Fog Detection using an Artificial Neural Network

Quanwei Li, Tiancheng Ma

Master's thesis
2023:E48

Lund University

Faculty of Science
Centre for Mathematical Sciences
Mathematical Statistics

Centrum Scientiarum Mathematicarum

Abstract

This project studies a method of image-based fog detection directly from a camera without using the transmissometer.

Fog can be detected using transmissometers which could be a very costly approach. This thesis presents an image-based approach for fog detection using Artificial Neural networks. A neural network model will be used to classify images either into two classes, whether it contains fog or not, or into multiple fog intensity classes, based on visibility derived from Koschmieder's Law. By applying neural network model foggy weather can be detected directly from surveillance cameras and makes fairly accurate predictions, which is useful in many aspects of industry and life.

The goal of the thesis is to improve on previous methods that have used neural networks for similar tasks. To achieve the goal, the work starts with creating a suitable dataset for training and validation based on a combination of real images and images generated from Unreal Engine or other simulation scrips. And then, create an effective model based on neural network with the dataset to solve the classification problem. Finally, validate the completed model and implement it on a real surveillance camera to test the actual performance, the model will be retrained if necessary according to the test result.

Keywords: Machine Learning, Deep Learning, Image Analysis, Computer Vision

Acknowledgement

First of all, our sincere thanks to our kind and knowledgeable supervisors Johan Lindström (LU) and Joakim Viklund (AXIS Communications AB) for providing the precious opportunity to work on this project. We could not imagine progressing through the hardest parts of the project without their kind support, informative discussions and valuable feedback. We are forever grateful.

In addition, we would like to thank our colleagues at the PTZ Imaging group for helping us come up with brilliant ideas. We are grateful for you all making us feel like a part of the team and welcoming us in the best way possible.

Finally, we would like to thank our parents and friends for their support during the hardest times of the thesis project and for taking an interest in our project.

Contents

Chapter 1

Introduction

Foggy weather could severely affect image quality and decreases the web-cam user experience. In addition, foggy weather poses a significant risk in transportation. Reduced visibility can seriously affect autonomous vehicles with image sensors and other applications such as sector and public safety surveillance, highway safety supervision, that require excellent visibility in the footage. Defogging filters are now widely applied to cameras, yet they come with the cost of reduced image quality. Therefore, one needs to think of a method to detect the reduction in visibility and determine when the filter should be applied to improve image quality.

Usually, the visibility measurement is performed by transmissometers, consisting of a ray transmitter emitting rays and a receiver that receives and detects the loss of ray energy. The measure is robust, yet the cost and complexity of such sensors are often prohibitive. Therefore, image-based visibility estimation methods would provide an interesting alternative. Most work regarding visibility estimation are based on the Koschmieder law which suggests that visibility is inversely proportional to the atmospheric extinction coefficient. The coefficient describes how the loss in light ray energy increases when passing through increasing aerosol concentrations.

Several previous studies have been carried out to achieve image-based visibility detection without using transmissometers, where some use image component analysis ([19, 7]) and some use artificial intelligence approaches ([12, 4]).

1.1 Previous Works

In 2016, [19] presented an image-based method of estimating air transmissibility through landmark discrimination via edge detection and using a dark-channel prior. That work was an extension of previous work from [7], which uses the dark channel prior and edge detection. Another method, presented in [11] uses images from a vehicle and is applied to estimate the horizon line based on an edge detection algorithm finding the point where the road feature (e.g. lane markings) vanishes. The author of [19] used the Sobel edge detection algorithm to detect the edge of the pedestrian crossing and the lane markings on the ground. Lastly, in [2], the authors used similar approaches which also includes Sobel edge detection. In addition, Hough line detection([8]) is used to estimate vanishing points to segment the road and sky, and finally proceed with all the image component information to classify various intensities of fog. However, this method mainly works on road images.

On the other hand, some research papers have applied Artificial Neural Networks to classify weather conditions and different visibility classes. In 2018, [12] presented the method of using a Deep Feed-Forward Neural Network to perform binary visibility classification based on road and weather cameras across Amsterdam. Analysing the model fitting result presented from [12], the model yielded high accuracy solely due to the sheer amount of images used. Judging from the confusion matrix, the model did not perform well because of the heavily imbalanced dataset that contains way more non-foggy pictures than foggy pictures, resulting in a high false negative rate and low F1-scores of 0.19, 0.35, 0.51 and 0.65 for each of the models they created.

The model presented in [12] does not seem appropriate as the data selection was not carefully performed. In particular, using a deep Feed-Forward Neural Network will not excel in dealing with images. On the bright side, [12] shows that applying an artificial neural network is a possible solution to fog detection. In another study [4], the researchers used Convolutional Neural Networks to classify different intensities of fog based on visibility, producing very robust results. However, their image data were mainly based on the FROSI (Foggy ROad Sign Images) dataset which only contains simulated road images with traffic signs and fog. Hence, this neural network model could be less useful when implemented on cameras located in more diverse

environments. The work presented in [4] indicates a possibility to use generated images for training, and using Convolutional Neural Network models seems very effective.

The overview of previous work motivated us to try the artificial neural network approach for image-based fog detection using image data that could be gathered from real life, from the internet or obtain by synthetic image generation using fog generation schematics based on Koschmieder law.

1.2 Project Description

The project aims to create several artificial neural network models that can detect visibility reduction and then use the predictions to determine which defogging filter to apply to improve image quality.

(a) A photo without fog. (b) A foggy photo.

Figure 1.1: Two image examples of foggy and clear photos. Author: Quanwei Li

The first step was to collect image data for model training and validation (see Chapter 2). This includes but is not limited to a) gathering images from the internet, b) fetching publically available web camera images through open-data APIs and c) taking pictures in real life (see 1.1. In addition, the weather data was collected from Trafikverket (Swedish Transportation Administration) and SMHI (Swedish Meteorological and Hydrologic Institution). The weather data will be used for the detection task of multiple intensities of fog. In addition to real pictures, synthetic image generation

will be used to generate more image data for training, so the network will yield good performance on a diverse image dataset. The data will be modelled using a neural network and an overview of the theory is given in Chapter 3

The second step (see Chapter 4.2.1 - 4.2.3) is to construct a pipeline that processes the images into regular sizes. After that, the neural network model design will be carried out, including proposing the network type, the network structure, the choices of activation functions, training methods and training regularization options etc. The network will be designed from small models to large models to avoid model underperforming or over-training. The network will be trained, validated and tested using different data set, to achieve robust model performance, and out-of-sample validation.

The third step (see Chapter 4.2.4 - 4.2.6) is to perform an extra testing phase of the neural network to diagnose the model's performance. This is done by using image data that were not involved in the train-validate-test dataset, for example, pictures taken from mobile phones and pictures provided by Trafikverket. The extra testing phase will provide diagnostic results and the neural network and dataset will be improved based on the performance indicators and which images it failed to classify correctly. The extra testing is an attempt to identify scenes for which the network performs poorly and could benefit from additional training data. Once the extra testing phase is carried out successfully, the neural network model will be tested on videos for the demo. If the video test is completed, the model will be connected to a webcam live stream for the product showcase. The results and conclusions are given in Chapter 5, with some example images presented in the appendix.

Chapter 2

Data Collection and Selection

This section describes how the data was collected and processed for the neural network model training. Throughout the project, the images will be collected using the following methods:

1. Searching for images in Google Images.
2. Gather road camera/weather camera clips, based on the visibility measured at nearby weather stations equipped with a transmissometer.
3. Simulate images with fog using Unreal Engine 5 or other image processing based on Koschmieder's Law

2.1 Collecting data from Trafikverket

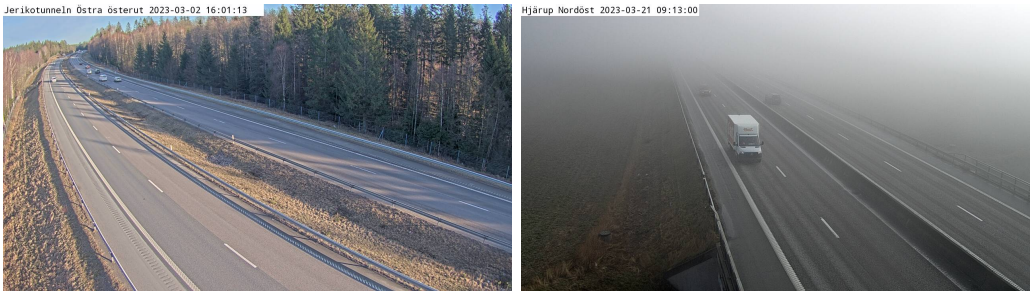
This subsection presents the data collection method for obtaining data via Trafikverket's Open-data API [17]. The primary interest is the availability of visibility data from weather stations, and road pictures taken from the the surveillance cameras nearby weather stations through the API.

