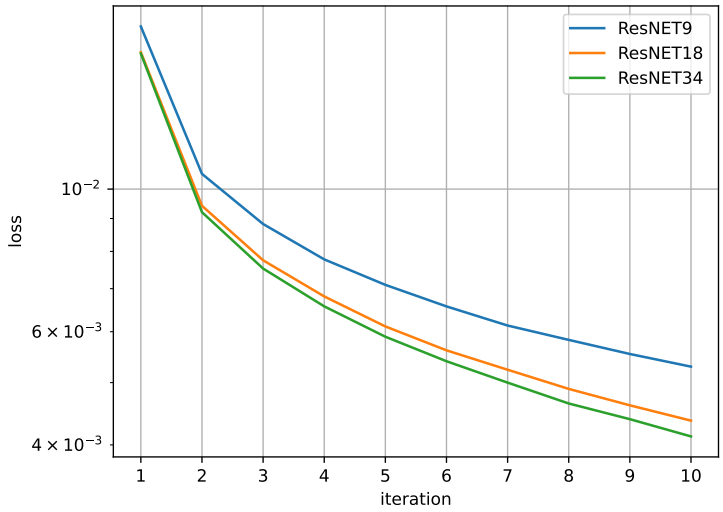


Figure 6.3: Loss during training for ResNET

6.1.2 Decision Threshold Optimization

All the experiments described above use a decision threshold (T) of 0.5. In other words, if the neural network outputs a value larger than 0.5, it will classify the image as foggy, and if the output is smaller or equal to 0.5, it will classify the image as clear. However, it is possible to optimize this decision threshold (T) by choosing a decision threshold (T) in the validation stage. In the experiments, recall and specificity are generally high, but precision is underperforming. As a result, MCC shows a weak correlation. In this section, we are trying to optimize performance by choosing a different decision threshold. One option would be to maximize Equation (6.1). We know that specificity is mostly close to one, and optimizing precision will result in a specificity that is even higher. Therefore, we can approximate MCC with Equation (6.2). This approximation enables us to use the built-in function (sklearn.metrics.precision_recall_curve) in scikit-learn (Pedregosa et al., 2011), which outputs precision and recall for every decision threshold. This makes it easy to implement.

$$\mathrm{MCC} \approx \sqrt{\mathrm{TPR} \cdot \mathrm{PPR}} \tag{6.2}$$

Therefore, the decision threshold that maximizes Equation (6.2) is given by Equation (6.3).

$$\mathrm{T} = \underset{\text{Thresholds}}{\operatorname{argmax}}\, \mathrm{TPR} \cdot \mathrm{PPR} \tag{6.3}$$

Equation (6.2) does give equal weight to precision and recall. However, in this particular problem, recall is more important. For this reason, a second function is proposed for the optimization of the decision threshold. In Equation (6.4), the expression is given for this function. Precision is penalized by taking the square root.

$$\mathrm{T} = \underset{\text{Thresholds}}{\operatorname{argmax}}\, \mathrm{TPR} \cdot \sqrt{\mathrm{PPR}} \tag{6.4}$$

To get a better understanding, precision and recall are plotted for every possible decision threshold for ResNET9 in Figure 6.4. The foggy-labeled images are oversampled such that 5 percent is foggy-labeled. Every dot corresponds to a particular decision threshold. The black dot corresponds to a decision threshold of 0.5; the red dot corresponds to the decision threshold in Equation (6.3), and the yellow dot corresponds to the decision threshold in Equation (6.4). From Figure 6.4, it is clear that Equation (6.4) indeed favors recall more than precision compared to Equation (6.3).

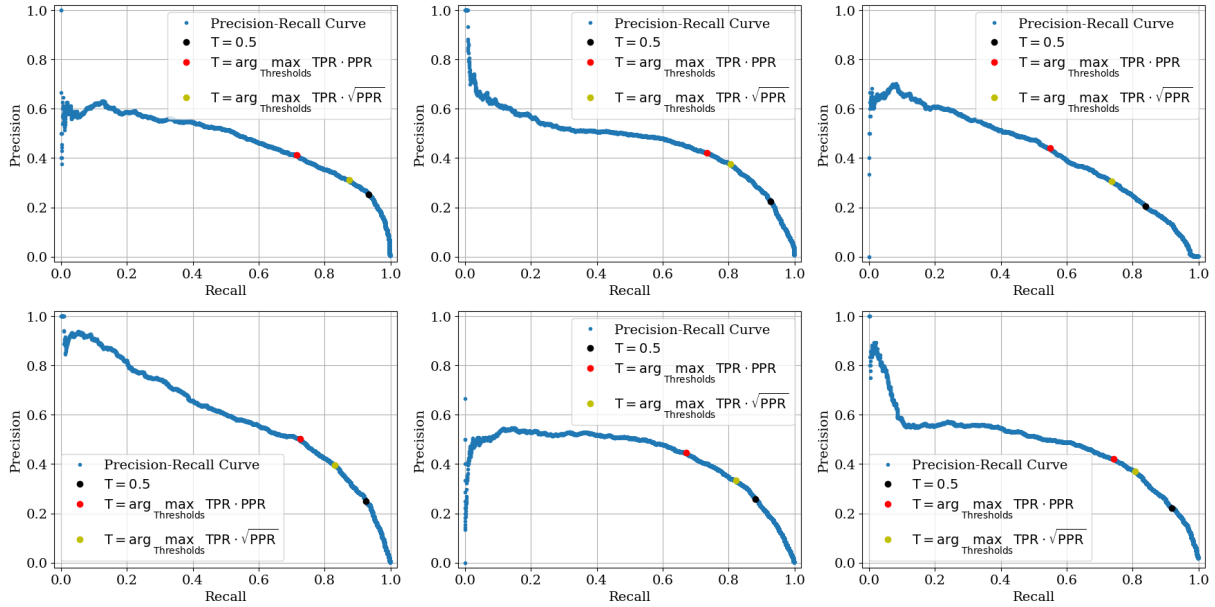


Figure 6.4: Precision-Recall Curve for ResNET9. Foggy-labeled images are oversampled such that 5 percent is foggy-labeled. A weighted binary cross-entropy loss is used. The precision-recall curve plots precision and recall for every possible decision threshold. Every blue dot corresponds to a certain decision threshold. The red dot is the decision threshold that satisfies Equation (6.3). The yellow dot is the decision threshold that satisfies Equation (6.4). There are six graphs because we must fit the decision threshold six times. This is because we use the algorithm described in Section 5.5.

This process is repeated for every model and setting. In Table 6.4, Table 6.5 and Table 6.6 the results for the decision threshold optimization are given. The results from Section 6.1.1 are also included for comparison. Overall, optimizing the decision threshold will increase precision, specificity, and MCC but decrease recall. In most cases, this trade-off will depend on the user's preference. For logistic regression, precision increases slightly after decision threshold optimization but remains very low. For all other models, performance increases substantially. Mostly, MCC shifts from a weak correlation (<0.4) to an average correlation ($0.4<\text{MCC}<0.6$). Precision improves substantially. However, since it is a trade-off, recall decreases. Especially if we use decision threshold optimization using Equation (6.3) (T1). For this reason, optimizing the decision threshold using Equation (6.4) (T2) seems most preferable. A combination between oversampling (till 5%) and a weighted binary cross entropy loss still shows the overall best performance after applying decision threshold optimization. The overall best-performing model is still ResNET9 after optimizing the threshold.

No Oversampling

Setting	Recall	Specificity	Precision	MCC
LR	0.8739	0.7368	0.0066	0.0619
LR with T1	0.7140	0.8427	0.0090	0.0682
LR with T2	0.8716	0.7665	0.0074	0.0673
ResNET9	**0.9340**	0.9825	0.0969	0.2978
ResNET9 with T1	0.6278	**0.9977**	**0.3548**	**0.4706**
ResNET9 with T2	0.7508	0.9961	0.2800	0.4570
ResNET18	0.8676	0.9829	0.0923	0.2797
ResNET18 with T1	0.5424	0.9973	0.2908	0.3956
ResNET18 with T2	0.6747	0.9952	0.2193	0.3827
ResNET34	0.9097	0.9819	0.0917	0.2857
ResNET34 with T1	0.6155	0.9971	0.2968	0.4258
ResNET34 with T2	0.7311	0.9952	0.2344	0.4121

Table 6.4: Foggy-labeled images are not oversampled. Models with decision threshold optimization compared with basic measures from Table 5.2 and MCC. T1 means that the decision threshold is optimized using Equation (6.3). T2 means that the decision threshold is optimized using Equation (6.4). The highest score has a bold font in each column.

Foggy-Labeled Images Oversampled to 5 Percent

Setting	Recall	Specificity	Precision	MCC
LR	0.8587	0.7196	0.0061	0.0575
LR with T1	0.6995	0.8204	0.0078	0.0604
LR with T2	0.8389	0.6977	0.0055	0.0522
ResNET9	0.9045	0.9941	0.2338	0.4582
ResNET9 with T1	0.6770	**0.9982**	**0.4287**	**0.5376**
ResNET9 with T2	0.7925	0.9969	0.3368	0.5153
ResNET18	0.9530	0.9800	0.0875	0.2855
ResNET18 with T1	0.7585	0.9936	0.1928	0.3803
ResNET18 with T2	0.8168	0.9922	0.1737	0.3745
ResNET34	**0.9559**	0.9861	0.1213	0.3379
ResNET34 with T1	0.6944	0.9979	0.4013	0.5267
ResNET34 with T2	0.8079	0.9967	0.3268	0.5125

Table 6.5: Foggy-labeled images are oversampled such that 5 percent is foggy-labeled. Models with decision threshold optimization are compared with basic measures from Table 5.2 and MCC. T1 means that the decision threshold is optimized using Equation (6.3). T2 means that the decision threshold is optimized using Equation (6.4). The highest score has a bold font in each column.

Foggy-Labeled Images Oversampled to 50 Percent

Setting	Recall	Specificity	Precision	MCC
LR	0.8082	0.7963	0.0079	0.0670
LR with T1	0.7504	0.8185	0.0082	0.0658
LR with T2	0.8617	0.7532	0.0070	0.0637
ResNET9	**0.9890**	0.9479	0.0367	0.1854
ResNET9 with T1	0.6414	**0.9969**	**0.2962**	**0.4343**
ResNET9 with T2	0.7660	0.9950	0.2346	0.4221
ResNET18	0.9847	0.8480	0.0128	0.1032
ResNET18 with T1	0.6194	0.9963	0.2540	0.3949
ResNET18 with T2	0.7514	0.9939	0.1978	0.3835
ResNET34	0.9443	0.7461	0.0074	0.0708
ResNET34 with T1	0.5511	0.9964	0.2370	0.3596
ResNET34 with T2	0.6974	0.9928	0.1630	0.3348

Table 6.6: Foggy-labeled images are oversampled such that 50 percent is foggy-labeled. Models with decision threshold optimization are compared with basic measures from Table 5.2 and MCC. T1 means that the decision threshold is optimized using Equation (6.3). T2 means that the decision threshold is optimized using Equation (6.4). The highest score has a bold font in each column.