By sending requests from a pre-made Python pipeline, one receives a JSON line that contains the visibility data and the weather station locations, which can be used to download road condition captures from nearby traffic cameras. However, not all Trafikverket's weather stations are equipped with transmissometers, resulting in missing visibility data for some weather stations, so one needs to detect these cases and implement appropriate error handling.

An example of available data is presented in Table 2.1 and Figure 2.1.

Table 2.1: Example of visibility output from the API

Weather Station Location	Visibility(m)
Persborg	8190
Stocksund	20000
Dannemora	2488
Bergfors	20000
Brottby	20000
Kullebo	10440
Luten	5377
Skulltorpsmotet	5173



(a) A clear photo with visibility > 2000m from trafikverket

(b) A foggy photo with visibility < 300m from trafikverket

Figure 2.1: Example of camera captures from Trafikverket's Database, note the information on the top left corner

Once the visibility data collection is done, the images will be gathered from the weather station (if applicable) and the road cameras close to the weather station. Images are 1920×1080 pixel in JPEG format, with information bars that usually sit on the top left. The bars will be cropped out in the image processing pipeline before being feed to the neural network.

However, it is not always possible to have foggy weather across Sweden, as the weather condition appears rather unlikely, and there are not many road cameras in the remote regions, which limits the availability of image data. To generate data, one needs to think of a way to simulate the fog directly in the image, which will be presented in the following subsection.

2.2 Simulating Images

The fog image simulation pipeline was made by Christos Sakaridis et al [14] and originally implemented in MATLAB. The workflow is to first take stereo images, with depth maps and camera data to generate transmission mappings. Finally, Koschmieder's Law is applied to simulate fog of different intensities in the images.

Let us assume that we have a non-foggy RGB image J to which a foggy effect should be added. The following formula shows the synthetics of the fog effect ([14]):

$$I(x,y) = \sum_{s}\sum_{m} t(x-s, y-m)J(s,m) \; + \; A \cdot (1 - t(x,y)) \tag{2.1}$$

The above equation represents a convolution operation between the image matrix and transmission mapping $t(x,y)$; (x,y) is the image coordinate in terms of pixels for the new images, $J(s,m)$ is each pixel within the original RGB image J, and A is white the atmospheric light represented by (255,255,255). Furthermore, the transmission mapping $t(x,y)$ depends on a distance mapping $d(x,y)$ according to the result of Koschmieder's Law:

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

Here, β is the atmospheric extinction coefficient. Its relation to meteorological visibility can be described as follows:

$$V = \frac{1}{\beta} \ln\left(\frac{1}{\epsilon}\right) \tag{2.3}$$

In equation (**??**), V is the meteorological visibility, β is the atmospheric extinction coefficient, and ϵ is a threshold constant set to 0.05. For the selection of β, we used $\beta = 0.002$, $\beta = 0.005$, $\beta = 0.01$ and $\beta = 0.02$ corresponding to roughly 1500, 600, 300 and 150 meters meteorological visibility.

The stereo-image dataset provided by Cityscapes Dataset [5], consists of 5000 images and corresponding distance mappings. The pipeline starts by using the depth map of the image to generate a transmission mapping for the images. Finally, by applying the transmit mapping to the pictures, the pipeline returns foggy images which will then be used for the training process.

(a) A light fog photo generated by the simulation pipeline, $\beta = 0.002$

(b) A medium fog photo generated by the simulation pipeline, $\beta = 0.004$

(c) A dense fog photo generated by the simulation pipeline, $\beta = 0.008$

Figure 2.2: Example of simulated pictures

2.3 Usage of Unreal Engine 5

Another effective method to generate foggy images is using Unreal Engine 5 [6]. The setup begins with downloading a map from the internet and then using a weather environment plugin called "Dynamic Weather", which has adjustable foggy weather parameters. Once the map is created and the weather parameters are set, the rendering pipeline will be designed. The rendering pipeline generates random render camera positions with random facing and then adds objects to the environment. Next, the fog parameter will be set, and finally, the capture will be rendered, labelled with the number of tasks and the fog parameter. The fog visibility can be determined before the rendering take place. It can be measured easily by looking at a black object in the scene, according the definition of visibility.

(a) An image generated with unreal engine 5, with fog intensity 1.

(b) Another image generated with unreal engine 5, with fog intensity 9.

Figure 2.3: Example of images generated by Unreal Engine 5.

Extra image test

The extra image set includes images taken by mobile phones or gathered from Google, which look different to those used in the training set. The extra validation images get labels manually and will pass through the same pre-processing pipeline described in 4.1.1. The accuracy of classification of this image set can represent the model's accuracy and performance in the real world, and the misclassified images can be evaluated and used for further correction and improvement of the training dataset.

Chapter 3

Theory: Machine Learning

3.1 The Artificial Neural Network

This section describes various components of neural networks including the choices of layers, activation functions and how performance can be evaluated during and after the training process. The actual design of the neural network models will be presented in the Analysis section (Chapter 4.2). The Python package TensorFlow [10] is used for all the neural network components and configurations.

3.1.1 Input layer

The input layer will take the image data, of either 128×128, 256×256 or 512×512 resolution, with normalized RGB intensities. This layer does not modify the image and only serves as the input that will be used for the deeper layers in the model.

3.1.2 Deep Feed-forward Neural Network

The deep feed-forward neural network or the Multi-Layer Perceptron (MLP) is one of the most basic neural networks that takes multiple data and extracts features via hidden layers consisting of a huge number of nodes. Each node is fully connected to the next layers via weights, with the last layer being the output layer, which gives a prediction based on the input image. The deep feed-forward neural network can effectively model non-linear patterns

in the data and works especially well for numerical data. Figure 3.1 shows a typical Deep Feed-forward Neural Network using an image as input:

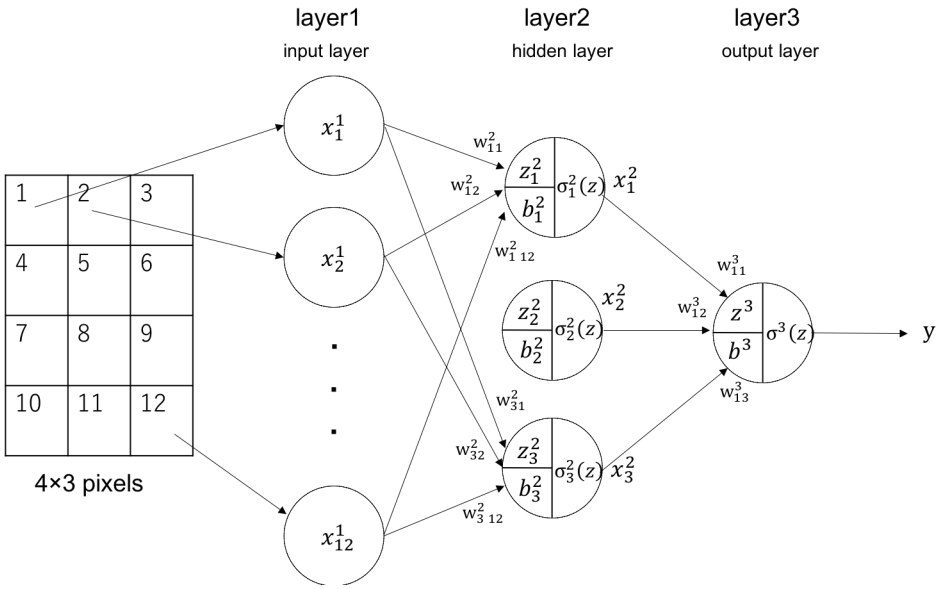


Figure 3.1: Example of using Deep Feed-forward neural network to classify data. Source: Internet

In Figure 3.1, x_n^1 represents the data input from 4×3 image pixels with indices n, z_m^L indicates the value of m'th node at layer L, b_m^L is the bias term adds to corresponding nodes z_m^L, σ_m^L is the activation function of node z_m^L and y is the output value computed by an activation function $\sigma^3(z)$ taking the argument of $z^3 + b^3$. Between the layers, there are weights w_{tf}^D represents the weight that are indexed such that the weight multiplies the value of a node with index $m = f$ and send to the node with index t in the next hidden layer D. Ultimately, according to the notations in Figure 3.1, the output of the MLP will be:

$$y = \sigma^3(b^3 + \sum_m w_m^3 \sigma_m^2(b_m^2 + \sum_n w_{m,n}^2 x_n))$$

MLP usually consists of multiple layers with different nodes; the number of output nodes will also differ depending on the number of classification categories. MLP usually deals well with numeric vector data as the nodes with activation functions serve as automatic feature extraction when training

the network. However, it usually struggles with image data as it could be inefficient to find the feature in 2 dimensions after flattening the image into a vector. In the next subsections, we will introduce another method which excels at image feature extraction, and it will become the main component of our project.