6.2 Additional Experiments

6.2.1 Transfer Learning

In Section 6.1, the weights of the neural networks are taken from pre-trained neural networks trained on ImageNET. In this way, the pre-trained neural network already knows what a car looks like, for instance. Therefore, the hypothesis is that this could lead to better performance because the neural network has already learned things. This idea is known as transfer learning (Bozinovski, 2020). Remark that ResNET9 was initialized with the weights of ResNET18. This is possible because ResNET9 is basically ResNET18 with nine layers removed.

The goal of this experiment is to investigate the effect of transfer learning. Did initializing the weights of a pre-trained neural network improve performance? To test if the pre-trained weights improve performance, the same experiments have been done with random weights. The results from this experiment are compared to the values in Section 6.1. The experiments where the foggy-labeled images are oversampled to 50 percent are not executed because there was not enough time left within the project. In Table 6.7 and Table 6.8, the results of these experiments are given.

No oversampling

For ResNET9, the results are very similar to those presented in Section 6.1.2. However, for ResNET18 and ResNET34, the results seem to be consistently better than the results presented in Section 6.1.2. Although the difference is small, it might not be random. The loss in Figure 6.5 is generally slightly higher than in Figure 6.1. This might indicate that initializing the weights with pre-trained weights decreases generalization performance. In other words, it increases performance on the training dataset but decreases performance when the model is evaluated on a different dataset.

Setting	Recall	Specificity	Precision	MCC
ResNET9	0.9194	0.9853	0.1115	0.3173
ResNET9 with T1	0.6123	**0.9978**	**0.3547**	**0.4647**
ResNET9 with T2	0.7346	0.9961	0.2758	0.4485
ResNET18	0.8839	0.9869	0.1196	0.3224
ResNET18 with T1	0.5912	0.9973	0.3084	0.4255
ResNET18 with T2	0.7467	0.9948	0.2241	0.4072
ResNET34	**0.9278**	0.9820	0.0937	0.2918
ResNET34 with T1	0.6204	0.9975	0.3296	0.4507
ResNET34 with T2	0.7508	0.9954	0.2452	0.4273

Table 6.7: Foggy-labeled images are not oversampled. The models are initialized with random weights. Models compared with basic measures from Table 5.2 and MCC. T1 means that the decision threshold is optimized using Equation (6.3). T2 means that the decision threshold is optimized using Equation (6.4). The highest score has a bold font in each column.

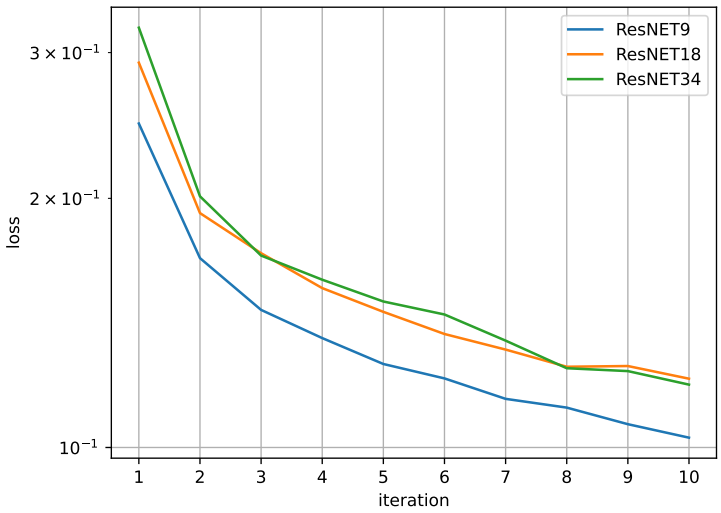


Figure 6.5: Loss during training for all ResNET models

Foggy-Labeled Images Oversampled to 5 Percent

ResNET9 generally scores a bit lower on recall in this experiment but higher on all the other measures. For ResNET18, precision doubles compared to the results in Table 6.5 but recall is slightly lower. Initializing with random weights increases performance for ResNET18 substantially. For ResNET34, the performance seems to have increased slightly. Without decision threshold optimization, precision increased from 0.12 to 0.16, but recall decreased slightly. After threshold optimization, recall is generally higher, but precision is slightly lower. Overall, performance did improve slightly for ResNET34. Overall, initializing with random weights enhances performance.

The fact that initializing the weights with pre-trained weights has a generally negative effect on the validation performance of the model was unexpected. At the same time, the loss in Figure 6.6 is generally slightly higher compared to the loss in Figure 6.2. This might indicate that initializing the weights with pre-trained weights decreases generalization performance. In other words, it increases performance on the training dataset but decreases performance when the model is evaluated on a different dataset.

Setting	Recall	Specificity	Precision	MCC
ResNET9	0.8714	0.9954	0.2768	0.4896
ResNET9 with T1	0.6777	**0.9983**	**0.4417**	**0.5460**
ResNET9 with T2	0.7551	0.9975	0.3811	0.5352
ResNET18	0.9275	0.9930	0.2112	0.4408
ResNET18 with T1	0.7102	0.9980	0.4211	0.5457
ResNET18 with T2	0.8071	0.9970	0.3527	0.5322
ResNET34	**0.9504**	0.9901	0.1622	0.3904
ResNET34 with T1	0.7215	0.9978	0.3923	0.5309
ResNET34 with T2	0.8136	0.9966	0.3217	0.5102

Table 6.8: Foggy-labeled images are oversampled such that 5 percent is foggy-labeled. The models are initialized with random weights. Models compared with basic measures from Table 5.2 and MCC. The highest score has a bold font in each column.

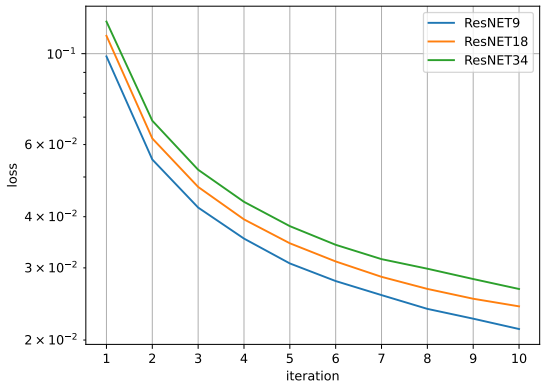


Figure 6.6: Loss during training for all ResNET models

6.2.2 Fast Fourier Transform

This experiment investigates the effect of the fast Fourier transform (FFT). The FFT is calculated for every image in the preprocessing stage. In Table 6.9, Figure 6.7, Table 6.10 and Figure 6.8, the results for this experiment are given. Logistic regression does perform poorly because recall is very low. Intuitively, this is a bad sign because, apparently, a simple model cannot capture the information anymore after doing the FFT. The models using ResNET are doing a lot better. The models seem to favor precision a little bit more over recall compared to the experiments in Section 6.1. However, doing an FFT generally doesn't seem to improve performance. The best-performing model is again ResNET9, with the foggy-labeled images, oversampled such that 5 percent is foggy-labeled. Here, precision is 0.259, but recall is 0.817. Without the FFT, precision would be 0.234, and recall would be 0.905. In some sense, this is a trade-off, but for this particular use case, the numbers without the FFT are better because recall is more important than precision.

No oversampling

Setting	Recall	Specificity	Precision	MCC
LR	0.2621	**0.9979**	0.1996	0.2270
LR with T1	0.5606	0.9931	0.1401	0.2776
LR with T2	0.7138	0.9848	0.0863	0.2447
ResNET9	**0.8597**	0.9896	0.1420	0.3469
ResNET9 with T1	0.5796	0.9973	**0.3009**	**0.4161**
ResNET9 with T2	0.7367	0.9945	0.2116	0.3929
ResNET18	0.7974	0.9879	0.1170	0.3025
ResNET18 with T1	0.5390	0.9963	0.2257	0.3469
ResNET18 with T2	0.6831	0.9931	0.1654	0.3338

Table 6.9: Foggy-labeled images are not oversampled. The FFT is calculated for every image in the preprocessing stage. Models compared with basic measures from Table 5.2 and MCC. The highest score has a bold font in each column.

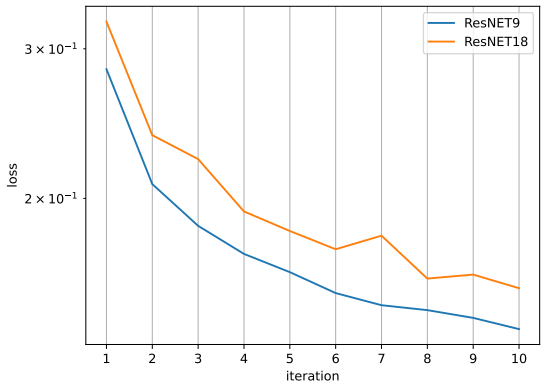


Figure 6.7: Loss during training for ResNET

Foggy-Labeled Images Oversampled to 5 Percent

Setting	Recall	Specificity	Precision	MCC
LR	0.0891	**0.9993**	0.2017	0.1329
LR with T1	0.5244	0.9936	0.1414	0.2698
LR with T2	0.6825	0.9841	0.0795	0.2293
ResNET9	0.8172	0.9953	0.2593	0.4586
ResNET9 with T1	0.6421	0.9980	**0.3964**	**0.5033**
ResNET9 with T2	0.7518	0.9967	0.3148	0.4851
ResNET18	0.8025	0.9941	0.2134	0.4119
ResNET18 with T1	0.5902	0.9978	0.3519	0.4544
ResNET18 with T2	0.7324	0.9959	0.2646	0.4386
ResNET34	**0.8591**	0.9892	0.1377	0.3414
ResNET34 with T1	0.6083	0.9970	0.2897	0.4182
ResNET34 with T2	0.7254	0.9949	0.2210	0.3985

Table 6.10: Foggy-labeled images are oversampled such that 5 percent is foggy-labeled. A weighted binary cross-entropy loss is used. The FFT is calculated for every image in the preprocessing stage. The models are initialized with random weights. Models compared with basic measures from Table 5.2 and MCC. The highest score has a bold font in each column.

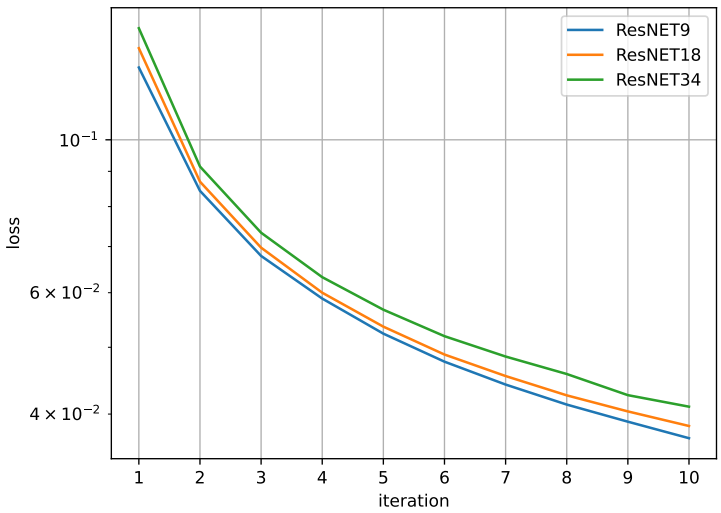


Figure 6.8: Loss during training for ResNET

6.3 Mislabeled images

The dataset is not perfect. As a result, errors in the dataset in conjunction with class imbalance could also be to blame for the resulting low precision. The percentage of mislabeled clear-labeled images should be low because we did not find any mislabeled images in the sample of 200 clear-labeled images in Chapter 4. However, since there are about 5 million clear-labeled images and 10,000 foggy-labeled images, there will be a lot of false positives if a small percentage of the clear-labeled images are foggy. A lot of false positives will result in low precision. In other words, if specificity is, for instance, 0.99, then there are about 50,000 false positives, and precision will be lower than $\frac{10{,}000}{50{,}000+10{,}000} = 0.167$.