3.1.3 Convolutional Neural Network

The convolutional neural network (CNN) was introduced by the famous Deep Learning pioneer Yann LeCun and his coworkers in 1998 [9]. They introduced a convolution-based neuron operator that classifies hand-written numbers images, also known as the MNIST dataset, with high accuracy. The CNN consists of two parts: it uses a convolution kernel to extract features from the image and create feature maps. After the convolution step, the maps will be flattened and passed through a fully-connected MLP, which returns the classification decision at the output nodes.

The convolution layer performs matrix convolution over the input images. The layer uses convolution with a square kernel to produce feature maps. For example, the convolution using a 3×3 kernel will be:

$$\boldsymbol{C} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \circledast \boldsymbol{I}$$

Here, $\boldsymbol{I}$ is an image from the previous layer and $\boldsymbol{C}$ is the feature map output after the convolution step. Figure 3.2 shows the mechanism of the convolution kernel convolving a matrix into a feature map (output) as an example.

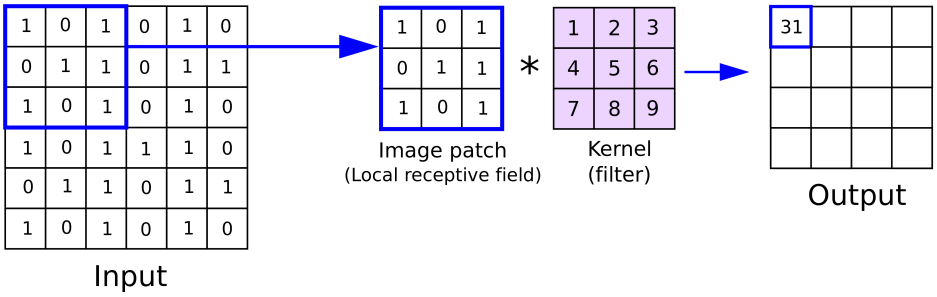


Figure 3.2: Illustration of the convolution layer's mechanism. Author: Anh H. Reynolds

Between each image patch and the corresponding feature map components, there will be weights connecting to each other that will change during the training process. Besides the regular weights, the convolution layer has additional parameters, such as padding that loops edge values on the image boundaries which could enhance the performance to classify certain features. To reduce the output dimension and improve the training speed, the "strides" can skip a certain number of pixels during the convolution, resulting in a smaller feature map. The feature maps will be expanded into multiple channels and fed into the fully connected layers. At the end of the convolution layers, the activation function will be applied to each component in the feature map.

In order to "highlight" some important features from the feature map, max pooling will be used.

3.1.4 Max-pooling

To make the features "stand-out" more, one could perform a process called "pooling". Hence, Max-Pooling is used to enhance image edge features. Figure 3.3 presents the mechanism of Max-Pooling convolutes a matrix as an example:

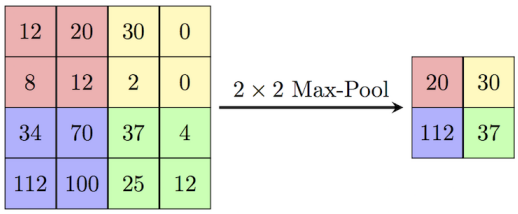


Figure 3.3: An example of 2D-Maxpooling using 2×2 kernel. The red output number 20, for example, comes from the maximum value of the red matrix in the input.

At this point, the feature map will be flattened, which is now ready to be fed into an MLP for feature analysis. In the next subsection, the choices of activation functions will be described, for those between hidden layers and for the output layer in our neural network model.

3.1.5 Choices of activation functions

Each layer in the neural network structure should have non-linear activation functions that allow the neural network to model complex relationships in the data. The Rectified Linear Unit (ReLU) is commonly used since it serves as a threshold similar to an "on/off switch". It is defined as:

$$f(x) = \max(0, x)$$

Hence, any mode with value x will be itself if x is positive, and 0 otherwise. Appending ReLU between structure layers helps regularize the training process and feature recognition. In the output layer, there will be two choices of activation function: The Logistic/sigmoid activation function or the Softmax activation function. The sigmoid activation function is:

$$f(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}, \tag{3.1}$$

The above function is typically used for binary classification since it takes values between 0 and 1 and is differentiable.

For the classification of multiple classes, the sigmoid would give different probabilities for each class and determine which class is most likely may be hard due to the lack of connection between probabilities, a better alternative,

is the Softmax activation function. The Softmax has the following expression:

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \tag{3.2}$$

where K is the number of classes and x_i are output from the K nodes in the final layer. Softmax normalizes each prediction x_i by their sum in the denominator so $\sigma(x_i)$ will become numbers between 0 and 1 that also sum to 1, making multi-class classification possible.

For our network, we will mainly be using ReLU activation functions between the convolution and max-pooling layers, as well as between the fully connected hidden layers. At the end of the MLP network, the sigmoid function will be used for the binary classification model and the Softmax function will be used for categorical classification.

With all the settings chosen for the neural network model design, the model will be compiled and it will undergo training as described in Chapter 4.2. The model will fit its weights to the training data (see Ch 3.3) and validate itself with the validation data. After each training epoch, performance indicators will be provided. The indicators show how the network performs as well as any abnormalities in network behaviour. The performance analysis will be presented in 3.3.2.

3.2 Edge detection algorithms

To improve image features, principle component analysis of images will be carried out. If one compares an image without fog and another image with dense fog, one should notice that image edges diminish for the image with fog. In contrast, the images without fog preserve clear outlines for many objects. The observation motivated us to use edge detection algorithms to detect the overall edge profile of images, and hopefully, the neural network models will learn from them.

This project uses the Sobel and Canny edge detection algorithm for outline sharpness detection based on image gradient computation. By doing this, the image data will be shrunken into single-channel arrays with significantly reduced dimensions while keeping most of the image features.

The Sobel edge detection algorithm is one of the most commonly-used methods in image analysis that computes the image intensity gradient, de-

veloped by Irwin Sobel and Gary Feldman in 1968 [15], inspired by Larry Roberts' "Robert's Cross" operator.

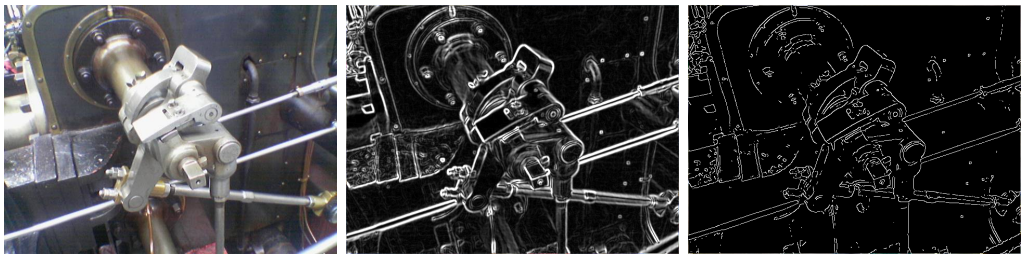
Generally speaking, the Sobel edge detection algorithm uses two convolution windows that compute the grey-scale image intensity derivative across the horizontal and vertical axis. Suppose P is the source image. The Sobel operator computes image gradients in the following manner:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \circledast \mathbf{A} \tag{3.3}$$

$$\mathbf{G}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \circledast \mathbf{A}, \tag{3.4}$$

where $\mathbf{G}_x$ and $\mathbf{G}_y$ are the images of derivative approximations across the horizontal and vertical directions.