For the true negatives, we already know that there are barely any mislabeled images (see Chapter 4). In addition, the foggy-labeled images are labeled by hand and should not have too many mistakes. For this reason, we will limit our investigation to the effect of mislabeled images on false positives in this section.

We do this by carefully checking each image individually and estimating if the visibility is below or above 250 meters. This is done by considering several key facts:

- The length of every line on the road is 3 meters, and the distance between the lines is 9 meters. Therefore, every line accounts for 12 meters. If we can count 20 consecutive lines on the image, then the visibility is definitely above 250 meters. This is because 20 consecutive lines account for 240 meters, and some lines will be out of sight.
- In most locations, the distance between the lampposts is about 50 meters.

If there is doubt, we will compare the image to the location on Google Maps. For some images, the camera is zoomed in quite far, which means it seems more foggy than it actually is. An example of this is given in Figure 6.9.

For this experiment, we will only consider ResNET9 initialized with random weights. The foggy-labeled images are oversampled such that 5 percent of the dataset is foggy-labeled. We use this configuration because it has the best overall performance in the experiments. We consider a decision threshold of 0.5, the decision threshold provided by Equation (6.3) (T1), and the decision threshold provided by Equation (6.4) (T2). It is quite time-consuming to determine if the visibility is higher or lower than 250 meters. Therefore, only 100 images for each decision threshold are checked.

Setting	Percentage incorrect	95% CI
No decision threshold optimization	25	[17.5, 34.3]
Decision threshold optimization (T1)	41	[31.9, 50.8]
Decision threshold optimization (T2)	41	[31.9, 50.8]

Table 6.11: Statistics of mislabeled images in the false positives for ResNET9 from a random sample of 100 images. CI stands for confidence interval. The confidence interval is estimated using the Wilson method (Wallis, 2013).

Figure 6.9: Image with a visibility of more than 250 meters. The camera is zoomed in, making it appear more foggy than it actually is. The matrix boards on the right are 250 meters away from the camera and are clearly visible. Therefore, the visibility is above 250 meters, and the image should be labeled as "clear".

From the estimated percentage and their confidence interval in Table 6.11, we can calculate the performance based on the confidence interval. A confidence interval is used because a random sample of 100 images is quite small and prone to randomness. In Table 6.12, the performance metrics are given based on the confidence interval. All three metrics increased after the correction. The results are substantially better. Precision becomes substantially better. MCC shifts from an average to a strong correlation (>0.6).

Setting	Recall	Specificity	Precision	MCC
ResNET9	[0.9080, 0.9278]	[0.9962, 0.9970]	[0.4033, 0.5248]	[0.6037, 0.6965]
ResNET9 with T1	[0.7468, 0.7754]	[0.9988, 0.9992]	[0.6198, 0.7253]	[0.6795, 0.7492]
ResNET9 with T2	[0.8240, 0.8491]	[0.9983, 0.9988]	[0.5785, 0.6955]	[0.6894, 0.7677]

Table 6.12: Corrected performance confidence intervals for ResNET9.

6.4 Discussion and Conclusion

6.4.1 Weighted binary cross entropy loss vs Oversampling

Oversampling the dataset such that 50 percent of the images are foggy-labeled seems to result in overfitting, which causes worse performance. Only about 6,700 foggy-labeled images are in the training dataset in each cross-validation iteration. In addition, there are only about 82 different locations in the training dataset in each cross-validation iteration. Therefore, the model might remember certain images or characteristics of the location to get high performance on the training dataset. However, only using a weighted cross-entropy loss doesn't seem ideal either. More than half of the batches during training don't contain any foggy-labeled images. In addition, if a batch contains one foggy-labeled image, the loss contribution of that one image will get multiplied by about 500 (since 99.8 percent of the dataset is clear-labeled). Some batches will have two or more foggy-labeled images because the images are randomly picked. Therefore, the loss and the gradient will oscillate from batch to batch. Adam optimizer has the property to decrease the step size in directions where the gradient oscillates too much. This is most likely the reason that no oversampling performs worse compared to oversampling the foggy-labeled images to 5 percent. Oversampling the foggy-labeled images to 5 percent such that every batch has around 13 foggy-labeled images seems to solve this problem. Oversampling the foggy-labeled images to 5 percent in combination with a weighted cross-entropy loss appears to be a nice balance between the two extremes. Another way to solve this problem might be to use a different optimizer, like mini-batch gradient descent.

6.4.2 Transfer Learning

Initializing the weights of a pre-trained neural network to improve performance seems like a good idea. In addition, if it doesn't help, it won't hurt, right? The results of the experiment in Section 6.2.1 indicate otherwise. The performance of the models on the validation dataset is actually slightly better and sometimes even substantially better. In addition, the loss during training is slightly lower. Is this a coincidence? Well, it might not be. A similar phenomenon happens in a comparable experiment. In (Ash & Adams, 2020), three ResNET18 models are compared. The models are trained on the CIFAR-10 dataset. This dataset contains images labeled in ten separate classes. The first model is trained on 50% of the data. The second model is trained on 100% of the data. The third model is initialized with the weights of the first model and trained on 100% of the data. The third model performs worse on the test dataset when compared to the second model. However, the loss during training is the same between the second and third models. They conclude that the hot start decreases the generalizability of the model. In other words, on the training dataset, both models achieve the same performance. However, when the

model is evaluated on a different dataset, like the test or the validation dataset, the model with the hot start performs worse.

6.4.3 Dataset quality

The dataset isn't perfect. This will have a negative effect on the model's performance. However, even worse, it makes validating and testing the model more challenging. Are the errors caused by a bad model, or are they caused by mislabeled images? From Section 6.3, it becomes clear that the imperfections in the dataset have a significant contribution to the results. In other words, low precision is partly caused by mislabeled images in the dataset. The imbalance of the dataset magnifies this effect. If we correct these results, MCC shifts from an average correlation ($0.4<\text{MCC}<0.6$) to a strong correlation ($\text{MCC}>0.6$).

6.4.4 Decision Threshold Optimization

After training, the model can be further tweaked by choosing a decision threshold. The selection of the decision threshold is a trade-off between recall and precision. In our case, precision was underperforming, which resulted in a weak correlation ($\text{MCC}<0.4$). Decision threshold optimization was able to improve precision substantially. In addition, MCC shifts from an average correlation ($0.4<\text{MCC}<0.6$) to a strong correlation. However, the only way to achieve that is to lower recall by about 10 or 20 percent. Therefore, it is a trade-off. However, the resulting model with decision threshold optimization is definitely preferable.

Chapter 7

Conclusions and Recommendations for Future Research

We formulated several research questions in Section 1.1. In this section, we will reflect on these research questions and determine if this thesis succeeded in answering them. In addition, we will make recommendations for future research in this chapter. The first two research questions concerning the dataset are:

- **Research Question 1:** How do we create a properly labeled dataset?
- **Research Question 2:** What is the quality of this dataset?

In Chapter 4, we created a labeled dataset with a low labeling error. The relabeling of the foggy-labeled images did increase the quality of the dataset compared to previous research ((Molleman, 2018), (Andrea Pagani & Wauben, 2018)). However, from Section 6.3, it becomes clear that the labeling errors in the false positives significantly affect the results. Although the percentage of labeling errors in the clear-labeled images is very low ($< 0.1\%$) because the dataset is very imbalanced (0.2%/99.8%), the effect gets magnified. Mainly, precision suffers, which causes MCC to suffer too. Specificity and recall aren't affected because they are class imbalance invariant measures.

One interesting subject for further research is improving the dataset even further. For instance, one way to do this is to relabel the false positives. There are about 10,000 false positives (for ResNET9 with a dataset where 5 percent is foggy-labeled because of oversampling, initialized with random weights and T1). However, it might be possible to take a subset of those 10,000 false positives to limit the amount of work. For instance, the images can be ranked based on model output since the model outputs a number between 0 and 1 (or a number between $-\infty$ and $+\infty$ if the last activation function of the neural network is removed). In some sense, a higher model output means the model is more

certain about an image. Therefore, it is possible to relabel the images where the model is most certain. In this way, the model will improve because the quality of the dataset will increase. In addition, even more importantly, the validation and testing of the model improve. The next two research questions concerning the machine learning models are:

- **Research Question 3:** How do we evaluate the performance of a machine learning model?
- **Research Question 4:** How do several machine learning models and configurations perform on this dataset?

In Section 5.6, we studied various verification measures. The verification measure we chose is MCC. MCC is unforgivable because it simultaneously requires high recall, specificity, precision, and a high negative prediction rate. In addition, it is invariant to class swapping. The fact that MCC is unforgivable and invariant to class swapping sets it apart from other measures described in Section 5.6. MCC enabled us to evaluate the performance of the machine learning models effectively.

After comparing several machine learning models and configurations, we can conclude several things:

- Just oversampling leads to overfitting. Especially for models with more layers (ResNET18, ResNET34).
- Just using a weighted binary cross-entropy loss isn't ideal either. See Section 6.4.1 for an explanation.
- Combining oversampling (till 5%) with a binary cross-entropy loss is a nice balance between the two extremes. This will result in the best performance.
- Initializing weights with pre-trained weights results in worse performance. See Section 6.4.2 for an explanation.
- Threshold optimization does increase performance in all experiments. In addition, it enables tweaking the model according to preferences after training. It is a trade-off between recall and precision. Therefore, implementing threshold optimization is highly recommended.
- ResNET9 performs best in all experiments. Adding more layers results in worse or similar performance. Logistic regression doesn't work very well because the model doesn't seem to be sophisticated enough.
- Using the fast Fourier transform (FFT) in the preprocessing stage generally decreases performance.

Threshold optimization was able to improve precision substantially. Although it will decrease recall, it is a trade-off. The results after threshold optimization are definitely preferable. One question someone might ask is if the same could be achieved differently. In this research, we completely compensated for the class imbalance with a weighted binary cross-entropy loss and/or oversampling. This will put a lot of weight on the foggy-labeled images, which is suitable for achieving high recall, but this might also be part of the reason that precision is underperforming. An idea for further research might be to investigate the effect of decreasing the compensation for the class imbalance.

Not oversampling with a weighted binary cross entropy loss combined with Adam optimizer does seem to underperform in the experiments. The hypothesis is that this could be prevented with a different optimizer like mini-batch gradient descent (Section 6.4.1 for an explanation). An idea for future research is to investigate if this is the case.

Overall, the experiments allowed for selecting the ideal configuration, substantially increasing performance. The best-performing configuration achieved a strong correlation in the Matthews correlation coefficient.

The next two research questions concerning the synthetic data are:

- **Research Question 5:** What are the possibilities for generating synthetic data?
- **Research Question 6:** Is it possible to improve the performance of the neural network by augmenting the dataset with synthetic data?

The literature study shows that it is possible to generate synthetic data using SMOTe (Chawla et al., 2002) or ADASYN (Haibo He et al., 2008). However, the same literature study shows that SMOTe and ADASYN usually don't do better than just plain oversampling (Elor & Averbuch-Elor, 2022). Therefore, we decided not to use SMOTe or ADASYN.

Another idea that came to mind during this research was to add artificial fog to clear images. In Appendix A, a conceptual idea for such an algorithm is proposed. Most synthetic foggy images in Figure A.3 already look convincing, but there is much room for improvement. Unfortunately, there was not enough time during this thesis to test the synthetic images. An idea for future research is to see if the neural network can learn anything from these synthetic images.

Appendix A

Adding Artificial Fog

One of the research objectives is to investigate the possibility of augmenting the dataset with synthetic data. This chapter describes thoughts and findings on adding artificial fog to clear images. A conceptual method is proposed for the addition of fog to clear images. The haze model used in this method is given in Equation (A.1) (Middleton, 2019). This model has been used in many papers about image dehazing until today.

$$\mathbf{I}(\mathbf{x}) = t(\mathbf{x}) \cdot \mathbf{J}(\mathbf{x}) + [1 - t(\mathbf{x})] \cdot \mathbf{A} \tag{A.1}$$

$\mathbf{x}$ is the pixel coordinate, $\mathbf{I}(\mathbf{x})$ is the hazy image, $\mathbf{J}(\mathbf{x})$ is the clear image, $\mathbf{A}$ is a color representing the so-called atmospheric light, which is the color of the fog, and $t(\mathbf{x})$ is the transmission map. The transmission map is given in Equation (A.2).

$$t(\mathbf{x}) = e^{-\beta d(\mathbf{x})} \tag{A.2}$$

Here β is the attenuation coefficient of the atmosphere, and $d(\mathbf{x})$ is the depth map. In the image formation model, β depends on the wavelength. Usually, this dependency is neglected to reduce the number of unknowns.

Firstly, we have to estimate atmospheric light. The idea is that we extract atmospheric light from all foggy images in the dataset. In this way, we can make a distribution of all atmospheric lights and randomly pick one for our algorithm. In most literature, the color of the most haze-opaque region of a foggy image is used as $\mathbf{A}$ or as $\mathbf{A}$'s initial guess. The method for estimating the atmospheric light described in (He et al., 2011) assumes that the atmospheric light is constant, the sky is visible, and the image has no saturated pixels. In this case, just the pixel with the highest intensity can be used as the atmospheric light. In our case, most images do not contain any regions where the sky is visible. Therefore, this assumption is generally not true. There are also more sophisticated methods to estimate atmospheric light from foggy images ((Tan, 2008), (Fattal, 2008)). Secondly, we have to

estimate the depth map. This is the most difficult to estimate. A simple way to estimate the depth map is to use the same depth map for every image. For almost all images, the depth at the bottom of the image is lower, and the depth gradually increases when moving to the top of the image. Suppose we have an image with a horizon and a flat grassland without any objects. We assume the depth map is only dependent on the vertical coordinate. In addition, we assume that the camera has no fish-eye effect in the horizontal direction. The image should look something like the image in Figure A.1.

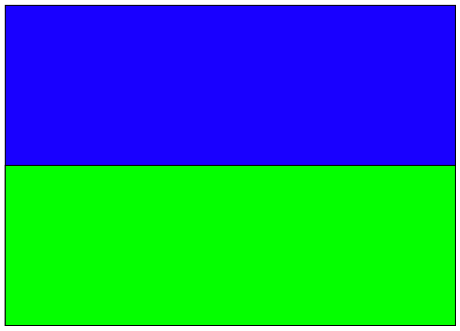


Figure A.1: Horizon image. The green rectangle is the flat grassland, and the blue rectangle is the sky

The depth above the horizon is infinite. In Figure A.2, a simplified optical camera model is illustrated. We will use this model to estimate the depth below the horizon in Figure A.1.

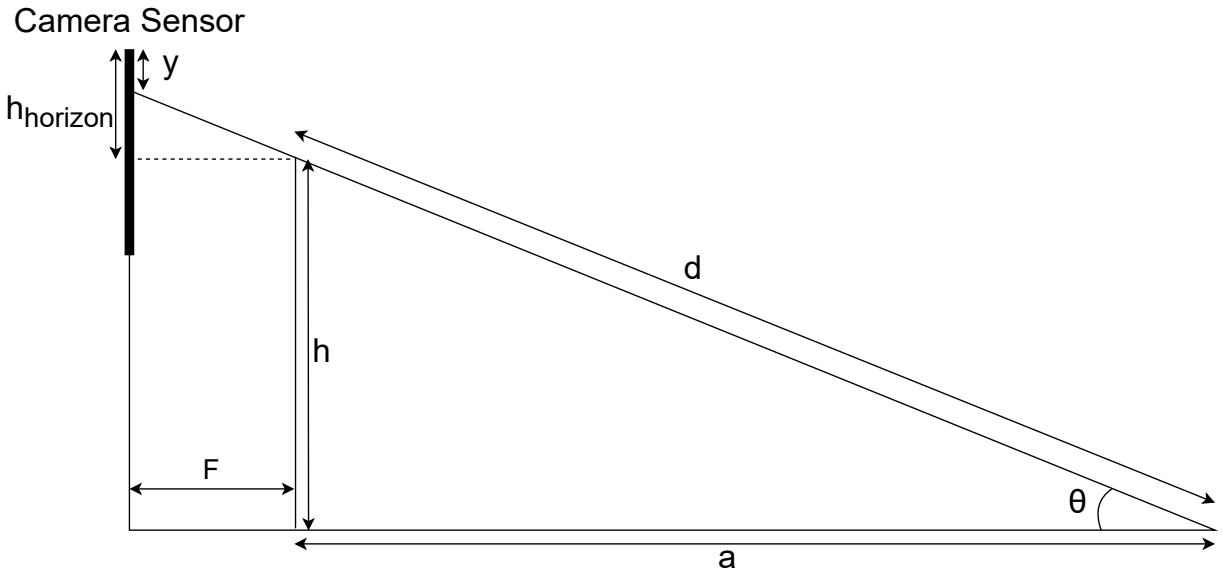


Figure A.2: Simplified optical camera model. The image in this camera model is one-dimensional (a line) because we assume that the depth map depends only on the vertical coordinate. For simplification, the camera is parallel with the grassland in this figure. F is the focal length of the camera. The bold line is the camera sensor, which captures the image upside down. y is the vertical coordinate of the image. The bottom line is the grassland. We are interested in the distance d.

From Figure A.2 it follows that:

$$\frac{h_{horizon} - y}{h} = \frac{F}{a} \tag{A.3}$$

Therefore:

$$a = \frac{Fh}{h_{horizon} - y} \tag{A.4}$$

In addition, we know that $\cos(\theta) = \frac{a}{d} \approx 1$ because we assumed that the height of the camera is much smaller than the depth in the image. Therefore, θ is small. From this, it follows that $a \approx d$. Therefore, we get the following:

$$d(y) = \frac{Fh}{h_{horizon} - y} \tag{A.5}$$

For simplicity, it is assumed that Fh is a constant. However, the focal length F is dependent on the amount of zoom. The height h of most cameras is the same.

This depth map is a huge oversimplification, but it gives us a slightly better understanding of depth in images. This depth map enables us to make an educated guess. In Equation (A.2), we have the constant β, so we can combine those two constants: $\eta = \alpha \cdot \beta$. Therefore, we get the following:

$$t(\mathbf{x}) = \begin{cases} e^{-\eta \frac{1}{h_{horizon} - y}} & y < h_{horizon} \\ 0 & y \geq h_{horizon} \end{cases} \tag{A.6}$$

The horizon has to be chosen. It seems that for most images, the horizon is close to the upper boundary of the image. Therefore, choosing $h_{horizon}$ to be equal to the height of the image might be a good educated guess. In addition, η has to be chosen such that the visibility of the image is below 250 meters. It is probably a good idea to introduce random perturbations to the parameters so that not every depth map is identical. It is also possible to add perturbations to the angle of the horizon. A conceptual algorithm is given in Algorithm 7.

With this conceptual algorithm, it is possible to add artificial fog. In Figure A.3, images with artificial fog are depicted to give an idea of what the algorithm can do. The atmospheric light is chosen such that it looks good because there was not enough time in the project to estimate the atmospheric light for all foggy-labeled images. In addition, perturbations to the atmospheric light are added.

In the simplified model from Equation (A.6), we only have to estimate the horizon height $h_{horizon}$ and η. For this, we had to assume that $\cos(\theta) = \frac{a}{d} \approx 1$ and that the focal length F is a constant. However, if we find a way to estimate F, another model might be better. If we estimate the transmission map without this assumption, we get

Algorithm 7 Concept algorithm Artificial Fog

Let $\mathbf{X}_{clear}$ be a set of clear images
Let $\mathbf{X}_{synthetic}$ be the empty set
Estimate Atmospheric light for every foggy-labeled image and add them to a list $\mathbf{A}_{list}$
Choose η in Equation (A.6) such that the images look convincingly foggy ($<$ 250meter).

for $X \in \mathbf{X}_{clear}$ **do**
Add some perturbations to the chosen parameters in Equation (A.6) and calculate the transmission map $t(y)$
Choose an atmospheric light randomly from the list $\mathbf{A}_{list}$
Use Equation (A.1) to add fog to the clear image
Add the image with synthetic fog to $\mathbf{X}_{synthetic}$
end for
The set $\mathbf{X}_{synthetic}$ now contains synthetic foggy images.

Equation (A.7).

$$t(\mathbf{x}) = \begin{cases} e^{-\kappa \frac{1}{\sin \arctan\left(\frac{h_{horizon}-y}{F}\right)}} & y < h_{horizon} \\ 0 & y \geq h_{horizon} \end{cases} \tag{A.7}$$

Where $\kappa = \beta \cdot h$. The model in Equation (A.7) should be slightly more realistic, but we have to estimate one more parameter, which is very challenging.

A.1 Discussion and Conclusion

Most images from Figure A.3 already look convincing. However, there is much room for improvement. Firstly, the depth map for every image is the same at the moment, which is not realistic. This is partly the reason some images look more foggy than others. If the camera is tilted downward, the image appears more foggy. On the other hand, if the camera is tilted upward so the sky is visible, the image appears less foggy. This is because the horizon is fixed at the moment. One improvement to the algorithm would be if we were able to estimate the horizon. Another improvement would be if we could somehow estimate the focal length F because this enables us to use Equation (A.7). However, this is very challenging. In addition, it is assumed that the ground is a flat plane without any objects at the moment, which is not realistic either. In order to solve this, we have to use a completely different model for the depth map estimation. One option would be to train a neural network on the KITTI dataset (Geiger, Lenz, Stiller, & Urtasun, 2013) to estimate the depth map. The KITTI dataset contains dash cam images with a depth map. However, the height of the camera is completely different compared to traffic cameras, which might be a problem.