An Alternative to the Sobel algorithm is the Canny edge detection [3].



(a) Original Image (b) Sobel Filtered Image (c) Canny Filtered Image

Figure 3.4: Images filtered by Sobel and Canny filters. Image source: Wikipedia

The Canny edge detector works similarly to the Sobel edge detector, it is also a convolutional kernel but it is a Gaussian Convolutional Kernel. The kernel has the following expression if the size is $(2k+1) \times (2k+1)$:

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}); 1 \leq i, j \leq (2k+1) \tag{3.5}$$

If we have $k = 5$ and $\sigma = 1$, we will then end up with the following kernel

and convolution operator:

$$B = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * A$$

the asterisk is the convolution operator and A is the original image.

When applying the Canny Image Filter with sigma parameter 3, the filter keeps fewer edge profiles for foggy images. In contrast, for the pictures with less fog or no fog, the filter keeps more edge profiles. This will be a significant feature to help the neural network. The filters will be used for image processing and will be used as image inputs to the CNN.

For this project, the Canny Image Filter will be used.

3.3 Loss and Gradient Descent Training

This section describes the loss functions selected and how they will be used for training of the neural network. Generally, the loss function represents the inaccuracy of the machine learning model predicting data labels based on its current choices of weights. To reduce the loss function, the model must adapt to the dataset by updating the weight. This process is called the "training" process, which is essentially an optimizing process. The main optimization process is based on gradient descent and several modifications will be presented in this section.

3.3.1 Cross-entropy Loss

Throughout this project, the Binary Cross-entropy loss and Categorical Cross-entropy loss were used:

$$L_{BCE}(\boldsymbol{\omega}; \boldsymbol{x}) = -\frac{1}{n}\sum_{1}^{n} y_i \cdot \log(\hat{y}_i(\boldsymbol{\omega}; x_i)) + (1 - y_i) \cdot \log(1 - \hat{y}_i(\boldsymbol{\omega}; x_i)), \quad (3.6)$$

$$L_{CCE}(\boldsymbol{\omega}; \boldsymbol{x}) = -\sum_{i=1}^{n}\sum_{k=1}^{C} t_{i,k} \log(p_{i,k}(\vec{\boldsymbol{\omega}}, \vec{x_i})) \quad (3.7)$$

In the Binary Cross-entropy loss function (3.6), n is the number of observations for which the network should produce predictions, y_i is the i'th true label and $\hat{y}_i(\boldsymbol{\omega}; \boldsymbol{x})$ is the i'th sigmoid prediction if the labels $\hat{y}$ is the probability of $y = 1$ from the neural network given weights $\boldsymbol{\omega}$ and the input data x_i corresponding to the i'th image. This corresponds to the negative log-likelihood of Bernoulli observations where $y_i \sim Be(p(\boldsymbol{\omega}, x_i))$

In the Categorical Cross-Entropy Loss Function (3.7), C is the number of classes, $t_{i,k}$ are true indications of class for the i'th observation where $t_{i,k} = 1$ for the correct class and 0 otherwise, and $p(\vec{\boldsymbol{\omega}}, \vec{x_i})$ are the softmax predictions of each class probability based on data $\vec{x_i}$

Using the true data and the loss function, the neural network model could adapt and update its weights to minimize the loss. Generally speaking, the choice for the weights $\boldsymbol{\omega}$ should satisfies the following:

$$\boldsymbol{\omega} = \arg\min_{\boldsymbol{\omega}} L(\boldsymbol{\omega}; \boldsymbol{x}),$$

This means that the neural network must adjust its weights $\boldsymbol{\omega}$ to find the global minimum of the loss function, which is known as training the weights.

Usually, there are plenty of training methods we could use, such as basic Stochastic Gradient Descent (SGD) learning:

$$\boldsymbol{\omega}_{t+1} := \boldsymbol{\omega}_t - \eta \nabla (L(\boldsymbol{\omega}_t; \boldsymbol{x})), \tag{3.8}$$

where t is the training epoch and η denotes the "learning rate" that controls how much the gradient $\nabla(L(\boldsymbol{\omega}; \boldsymbol{x}))$ affects the correct weights. SGD is efficient and easy to implement, which leaves plenty of room for the code and the neural network to be tuned. However, SGD requires some regularization to prevent the weights from wandering of at plateaus or being trapped at local minima. To prevent the training from encountering those problems, engineers have made several variations to the SGD. One commonly used variation of SGD is adding "momentum" to the gradient. The idea was introduced by Rumelhart, David E., Hinton, Geoffrey E. and Williams, Ronald J. in 1986 during an experiment with a back-propagation learning procedure described in [13]. In 2016, Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton presented the linear combination of momentum coefficient to the SGD weight update (3.8) in their work [16]. In their words, momentum-augmented SGD learning has the following schematics:

$$\begin{cases} \boldsymbol{v}_{t+1} = \mu \boldsymbol{v}_t - \epsilon \nabla L(\boldsymbol{\omega}_t) \\ \boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t - \eta(\boldsymbol{\omega}_t) + \alpha \Delta \boldsymbol{\omega}_t \end{cases}$$

where α, μ are the momentum coefficient and $\boldsymbol{v}_t$ is the an average of past gradient direction.

The second modification to SGD is called Root Mean Square Propagation (RMSProp) and regularizes the learning rate via a function. The recursion is:

$$\boldsymbol{\omega} := \boldsymbol{\omega} - \frac{\eta}{\sqrt{\vec{v}(\omega, t)}} \nabla(L(\boldsymbol{\omega}; \boldsymbol{x})), \tag{3.9}$$

where

$$\vec{v}(\boldsymbol{\omega}, t) = \gamma \vec{v}(\boldsymbol{\omega}, t-1) + (1-\gamma)|\nabla L(\boldsymbol{\omega}; \boldsymbol{x})|_2^2.$$

and γ is called the forgetting rate. RMSprop performs well for non-convex optimization problems with the help of the moving average of the squared gradients giving faster convergence speed.

Finally, Adaptive Moment Estimation (ADAM) is one of the most commonly used modifications of SGD, it makes use of both mechanisms of the RMSProp and Momentum method, together with other parameters, making itself an efficient training method. ADAM has the following parameter recursion:

- $m_{\boldsymbol{\omega}}^{(t+1)} = \beta_1 m_{\boldsymbol{\omega}}^{(t)} + (1-\beta_1)\nabla^{(t)}(L(\boldsymbol{\omega}; \boldsymbol{x}))$
- $v_{\boldsymbol{\omega}}^{(t+1)} = \beta_2 v_{\boldsymbol{\omega}}^{(t)} + (1-\beta_2)(\nabla^{(t)}(L(\boldsymbol{\omega}; \boldsymbol{x})))^2$
- $\hat{m}_{\boldsymbol{\omega}} = \frac{m_{\boldsymbol{\omega}}^{(t+1)}}{1-\beta_1^t}$
- $\hat{v}_{\boldsymbol{\omega}} = \frac{m_{\boldsymbol{\omega}}^{(t+1)}}{1-\beta_2^t}$
- $\boldsymbol{\omega}^{(t+1)} \leftarrow \boldsymbol{\omega}^{(t)} - \eta \frac{\hat{m}_{\boldsymbol{\omega}}}{\sqrt{\hat{v}_{\boldsymbol{\omega}}} + \epsilon}$

Where β_1 is the forgetting factor for the gradients and β_2 is the forgetting factor for the second moments of the gradient, with β_1^t and β_2^t the beta parameters to the power of epochs. The tiny (1×10^{-8}) number ϵ is applied to avoid division by zero.

When training neural network models, it is common that over-training effect will occur as the training epochs increase due to the neural network learning too much from the training dataset. To reduce the over-training, one could perform L2-regularization to suppress weight developments when training, or one could perform Branch drop-out which, with a certain probability between 0 and 1, randomly removes weight branches between the layers

to make room for generalization. In this project, we use the drop-out to regularize over neural networks in training because drop-out works very well when the neural network model has many trainable weights.

3.3.2 Result of training

The performance indicators of the model include training, validation and test accuracy. The training accuracy and validation accuracy together tell how well the model is training: If the validation accuracy starts to fall too far behind the training accuracy, the model is overtrained. If validation accuracy is better than testing accuracy, then it is considered undertrained.

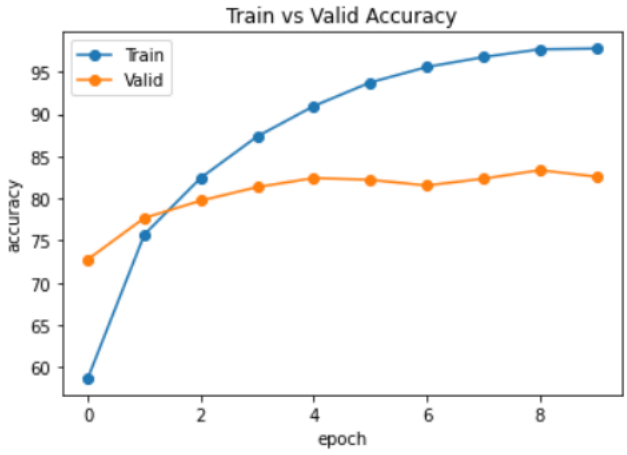


Figure 3.5: A plot of training (blue) and validation (orange) accuracy against training epochs of an overfitted model

Over-trained models are not good as they lose generalization when classifying images. To avoid over-training, one should apply a method called "early-stopping" to detect increasing validation loss and stop the training, this method is provided by Keras ([10]).

However, sometimes it is not enough to value the high validation accuracy. To obtain a robust model, one must keep a dataset that is not involved in the training process and use it to test the model. This process serves as a sanity test as some models could learn the wrong feature to achieve a high validation accuracy. This test dataset will be used after the training is

finished and serves as a sanity check. The performance indicators such as accuracy and F1-score will be the main focuses.

Consider an image classification result with True Positive (TP, given a picture belongs to the test label and the model classifies that picture belongs to the test label), True Negative (TN, given a picture *does not* belong to the test label and the model classifies that picture *does not* belong to the test label), False Positive (FP, given a picture belongs to the test label *but* the model classifies that picture *does not* belong to the test label) and False Negative (FN, given a picture *does not* belong to the test label *but* the model classifies that picture belongs to the test label) numbers, the accuracy is defined as:

$$\text{Acc} = \frac{TP + TN}{n}$$

where n is the total number of test images. The accuracy gives a straightforward overall performance indication, but it gets less sensitive to inaccuracy when the test dataset is huge and unbalanced as presented in [12]. To avoid losing sensitivity, the F1-score is chosen for the thesis instead of accuracy. The F1 score presents a combination measure of how the model makes correct decision is and how it classifies positive cases for all the classes. The F1 score is defined as:

$$F_1 = \frac{2 \cdot TP}{2TP + FP + FN}$$

Chapter 4

Methods

Now that the foundation has been laid, let us describe the training and implementation of a reliable Artificial Neural Network for detection of reduced visibility. Output from the network will be used to trigger a defogging filter in the camera.

This section describes the pre-processing of the dataset, training and the validation of the network.

4.1 Overview

4.1.1 Image processing pipeline

The original images before the pre-processing can include watermarks or have different resolutions, which will be handled by an image pre-processing pipeline before the database creation. Because it will be impossible for the neural network to execute the training if the input images have different resolutions, and the watermark will influence the features learned by the neural network model.

4.1.2 Training

To give the neural network model a robust ability to classify images, the training process will carefully proceed and will be mainly discussed in the later chapter. However, it is hard to validate the dataset quality since there was no data set initially. Therefore, an important step is to retrain the neural

networks with increased dataset amount consisting of new images targeted to the probablistic training result after a few training epochs.

At the same time, To validate the actual performance of a neural network, it is essential to focus on its test accuracy. The test accuracy can be provided from the testing over the test set which is made using the Python function *train_test_split*() from the Scikit-Learn package. But the result of the test set generated by train-test-split cannot fully represent the neural network performance in the real world. Therefore, an extra validation method based on a completely different validation set is required to represent the models' performance in real-life scenarios.

4.1.3 Validation process

Because the final neural network will be used on web cameras, it is suitable to validate the model directly on webcams. This is achieved by letting the neural network classify images from the web camera and monitor the results manually. By examining the performance from a web camera, one can respond to the actual performance and accuracy of the neural network in the real world.

Although testing on a real webcam seems reasonable, it also has a drawback since foggy weather is rare in the real world. To solve the issue, foggy scenery videos can be used to validate the neural networks and they can be downloaded from the internet.

On the other hand, an extra data set of images taken from mobile phones can also be used to validate the neural networks further. They can also represent the actual performance and accuracy of the network in real life because the images are independent from the dataset used to train the model.

4.2 Analysis

4.2.1 Data set and image processing pipeline

The dataset used to train and validate the models in this project consists of several photos with and without fog. As described in Chapter 2, the source of the images includes the internet, Trafikverket, simulation from Unreal Engine 5, Foggy Cityscape Dataset and photos from mobile phones. The data set is divided into two parts, one for the binary classification and another for

the multi-class classification of visibility. For the first part, the accurate value of visibility is not required. Therefore, this set includes the images from the internet and mobile phones, the images from Trafikverket. In the second part of the project, the multi-classes classification requires values of visibility and images from Trafikverket, and image with simulated fog are used. The simulation parameters can be converted into a visibility metric. In the end, the images will be loaded and will be labelled as "clear" (Vis > 2000m), "lighter fog" (2000m > Vis > 1000m), "medium fog" (1000m > Vis > 400m) or "dense fog" (400m > Vis) in one-hot encoding for the multi-label classification task.

The original images have different resolutions and proportions, hence it requires an image pre-processing pipeline to correct the images to a unified format before training. The first step of the pipeline is to check the aspect ratio of images and then crop two square images along the longer side using the package PIL in Python [18]. The second step is to resize the squares to the largest resolutions and mirror one of the two squares horizontally. Finally, these two images are given identity numbers and class indices and stored in folders of processed images.

4.2.2 Training the Neural Network Model

Included packages and setup

For training, the dataset will then be split into three parts: 60% of the data will be the training data, 20% will be the validation data, and the remaining 20% will be used for testing. The motivation is to ensure that the neural network model undergoes sufficient training while allowing for rigorous testing and validation.

Similar to conventional statistical parameter estimation, the training process for the neural network model is based on the error (loss) function and the goal is to minimize the loss function using the function gradient to update the network weights. Once the dataset is established, the structure design of the neural network is carried out.

Structure design

Ultimately we have created the model structures for the binary classifiers presented in Table 4.1 and the model structures for the 4-class classifiers

presented in Table 4.2:

Table 4.1: Binary ANN structure (poolsz = maxpooling size, strd = stride)

	convx1-2-128p	covx1-256p	covx1-512p
Input size	$128 \times 128 \times 3$	$256 \times 256 \times 3$	$512 \times 512 \times 3$
Conv 1	128, 5×5, poolsz = 2	128, 5×5, poolsz = 2, strd = 2	128, 7×7, poolsz = 2, strd = 2
Conv 2	128, 3×3, poolsz = 2	128, 5×5, poolsz = 2, strd = 2	160, 5×5, poolsz = 2, strd = 2
Conv 3	256, 3×3, poolsz = 2	256, 3×3, poolsz = 2	192, 5×5, poolsz = 2, strd = 2
Conv 4	256, 3×3, poolsz = 2	256, 3×3, poolsz = 2	256, 3×3, poolsz = 2
Conv 5	/	/	/
Conv 6	/	/	/
dense 1	1024, dropout = 0.25	1024, dropout = 0.2	1024, dropout = 0.2
dense 2	128, dropout = 0.2	256, dropout = 0.2	256, dropout = 0.2
dense 3	/	/	/
dense 4	/	/	/
add-on	/	/	/
output	1, sigmoid	1, sigmoid	1, sigmoid