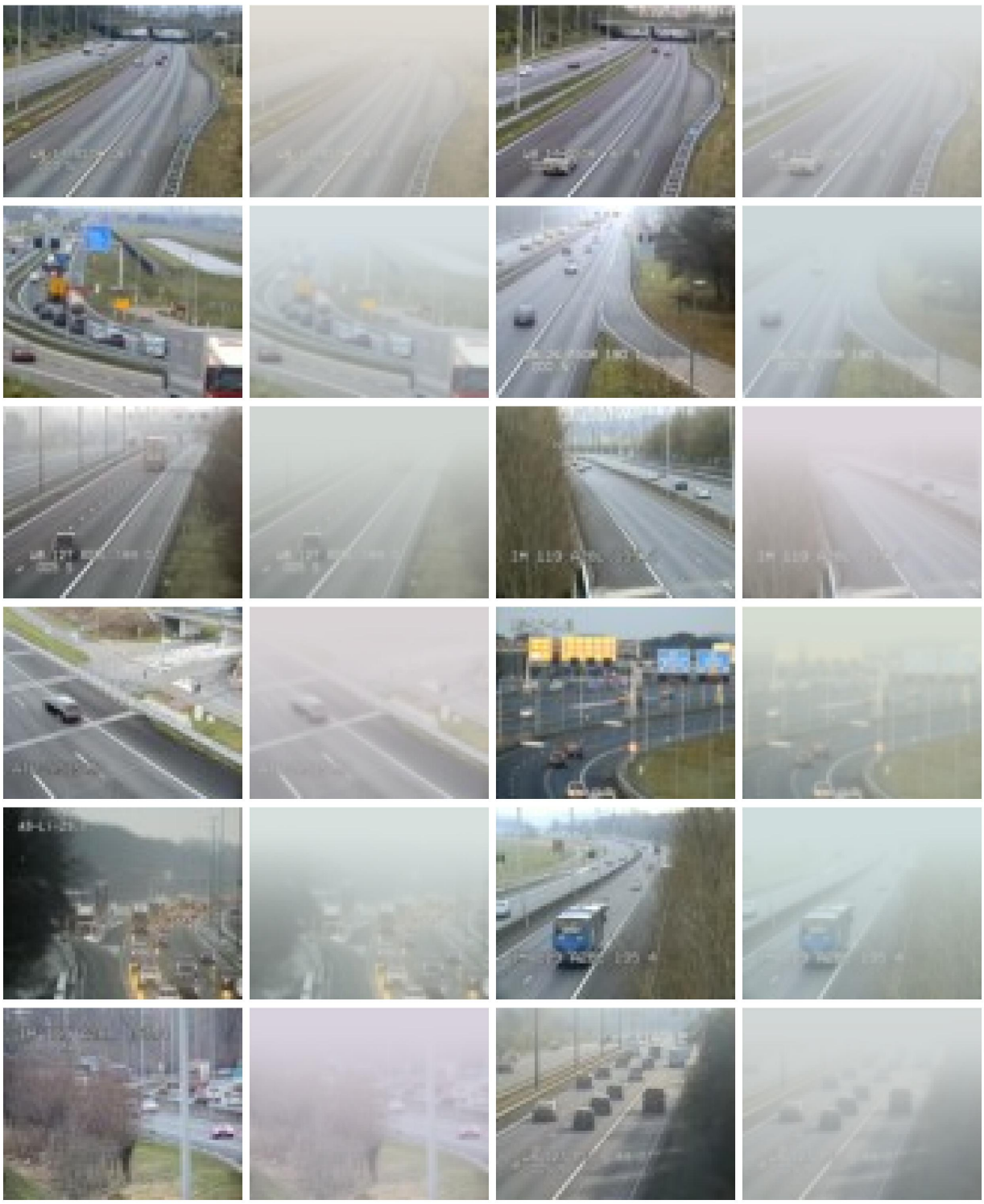


Figure A.3: Illustration of the effect of adding artificial fog to images. Images are randomly picked. The first and third columns contain the original clear-labeled images, and the second and fourth columns contain the same images with artificial fog. The images are of the same resolution as those used for training. $\eta = 40$. The horizon $h_{horizon}$ is chosen to be one pixel above the image. The atmospheric light is chosen such that it looks good because there was not enough time to fully implement Algorithm 7. In addition, perturbations to the atmospheric light are added.

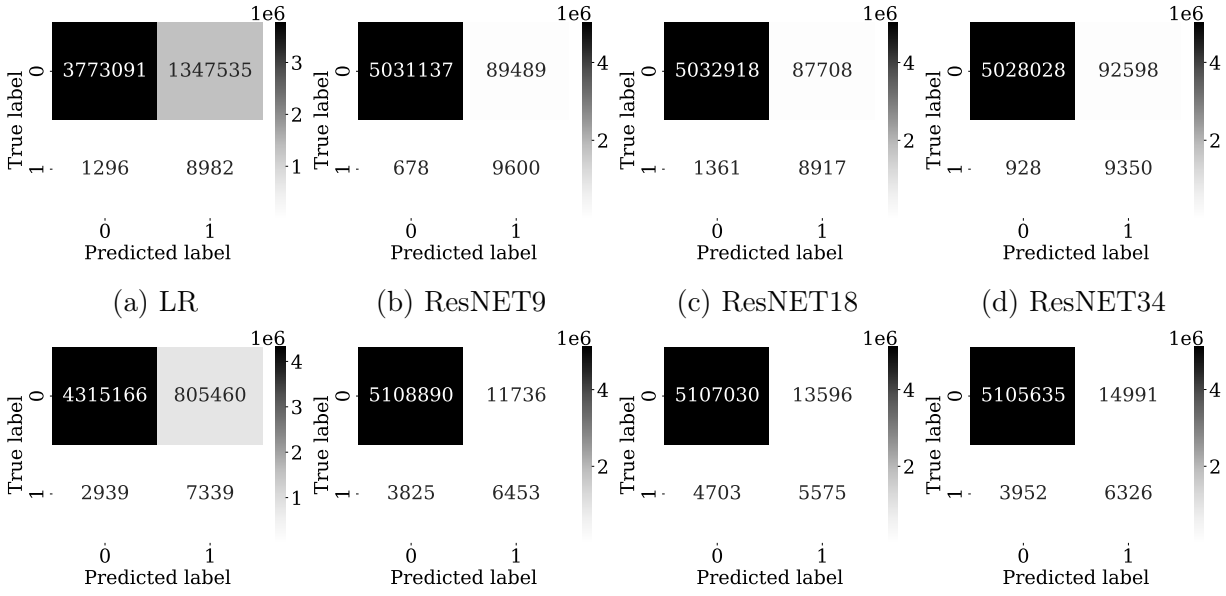
Appendix B

Confusion Matrices

B.1 Pre-trained Weights

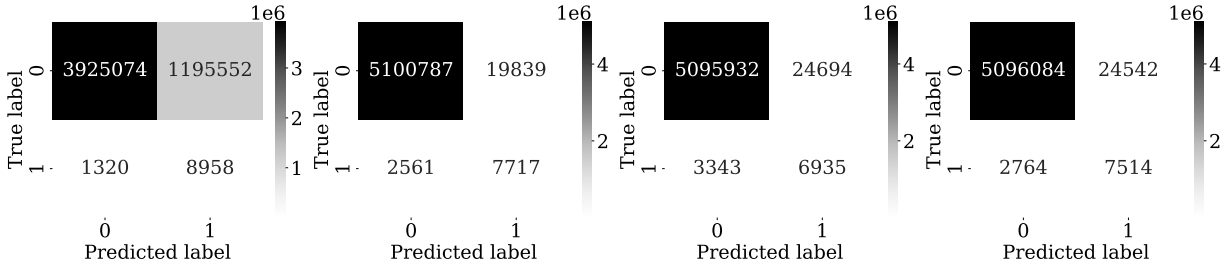
Confusion matrices for all the experiments described in Section 6.1.1 and Section 6.1.2.

No Oversampling



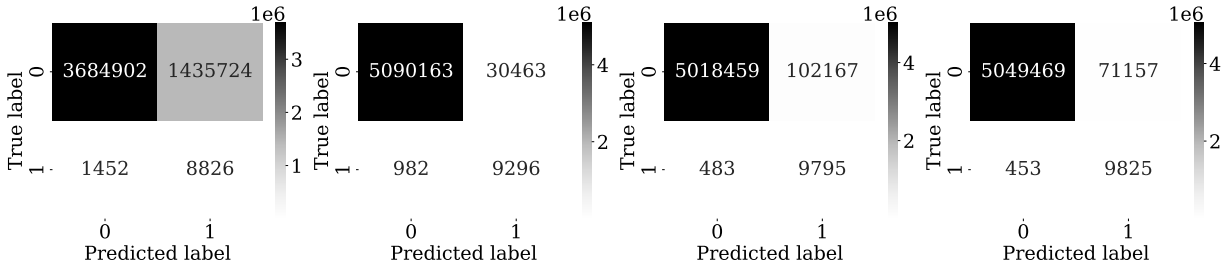
(a) LR (b) ResNET9 (c) ResNET18 (d) ResNET34

(e) LR with T1 (f) ResNET9 with T1 (g) ResNET18 with T1 (h) ResNET34 with T1

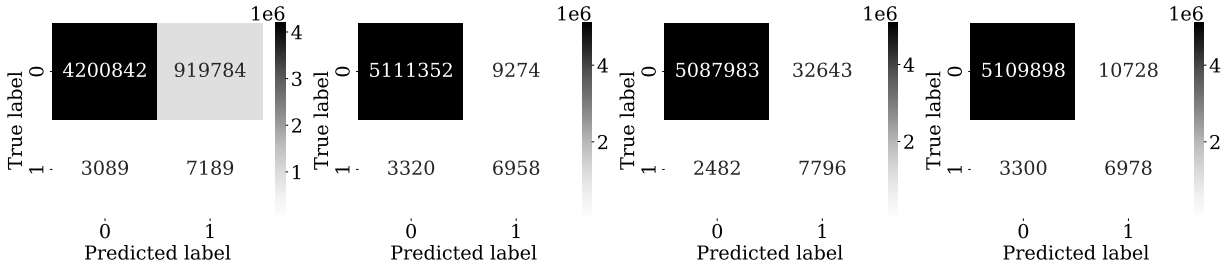


(i) LR with T2 (j) ResNET9 with T2 (k) ResNET18 with T2 (l) ResNET34 with T2

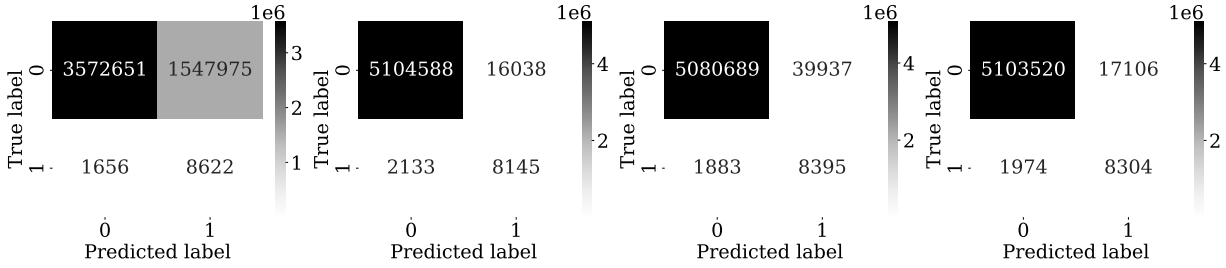
Foggy-Labeled Images Oversampled to 5 Percent