Table 4.2: 4-class ANN structure (poolsz = maxpooling size, strd = stride)

	convx1-2-128p-4c	covx1-256p-4c	covx1-512p-4c
Input size	$128 \times 128 \times 3$	$256 \times 256 \times 3$	$512 \times 512 \times 3$
Conv 1	128, 5×5, poolsz = 2	128, 5×5, poolsz = 2, strd = 2	128, 5×5, maxpool = 2, strd= 2
Conv 2	128, 3×3, poolsz = 2	128, 3×3, poolsz = 2	128, 5×5, poolsz = 2, strd = 2
Conv 3	128, 3×3, poolsz = 2	128, 3×3, poolsz = 2	128, 3×3, poolsz = 2
Conv 4	256, 3×3, poolsz = 2	256, 3×3, poolsz = 2	256, 3×3, poolsz = 2
Conv 5	256, 3×3, poolsz = 2	256, 3×3, poolsz = 2	256, 3×3, poolsz = 2
Conv 6	/	/	/
dense 1	1024, dropout = 0.25	1024, dropout = 0.25	1024, dropout = 0.25
dense 2	128, dropout = 0.2	128, dropout = 0.2	128, dropout = 0.2
dense 3	/	/	/
dense 4	/	/	/
add-on	/	/	/
output	4, Softmax	4, Softmax	4, Softmax

These neural networks will then be applied to adjust the defogging filter once the model is trained and considered satisfactory. The tweaking procedure will be presented in the next section.

4.2.3 Tweaks to the networks

In this section, the neural network model mechanism tweaking will be described and motivated. Overall, the model will be tweaked according to the test results, such as the performance index (F1-score, accuracy, precision and recall) and the type of misclassified images. The reason for tweaking the models can be the insufficiency of the current training set, which can not cover every situation of foggy weather in the real world. The prediction performance could also be affected by faulty images in the current training set, which can decrease the accuracy, or cause incorrectness in the hyperparameters of the model.

The whole tweaking process includes the following: first, validate the model with an extra validation process, like a video test, camera test or the extra image set test; second, adjust the hyperparameters and validate the new models again; third, update the current training set according to the result of each validation.

4.2.4 Validations

As described before, the trained neural networks will validate the model with a test set independent of the training set. But this validation cannot represent the performance and the accuracy of the model in real life because it only represents the performance and the accuracy of a limited dataset. Therefore the extra validation stages will be taken, including testing over an extra image set, video streams and a camera test.

The extra image dataset consists of images taken from a mobile phone and is not involved in the training dataset. The video test will be performed by reading video stream captures into the neural network model, using the OpenCV-Python [1] package for capture. The neural network predictions will be manually verified. Results from the neural network will be diagnosed to see if the neural network is working well or not. In the following subsections, the details will be explained.

Camera test

The camera test is a test script based on Python. At the same time, the script can also control the camera defog filter by sending requests, that change the parameters in the camera according to Table 4.3.

Table 4.3: Web camera's defogging filter parameters

Variable name	default	explain
Defog	false	turn on/off the defog filter.
Defog effect	0	Effect of the defog filter, max 100.

The selected web camera has a defogging filter implemented, and the strength of the filter can be chosen as an integer number between 0 and 100. The camera test script will try to boost up the sampling speed and the effect of the defog filter when it detects fog in the image until the video stream is no longer foggy. Then, it will try to decrease the effect of the defog filter to check if the fog has cleared. If the model finds fog in the video stream again, the value of defog effect will increase according to the model's decision.

Video test

The video test script works similarly to the camera test, but it loads the video from a local device and then converts it into a video stream. It takes images from the video stream at a fixed interval with a certain rate and then lets the model classify the captured images. The whole process will be monitored to evaluate model performance.

Once we have evaluated and retrieved the performance indicators of the models, we will carry out model adjustments as described in section 4.2.5 below.

4.2.5 Adjustment

The adjustment of the models includes changing the hyperparameters or the structure of the networks. The hyperparameters include the learning rate, MaxPooling kernel size, the number of filters and the size of the kernel of the convolution layers and the number of nodes in the hidden layer. For the overall structure, the number of convolution and hidden layers and the chosen optimizer will also be adjusted. According to the theory, the same

hyperparameters are not 100% suitable for every similar network. Therefore, it is vital to get the specific and most suitable hyperparameter for each model. For example, the learning rate should be increased when the network converges too slowly, and the dropout should be increased when overfitting.

4.2.6 Tweaks and retrain

Once the adjustment has been established, the model needs to retrain and the data set needs to be edited to address any errors in the extra validation procedure. Such as adding more images to the training dataset which are similar to the misclassified images. This action will fill the data set with the missing scenarios of foggy weather in the real world and will hopefully improve the model's performance.

With all the model tweaks and retraining, the model will be further verified by training it from scratch 20 times to verify reliability. After several iterations of this procedure, the results have been gathered and will be presented in the next chapter.

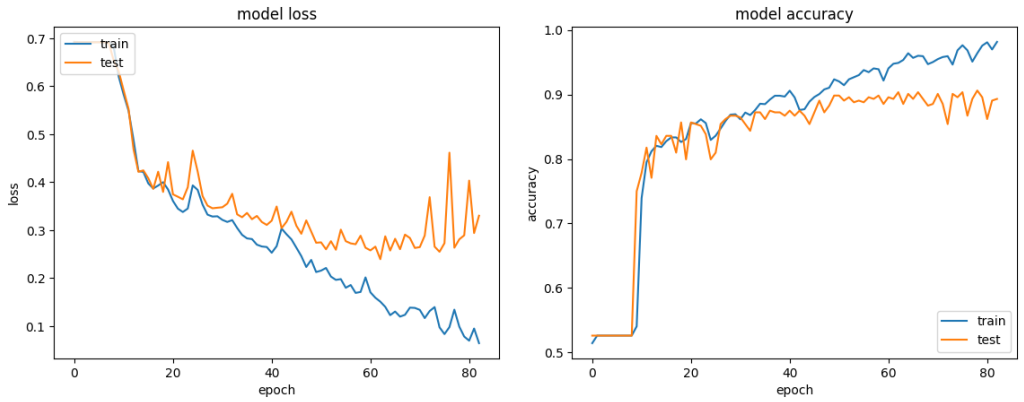
Chapter 5

Experiment Results

This chapter will present the results of the most important models developed during this master thesis work, and the models will be represented by their identification number. All the test results are generated by the confusion table from the SciKit-Learn package.

5.1 Binary Classification Neural Network

Record of one training run and shown in Figure 5.1. The blue line in Figure 5.1a represents the loss under training, and the orange line represents loss for validation data under training. And the blue line in Figure 5.1b presents the accuracy under training, and the orange line presents the accuracy of the validation data under training.



(a) The loss for both training and validation data during a training.

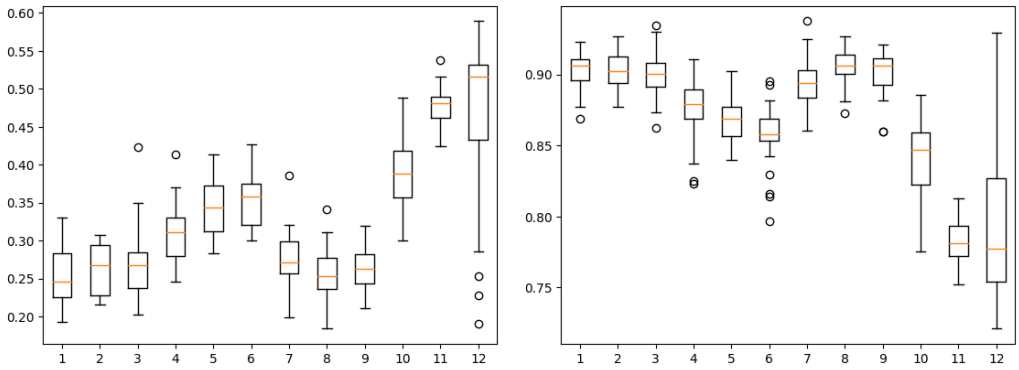
(b) The accuracy for both training and validation data during a training.

Figure 5.1: The training records of binary model.

The relation between each model and its identity number are presented in Table 5.1, and box plots of every result of the models are shown in Figure 5.2. The losses are presented in Figure 5.5a and the accuracy are presented in 5.5b. At the same time, the table of every model's average and median are shown in Table 5.2. Below is the table of all binary classification models we have experimented with:

Table 5.1: The table of identify number to model

No.	Input	Size	Structure
1	RGB	128P	4 Normal convolution layers
2	RGB	256P	4 Normal convolution layers
3	RGB	512P	4 Normal convolution layers
4	G+canny	128P	4 Normal convolution layers
5	G+canny	256P	4 Normal convolution layers
6	G+canny	512P	4 Normal convolution layers
7	RGB	128P	5 Normal convolution layers
8	RGB	256P	5 Normal convolution layers
9	RGB	512P	5 Normal convolution layers
10	G+canny	256P	5 Normal convolution layers
11	RGB	32P	No convolution 7 hidden layers
12	RGB+histogram	256p	4 convolution layers



(a) The losses of all models (b) The accuracy of all models

Figure 5.2: Boxplots of testing's losses and accuracy for all models, the definition of the models is presents in Table 5.1.

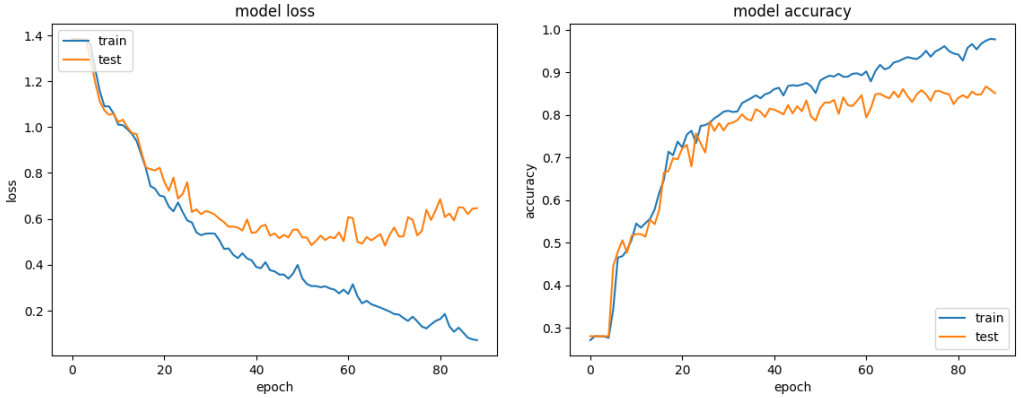
The statistic of every models performance in Table 5.1 presents in Figure 5.2, and best results of every terms are shown in bold.

Table 5.2: Average, median and best results of testing and the result of extra test of the binary classification models, note the best results of every term are highlighted in bold.

ID	Average	Median	Best	Extra
1	0.902	**0.923**	0.906	0.967
2	**0.903**	0.902	0.927	0.967
3	0.901	0.900	**0.934**	**0.978**
4	0.875	0.879	0.910	0.956
5	0.869	0.869	0.902	0.945
6	0.856	0.858	0.895	0.956
7	0.893	0.894	0.938	0.967
8	0.904	0.906	0.927	0.967
9	0.901	0.906	0.921	0.967
10	0.838	0.847	0.885	0.945
11	0.783	0.781	0.813	0.846
12	0.798	0.777	0.929	0.923

5.2 4-class Classification Neural Network

Record of one training run of the 4-class model is shown in Figure 5.3. The blue line in Figure 5.3a presents the total loss under training, and the orange line presents the total loss to the validation test under training. And the blue line in Figure 5.3b presents the total accuracy under training, and the orange line presents the total accuracy to the validation test under training.



(a) The loss of both the train set and test set during a training.

(b) The accuracy for both the train set and test set during a training.

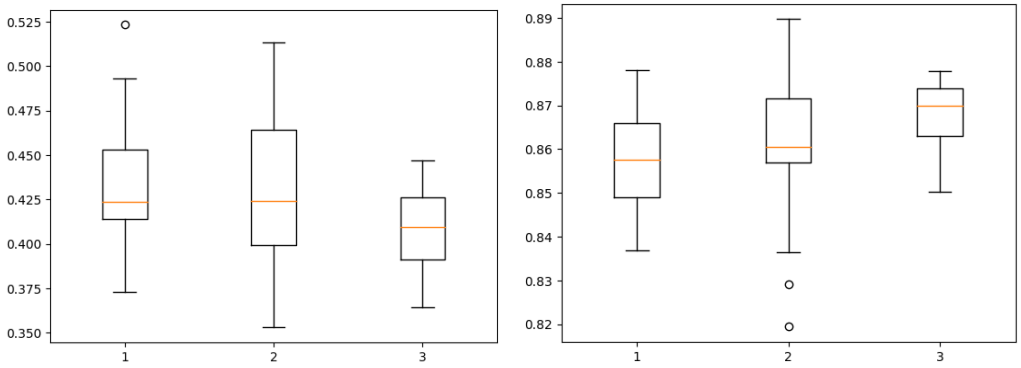
Figure 5.3: The training records of a 4-classes model.

The table below shows the information for all 4-class classification models which have been used in the experiment:

Table 5.3: The table of identification numbers for the models

No.	Input	Size	Structure
1	RGB	128P	5 Normal convolution layers
2	RGB	256P	5 Normal convolution layers
3	RGB	512P	5 Normal convolution layers

The statistical information of both accuracy and loss of 20 training intervals of models is shown in Figure 5.4, and more details of the result show in Table 5.4.



(a) The losses of all 4 classes models (b) The accuracy of all 4 classes models

Figure 5.4: The result of the 4 classes models, the first one has the size 128, the second has 256 and the third has 512.

Table 5.4: Average, median and best results of testing and the result of extra test of the 4-class classification models, note the best results of every term are highlighted in bold.

Model No.	1	2	3
average	0.857	0.860	**0.867**
median	0.857	0.861	**0.870**
best	0.878	0.890	0.878
extra testing	**0.570**	0.557	0.468

Table 5.5: The classification result of the model with 128P input, note the best results of every term are highlighted in bold.

	precision	recall	f1-score	number
0	0.86	**0.92**	0.89	**232**
1	0.87	0.86	0.87	184
2	0.87	0.82	0.85	202
3	**0.94**	0.88	**0.91**	210

Table 5.6: The classification result of the model with 256P input, note the best results of every term are highlighted in bold.

	precision	recall	f1-score	number
0	0.88	**0.92**	**0.90**	**231**
1	**0.92**	0.87	0.89	183
2	0.86	0.85	0.86	202
3	**0.92**	0.88	**0.90**	209

Table 5.7: The classification result of the model with 512P input, note the best results of every term are highlighted in bold.

	precision	recall	f1-score	number
0	**0.92**	0.87	0.90	**198**
1	0.85	0.85	0.85	182
2	0.88	0.80	0.84	**198**
3	**0.92**	**0.90**	**0.91**	183

5.3 Model Infeasability

The following images in Figure 5.5 were extremely difficult for the binary-classify models to predict, These images were used in the extra testing dataset.

(a) An example of a clear image which was predicted as foggy.

(b) An example of a foggy image which was predicted as clear.

Figure 5.5: Example of the wrongs during the experiment, for the binary classification networks.

5.4 Results and Discussion

5.4.1 Results for the binary classification model

Models using raw RGB images

From the result, it could be seen that the overall performance of the binary classification neural network using raw RGB input was high, yielding an overall accuracy of over 90%. Among the raw-RGB input models, the one using 512p images (no.3 from Table 5.2) yields the highest accuracy. Meanwhile, the one using 256p and 128p image inputs are ranked 2nd and 3rd. This is very likely due to the higher amount of information provided by the higher dimension images, which allows the model to make more accurate predictions. The convolutional neural network works much better than model 11, consisting of 7 fully-connected layers, which only yielded 78% accuracy. None the less, model 7 yields better results compared to the model presented in [12]

From the models with 5 convolution layers (model number 7,8,9), it seems that an additional layer has little effect on the model's performance. Still, it enhances the performance of the models that use smaller image input dimensions (such as model 7). The reason for yielding the lower average accuracy is probably due to the one additional convolution layer making the model too powerful, so it overtrained during the training process and hence loses a certain portion of generality.