(a) LR (b) ResNET9 (c) ResNET18 (d) ResNET34

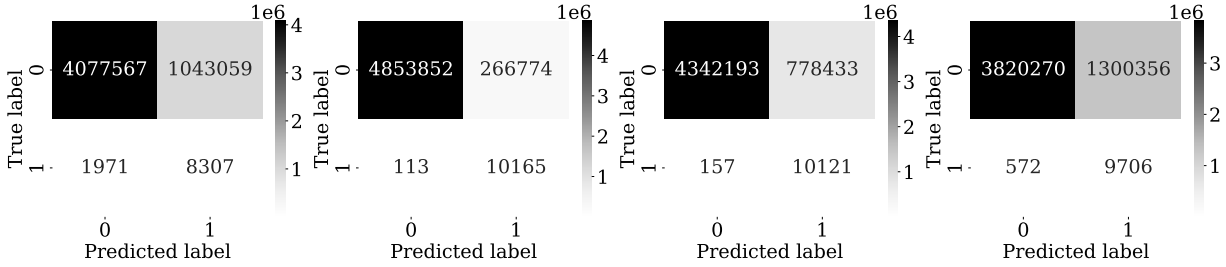


(e) LR with T1 (f) ResNET9 with T1 (g) ResNET18 with T1 (h) ResNET34 with T1

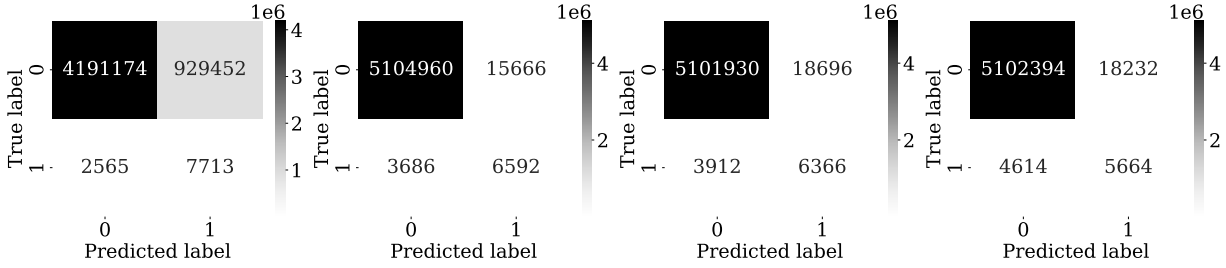


(i) LR with T2 (j) ResNET9 with T2 (k) ResNET18 with T2 (l) ResNET34 with T2

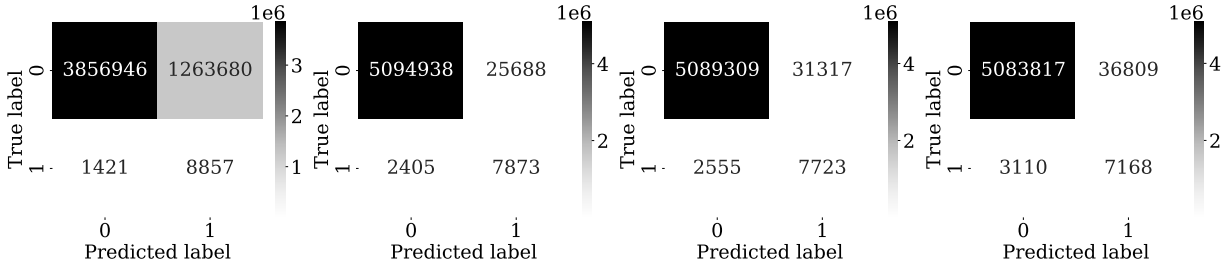
Foggy-Labeled Images Oversampled to 50 Percent



(a) LR (b) ResNET9 (c) ResNET18 (d) ResNET34



(e) LR with T1 (f) ResNET9 with T1 (g) ResNET18 with T1 (h) ResNET34 with T1

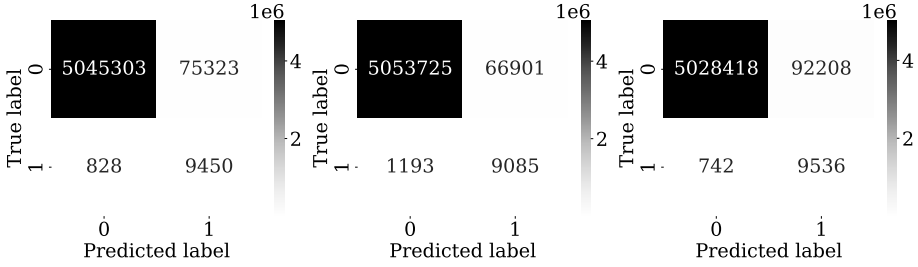


(i) LR with T2 (j) ResNET9 with T2 (k) ResNET18 with T2 (l) ResNET34 with T2

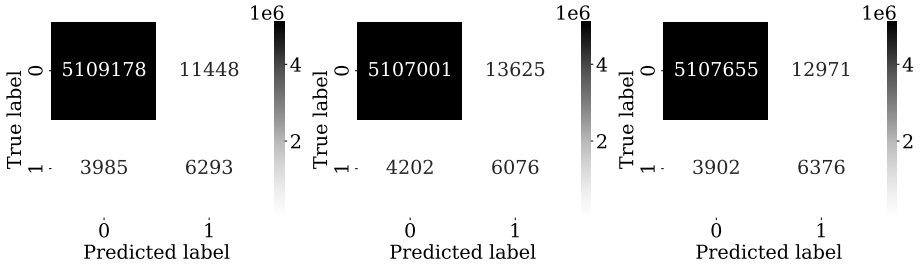
B.2 Random Weights

Confusion matrices for all the experiments described in Section 6.2.1

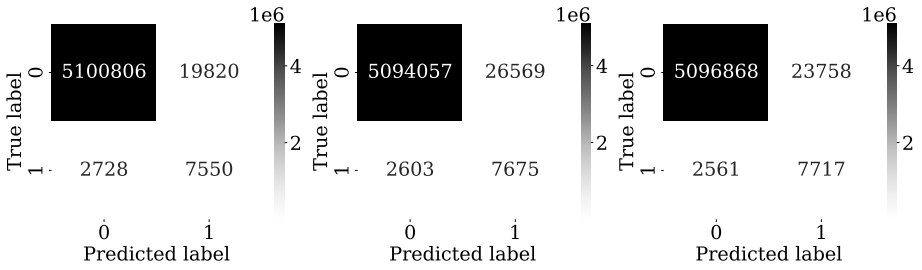
No Oversampling



(a) ResNET9 (b) ResNET18 (c) ResNET34

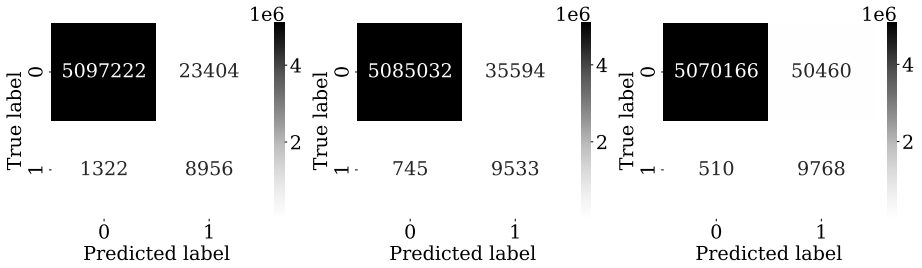


(d) ResNET9 with T1 (e) ResNET18 with T1 (f) ResNET34 with T1

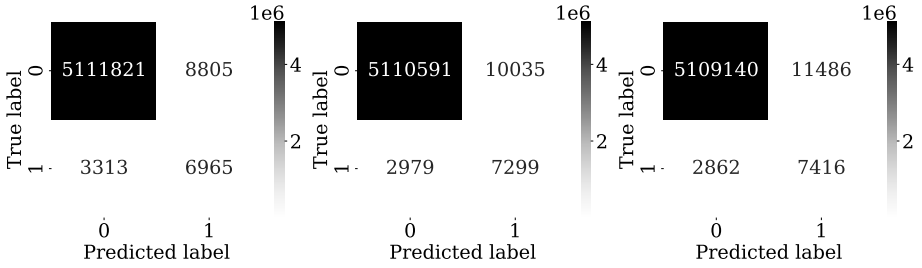


(g) ResNET9 with T2 (h) ResNET18 with T2 (i) ResNET34 with T2

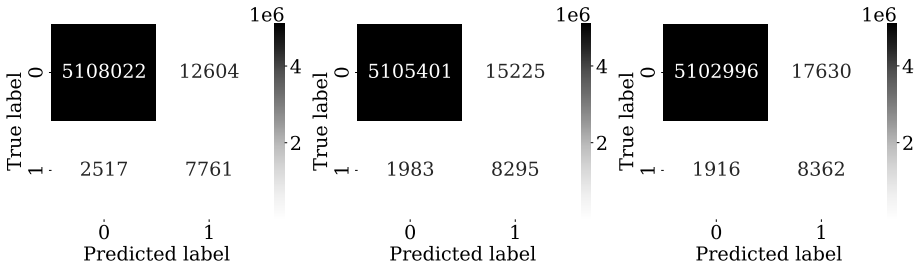
Foggy-Labeled Images Oversampled to 5 Percent



(a) ResNET9 (b) ResNET18 (c) ResNET34



(d) ResNET9 with T1 (e) ResNET18 with T1 (f) ResNET34 with T1

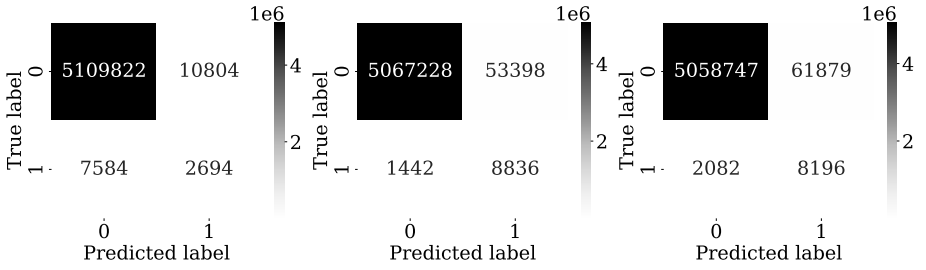


(g) ResNET9 with T2 (h) ResNET18 with T2 (i) ResNET34 with T2

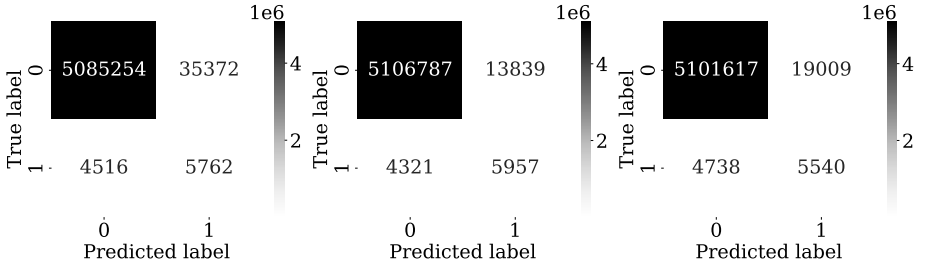
B.3 Using FFT in Preprocessing Stage

Confusion matrices for all the experiments described in Section 6.2.2

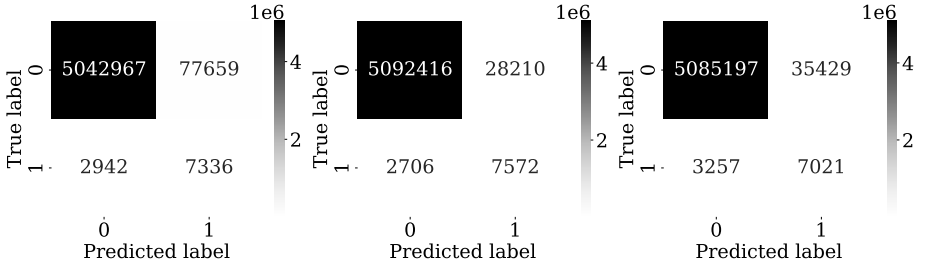
No Oversampling



(a) LR (b) ResNET9 (c) ResNET18

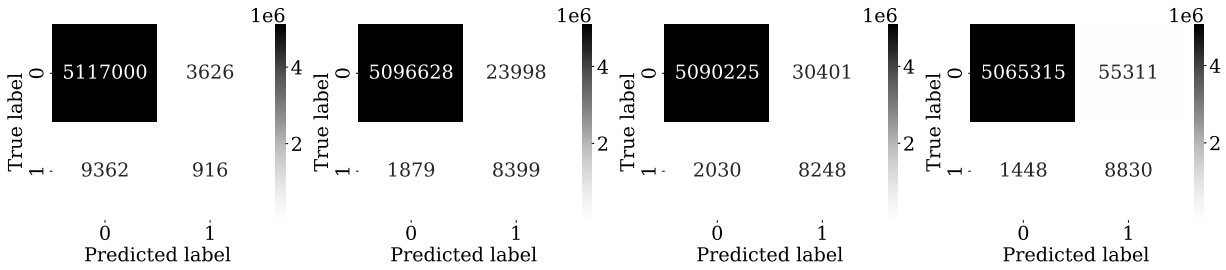


(d) LR with T1 (e) ResNET9 with T1 (f) ResNET18 with T1

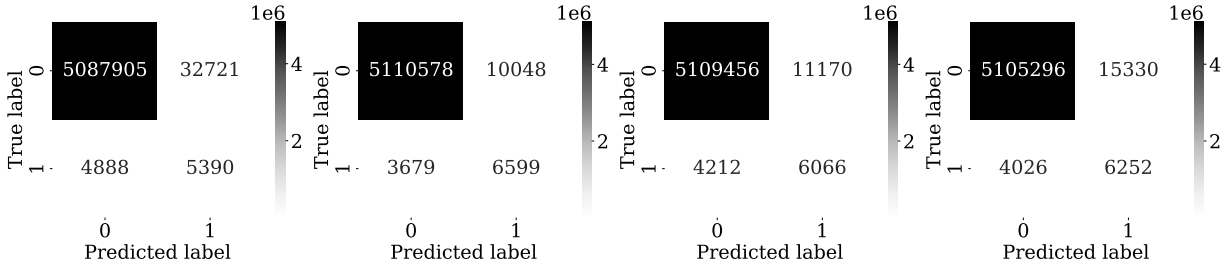


(g) LR with T2 (h) ResNET9 with T2 (i) ResNET18 with T2

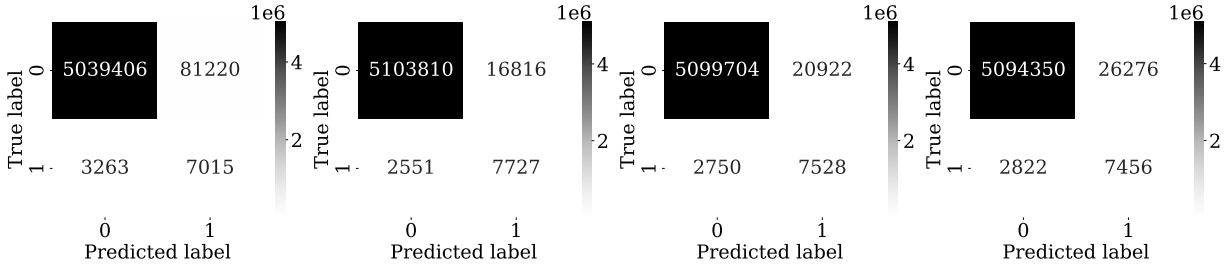
Foggy-Labeled Images Oversampled to 5 Percent



(a) LR (b) ResNET9 (c) ResNET18 (d) ResNET34



(e) LR with T1 (f) ResNET9 with T1 (g) ResNET18 with T1 (h) ResNET34 with T1

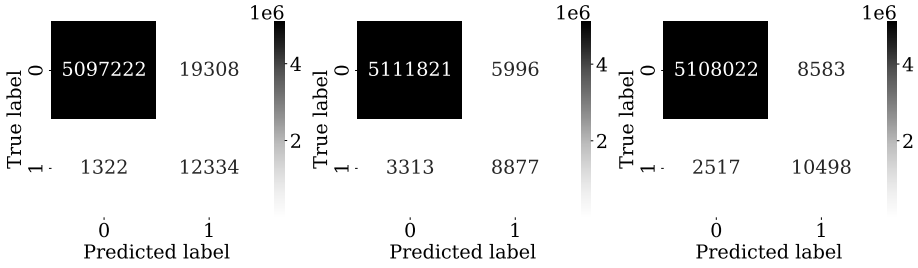


(i) LR with T2 (j) ResNET9 with T2 (k) ResNET18 with T2 (l) ResNET34 with T2

B.4 Corrected Performance of ResNET9

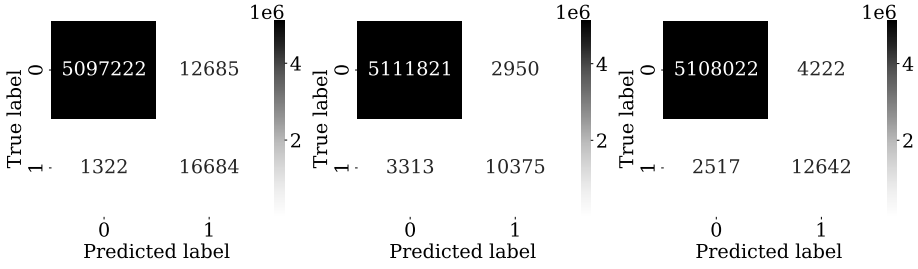
Confusion matrices for all the experiments described in Section 6.3.

Lower Boundary 95% confidence interval



(a) ResNET9 (b) ResNET9 with T1 (c) ResNET9 with T2

Upper Boundary 95% confidence interval



(a) ResNET9 (b) ResNET9 with T1 (c) ResNET9 with T2

References

Abdel-Aty, M., Ekram, A.-A., Huang, H., & Choi, K. (2011). A study on crashes related to visibility obstruction due to fog and smoke. *Accident Analysis & Prevention*, *43*(5), 1730 - 1737. Retrieved from http://www.sciencedirect.com/science/article/pii/S0001457511000844 doi: https://doi.org/10.1016/j.aap.2011.04.003

Andrea Pagani, J. W. N., & Wauben, W. (2018). *Deep neural network approach for automatic fog detection using traffic camera images.*

Ash, J. T., & Adams, R. P. (2020). *On warm-starting neural network training.*

Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., & Vaughan, J. W. (2010, May 01). A theory of learning from different domains. *Machine Learning*, *79*(1), 151–175. Retrieved from https://doi.org/10.1007/s10994-009-5152-4 doi: 10.1007/s10994-009-5152-4

Ben-David, S., & Urner, R. (2012). On the hardness of domain adaptation and the utility of unlabeled target samples. In N. H. Bshouty, G. Stoltz, N. Vayatis, & T. Zeugmann (Eds.), *Algorithmic learning theory* (pp. 139–153). Berlin, Heidelberg: Springer Berlin Heidelberg.

Ben-David, S., & Urner, R. (2014). Domain adaptation-can quantity compensate for quality? *Annals of Mathematics and Artificial Intelligence*, *70*(3), 185-202. Retrieved from https://doi.org/10.1007/s10472-013-9371-9 doi: 10.1007/s10472-013-9371-9

Bengtsson, L., Andrae, U., Aspelien, T., Batrak, Y., Calvo, J., de Rooy, W., ... Ødegaard Køltzow, M. (2017). The harmonie–arome model configuration in the aladin–hirlam nwp system. *Monthly Weather Review*, *145*(5), 1919 - 1935. Retrieved from https://journals.ametsoc.org/view/journals/mwre/145/5/mwr-d-16-0417.1.xml doi: 10.1175/MWR-D-16-0417.1

Berman, D., Treibitz, T., & Avidan, S. (2016, June). Non-local image dehazing. In *2016 ieee conference on computer vision and pattern recognition (cvpr)* (p. 1674-1682). doi: 10.1109/CVPR.2016.185

Bishop, C. M. (1995). Training with noise is equivalent to tikhonov regularization. *Neural Computation*, *7*(1), 108-116. doi: 10.1162/neco.1995.7.1.108

Bozinovski, S. (2020, 09). Reminder of the first paper on transfer learning in neural networks, 1976. *Informatica*, *44*. doi: 10.31449/inf.v44i3.2828

Buda, M., Maki, A., & Mazurowski, M. A. (2018, oct). A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, *106*, 249–259. Retrieved from https://doi.org/10.1016%2Fj.neunet.2018.07.011 doi: 10.1016/j.neunet.2018.07.011

Cai, B., Xu, X., Jia, K., Qing, C., & Tao, D. (2016). Dehazenet: An end-to-end system for single image haze removal. *CoRR*, *abs/1601.07661*. Retrieved from http://arxiv.org/abs/1601.07661

Cavallo, V., Colomb, M., & Doré, J. (2001). Distance perception of vehicle rear lights in fog. *Human Factors*, *43*(3), 442-451. Retrieved from https://doi.org/10.1518/001872001775898197 (PMID: 11866199) doi: 10.1518/001872001775898197

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002, jun). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, *16*, 321–357. Retrieved from https://doi.org/10.1613/jair.953 doi: 10.1613/jair.953

Chicco, D., & Jurman, G. (2020, 01). The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, *21*. doi: 10.1186/s12864-019-6413-7

Chicco, D., Tötsch, N., & Jurman, G. (2021, February). The matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation. *BioData Min*, *14*(1), 13.

Chinchor, N. (1992). Muc-4 evaluation metrics. In *Proceedings of the 4th conference on message understanding* (p. 22–29). USA: Association for Computational Linguistics. Retrieved from https://doi.org/10.3115/1072064.1072067 doi: 10.3115/1072064.1072067

Cortes, C., & Mohri, M. (2014). Domain adaptation and sample bias correction theory and algorithm for regression. *Theoretical Computer Science*, *519*, 103 - 126. Retrieved from http://www.sciencedirect.com/science/article/pii/S0304397513007184 (Algorithmic Learning Theory) doi: https://doi.org/10.1016/j.tcs.2013.09.027

Cortes, C., Mohri, M., & Medina, A. M. (2019). Adaptation based on generalized discrepancy. *Journal of Machine Learning Research*, *20*(1), 1-30. Retrieved from http://jmlr.org/papers/v20/15-192.html

David, S. B., Lu, T., Luu, T., & Pal, D. (2010, 13–15 May). Impossibility theorems for domain adaptation. In Y. W. Teh & M. Titterington (Eds.), *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (Vol. 9, pp. 129–136). Chia Laguna Resort, Sardinia, Italy: PMLR. Retrieved from http://proceedings.mlr.press/v9/david10a.html

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, *29*(6), 141–142.

Elor, Y., & Averbuch-Elor, H. (2022). To smote, or not to smote? *CoRR*, *abs/2201.08528*. Retrieved from https://arxiv.org/abs/2201.08528

Fattal, R. (2008). Single image dehazing. In *Acm siggraph 2008 papers.* New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1399504.1360671 doi: 10.1145/1399504.1360671

Fattal, R. (2015, December). Dehazing using color-lines. *ACM Trans. Graph.*, *34*(1). Retrieved from https://doi.org/10.1145/2651362 doi: 10.1145/2651362

Fawcett, T. (2006). An introduction to roc analysis. *Pattern Recognition Let-*

ters, *27*(8), 861-874. Retrieved from https://www.sciencedirect.com/science/article/pii/S016786550500303X (ROC Analysis in Pattern Recognition) doi: https://doi.org/10.1016/j.patrec.2005.10.010

Fukushima, K. (1980, Apr 01). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, *36*(4), 193-202. Retrieved from https://doi.org/10.1007/BF00344251 doi: 10.1007/BF00344251

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., ... Lempitsky, V. (2015). *Domain-adversarial training of neural networks.*

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

Germain, P., Habrard, A., Laviolette, F., & Morvant, E. (2015). *Pac-bayesian theorems for domain adaptation with specialization to linear classifiers.*

Haibo He, Yang Bai, Garcia, E. A., & Shutao Li. (2008, June). Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 ieee international joint conference on neural networks (ieee world congress on computational intelligence)* (p. 1322-1328). doi: 10.1109/IJCNN.2008.4633969

He, K., Sun, J., & Tang, X. (2011, Dec). Single image haze removal using dark channel prior. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(12), 2341-2353. doi: 10.1109/TPAMI.2010.168

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, *abs/1512.03385*. Retrieved from http://arxiv.org/abs/1512.03385

Hinton, G. (2012). *Lecture slides - lecture 6 - rmsprop.* Retrieved from https://www.cs.toronto.edu/~tijmen/csc321//slides/lecture_slides_lec6.pdf

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, *abs/1502.03167*. Retrieved from http://arxiv.org/abs/1502.03167

James, G., Witten, D., Hastie, T., & Tibshirani, R. ([2013]). *An introduction to statistical learning : with applications in r.* New York : Springer, [2013] ©2013. Retrieved from https://search.library.wisc.edu/catalog/9910207152902121 (Includes bibliographical references and index.)

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

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization.* arXiv. Retrieved from https://arxiv.org/abs/1412.6980 doi: 10.48550/ARXIV.1412.6980

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient backprop. In G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), *Neural networks: Tricks of the*

trade: Second edition (pp. 9–48). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from https://doi.org/10.1007/978-3-642-35289-8_3 doi: 10.1007/978-3-642-35289-8_3

Lobo, J. M., Jiménez-Valverde, A., & Real, R. (2008). Auc: a misleading measure of the performance of predictive distribution models. *Global Ecology and Biogeography*, *17*(2), 145-151. Retrieved from https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1466-8238.2007.00358.x doi: https://doi.org/10.1111/j.1466-8238.2007.00358.x

Mann, H. B., & Whitney, D. R. (1947). On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, *18*(1), 50 – 60. Retrieved from https://doi.org/10.1214/aoms/1177730491 doi: 10.1214/aoms/1177730491

Mansour, Y., Mohri, M., & Rostamizadeh, A. (2009). Domain adaptation: Learning bounds and algorithms. *CoRR*, *abs/0902.3430*. Retrieved from http://arxiv.org/abs/0902.3430

Matthews, B. W. (1975, October). Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochim Biophys Acta*, *405*(2), 442–451.

Maxouris, C. (2019, January). *Heavy fog causes 20-plus vehicle collision in texas.* CNN. Retrieved from https://edition.cnn.com/2019/01/01/us/austin-20-vehicle-crash/index.html

McCulloch, W. S., & Pitts, W. (1943, Dec 01). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115-133. Retrieved from https://doi.org/10.1007/BF02478259 doi: 10.1007/BF02478259

Middleton, W. (2019). *Vision through the atmosphere.* Toronto: University of Toronto Press. Retrieved from https://www.degruyter.com/view/title/543450

Molleman, T. (2018). *Dehazing fog classification.*

Nielsen, M. A. (2019, Dec). *How the backpropagation algorithm works.* Determination Press. Retrieved from http://neuralnetworksanddeeplearning.com/chap2.html

Omer, I., & Werman, M. (2004, 06). Color lines: image specific color representation. In (Vol. 2, p. II-946). doi: 10.1109/CVPR.2004.1315267

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2022). *Bcewithlogitsloss.* Retrieved from https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html ([Online; accessed 2023-02-03])

Pearson, K. (1895). Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, *58*, 240–242. Retrieved 2023-02-17, from http://www.jstor.org/stable/115794

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Redondo, A. R., Navarro, J., Fernández, R. R., de Diego, I. M., Moguerza, J. M., & Fernández-Muñoz, J. J. (2020). Unified performance measure for binary classification problems. In C. Analide, P. Novais, D. Camacho, & H. Yin (Eds.), *Intelligent data engineering and automated learning – ideal 2020* (pp. 104–112). Cham: Springer International Publishing.

Ren, W., Pan, J., Zhang, H., Cao, X., & Yang, M.-H. (2020). Single image dehazing via multi-scale convolutional neural networks with holistic edges. *International Journal of Computer Vision*, *128*(1), 240-259. Retrieved from `https://doi.org/10.1007/s11263-019-01235-8` doi: 10.1007/s11263-019-01235-8

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton project para.* Cornell Aeronautical Laboratory. Retrieved from `https://books.google.nl/books?id=P_XGPgAACAAJ`

Sakaridis, C., Dai, D., & Van Gool, L. (2018, Sep). Semantic foggy scene understanding with synthetic data. *International Journal of Computer Vision*, *126*(9), 973–992. Retrieved from `https://doi.org/10.1007/s11263-018-1072-8`

Schechner, Y. Y., Narasimhan, S. G., & Nayar, S. K. (2001, Dec). Instant dehazing of images using polarization. In *Proceedings of the 2001 ieee computer society conference on computer vision and pattern recognition. cvpr 2001* (Vol. 1, p. I-I). doi: 10.1109/CVPR.2001.990493

Schechner, Y. Y., Narasimhan, S. G., & Nayar, S. K. (2003, Jan). Polarization-based vision through haze. *Appl. Opt.*, *42*(3), 511–525. Retrieved from `http://ao.osa.org/abstract.cfm?URI=ao-42-3-511` doi: 10.1364/AO.42.000511

Schisterman, E. F., Perkins, N. J., Liu, A., & Bondell, H. (2005). Optimal cut-point and its corresponding youden index to discriminate individuals using pooled blood samples. *Epidemiology*, *16*(1). Retrieved from `https://journals.lww.com/epidem/Fulltext/2005/01000/Optimal_Cut_point_and_Its_Corresponding_Youden.11.aspx`

Sheth, K. (2019, March). *Largest traffic accident pile-ups in history.* Worldatlas. Retrieved from `https://www.worldatlas.com/articles/largest-traffic-accident-pile-ups-in-history.html`

Shijie, J., Ping, W., Peiyi, J., & Siping, H. (2017). Research on data augmentation for image classification based on convolution neural networks. *2017 Chinese Automation Congress (CAC)*, 4165-4170.

Shorten, C., & Khoshgoftaar, T. M. (2019, Jul 06). A survey on image data augmentation for deep learning. *Journal of Big Data*, *6*(1), 60. Retrieved from `https://doi.org/10.1186/s40537-019-0197-0` doi: 10.1186/s40537-019-0197-0

Simard, P., Steinkraus, D., & Platt, J. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Seventh international conference on document analysis and recognition, 2003. proceedings.* (p. 958-963). doi: 10.1109/ICDAR.2003.1227801

Simonyan, K., & Zisserman, A. (2014). *Very deep convolutional networks for large-scale image recognition.* arXiv. Retrieved from https://arxiv.org/abs/1409.1556 doi: 10.48550/ARXIV.1409.1556

Swami, K., & Das, S. K. (2018). CANDY: conditional adversarial networks based fully end-to-end system for single image haze removal. *CoRR*, *abs/1801.02892*. Retrieved from http://arxiv.org/abs/1801.02892

Swets, J. A., Dawes, R. M., & Monahan, J. (2000, October). Better decisions through science. *Sci Am*, *283*(4), 82–87.

Tan, R. T. (2008). Visibility in bad weather from a single image. In *2008 ieee conference on computer vision and pattern recognition* (p. 1-8).

U.S. Department of Commerce, N. O., & Administration, A. (2017). *Federal meteorological handbook no. 1: Surface weather observations and reports.* Washington, D.C..

Wallis, S. (2013). Binomial confidence intervals and contingency tests: Mathematical fundamentals and the evaluation of alternative methods. *Journal of Quantitative Linguistics*, *20*(3), 178-208. Retrieved from https://doi.org/10.1080/09296174.2013.799918 doi: 10.1080/09296174.2013.799918

Wang, Y., Liang, L., & Evans, L. (2017). Fatal crashes involving large numbers of vehicles and weather. *Journal of Safety Research*, *63*, 1 - 7. Retrieved from http://www.sciencedirect.com/science/article/pii/S0022437517302736 doi: https://doi.org/10.1016/j.jsr.2017.08.001

Wauben, W., & Roth, M. (2016). *Exploration of fog detection and visibility estimation from camera images.* Retrieved from https://www.wmocimo.net/eventpapers/session2/O2(8)_Wauben_Camera-fog-visibility.pdf

Werbos, P., & John, P. (1974, 01). *Beyond regression : new tools for prediction and analysis in the behavioral sciences /.*

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., & Recht, B. (2017). *The marginal value of adaptive gradient methods in machine learning.* arXiv. Retrieved from https://arxiv.org/abs/1705.08292 doi: 10.48550/ARXIV.1705.08292

Xu, H., Guo, J., Liu, Q., & Ye, L. (2012, March). Fast image dehazing using improved dark channel prior. In *2012 ieee international conference on information science and technology* (p. 663-667). doi: 10.1109/ICIST.2012.6221729

Zhang, Y., Liu, T., Long, M., & Jordan, M. I. (2019). Bridging theory and algorithm for domain adaptation. *CoRR*, *abs/1904.05801*. Retrieved from http://arxiv.org/abs/1904.05801

Zhou, P., Feng, J., Ma, C., Xiong, C., Hoi, S., & E, W. (2020). *Towards theoretically understanding why sgd generalizes better than adam in deep learning.* arXiv. Retrieved from https://arxiv.org/abs/2010.05627 doi: 10.48550/ARXIV.2010.05627