Models using Canny filtered images

Compared to the models with raw RGB images, the neural network model using Canny filter images yielded slightly lower prediction accuracy, as the image processing step could have removed important feature information which could be caused by a high sigma value set for the filter. Despite this, the memory footprint was smaller than the models taking in raw RGB images. Those models could make themselves easier to implement in compact camera products. It can also be seen that an extra convolution layer has made this model too powerful and has over-fitted to the data.

There are other attempts, such as using the Sobel image filter but the

idea quickly became unfeasible as the filtered image quality was bad. Pre-filtering using 2D-FFT was also ineffective. By far the most effective filter method is the Canny image filter.

Models using raw RGB image and RGB Histogram

The binary neural network that classifies RGB Histogram was unsuccessful(model 12 in Table 4.1), as it only yielded 79% average accuracy. From this, we conclude that using RGB Histogram as the feature to classify foggy pictures seems to be a failure. Interestingly, in one training epoch, the model obtained 93% accuracy, but the results were not repeatable. The main reason is because of the RGB histogram algorithm takes all pixels as a group and erases spatial information.

5.4.2 Results of the 4-class classification model

All the 4-class classification models performed well, with an average accuracy of 0.857, 0.860 and 0.873 respectively. However, the models did not match our expectations as they only yielded F1-scores of 0.55, 0.57 and 0.468. The worst result was obtained from the model with 512p resolution that failed to converge. The inferior accuracy of the models during the extra testing phase is possibly caused by undiscovered image feature differences between the training dataset and the extra testing dataset. This might be resolved in future works as the dataset gathering would be rather time-consuming.

In terms of the generated images, some images had poor generation quality which could potentially corrupt the models' feature extraction, such as the similarity of the overall colour tone and the scenery of the captures. The improvement for future works is to generate more images with better image sources and more efficient algorithms or to find and use more real-life images rather than simulated images.

However, there exist pictures mentioned in section **??** that the neural network could not correctly classify, which are pictures with tree branches or any object close to the camera that has a significantly foggy background. The problem could be caused by the weakness of convolutional layers that could not perceive spatial differences of the object.

5.5 Conclusions and Possible Future work

Throughout this project, we have explored the possibilities of applying Artificial Neural Network models for image-based fog detection and using generated images for neural network model training. The result shows the approach has great potential and could be developed further from this project or could inspire other tasks.

From the results, we concluded that the CNN with RGB image inputs with the highest acceptable resolution is the best combination to classify fog accurately. The other CNN models with alternative input sizes and all other CNN models with image feature input have inferior performance compared to the CNN with 512p image input size. Nevertheless, the CNN models yielded better results than the Deep Feed-forward neural network approach used in [12].

When preparing the data, the data quality was mostly fine. The MATLAB image generation pipeline worked great, except sometimes it generated strange fog patterns in the image. This indicates a drastic difference between MATLAB and Unreal engine. The images rendered from Unreal Engine 5 were also quite successful. For the model 2nd-round validation, we employed the strategy of adding images similar to the misclassified images to the training dataset. The neural network efficiency improved and it led the neural network to classify more images accurately. And yet, the process should be repeated for future work to strengthen the model performance and generalization.

The neural network model based on Gray-scale images and the Canny image filter performed well, yet they have lower overall test accuracy compared to the CNN models with RGB inputs. It is likely due to several different reasons, mainly due to the loss of image information caused by the reduction of spatial information, or the improper parameter settings. This could be improved by alternating or combining multiple image feature extraction methods to process the image data. However, as shown in this project, it is a valid solution to recommend against using the RGB histogram method.

Finally, the 4-class classification neural network was a possible attempt

to classify visibility reduction, but it failed miserably as the model did not survive the extra image testing phase, which is likely due to the problem with the mismatched dataset features. In future works, one should improve the dataset quality by selecting more realistic camera captures and increasing its size and balance.

Many future works could be developed, or inspired by this project. As one used the neural network to detect aerosol-caused visibility reduction based on the captured images, one could also make a neural network model that could detect smoke in indoor or outdoor environments so one could employ the model for webcam-based fire safety, for example. Similar approaches could also apply to other types of visibility reductions such as detecting fog forming on the camera dome and detecting sandstorms or snowstorms.

Bibliography

[1] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[2] Sebastian Bronte, Luis Bergasa, and Pablo Fernández Alcantarilla. "Fog Detection System Based on Computer Vision Techniques". In: Nov. 2009, pp. 1–6. DOI: 10.1109/ITSC.2009.5309842.

[3] John Canny. "A Computational Approach To Edge Detection". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* PAMI-8 (Dec. 1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[4] Hazar Chaabani et al. "A Neural network approach to visibility range estimation under foggy weather conditions". In: *Procedia Computer Science* 113 (2017). The 8th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2017) / The 7th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2017) / Affiliated Workshops, pp. 466–471. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2017.08.304. URL: https://www.sciencedirect.com/science/article/pii/S1877050917317131.

[5] Marius Cordts et al. *The Cityscapes Dataset for Semantic Urban Scene Understanding*. 2016. arXiv: 1604.01685 [cs.CV].

[6] Epic Games. *Unreal Engine*. Version 4.22.1. Apr. 25, 2019. URL: https://www.unrealengine.com.

[7] Kaiming He, Jian Sun, and Xiaoou Tang. "Single image haze removal using dark channel prior". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 1956–1963. DOI: 10.1109/CVPR.2009.5206515.

[8] P V.C. Hough. “Method and means for recognizing complex patterns”. In: (Dec. 1962).

[9] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[10] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[11] Mihai Negru and Sergiu Nedevschi. “Image based fog detection and visibility estimation for driving assistance systems”. In: Sept. 2013, pp. 163–168. ISBN: 978-1-4799-1493-7. DOI: 10.1109/ICCP.2013.6646102.

[12] Giuliano Andrea Pagani, Wiel Wauben, and Jan Willem Noteboom. “Neural network approach for automatic fog detection using surveillance camera images”. In: EGU General Assembly Conference Abstracts (Apr. 2018), p. 2988.

[13] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[14] Christos Sakaridis, Dengxin Dai, and Luc Van Gool. “Semantic Foggy Scene Understanding with Synthetic Data”. In: *International Journal of Computer Vision* 126.9 (Sept. 2018), pp. 973–992. URL: https://doi.org/10.1007/s11263-018-1072-8.

[15] Irwin Sobel. “An Isotropic 3x3 Image Gradient Operator”. In: *Presentation at Stanford A.I. Project 1968* (Feb. 2014).

[16] Ilya Sutskever et al. “On the importance of initialization and momentum in deep learning”. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. URL: https://proceedings.mlr.press/v28/sutskever13.html.

[17] *Trafikverkets öppna API för trafikinformation*. URL: https://api.trafikinfo.trafikverket.se/.

[18] P Umesh. “Image Processing in Python”. In: *CSI Communications* 23 (2012).

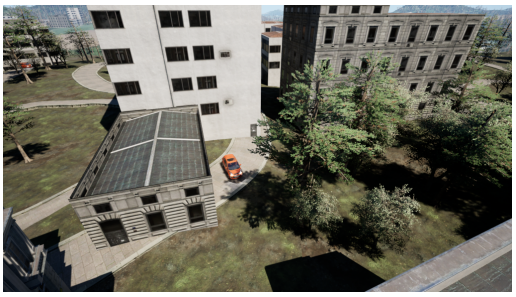
[19] Wiel M. F. Wauben and Martin Roth. “Exploration of fog detection and visibility estimation from camera images”. In: (2016).

Appendix A

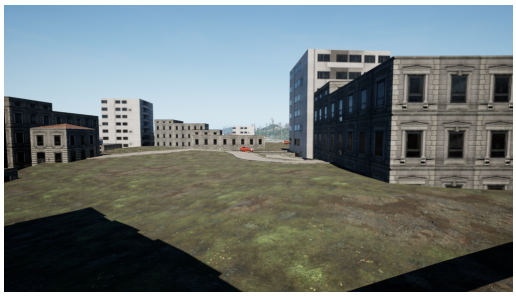
Images

This appendix is an album of the images which were used/generated under the experiment.

A.1 Simulated images from Unreal Engine 5



(a) Generated image from UE5, with no fog.

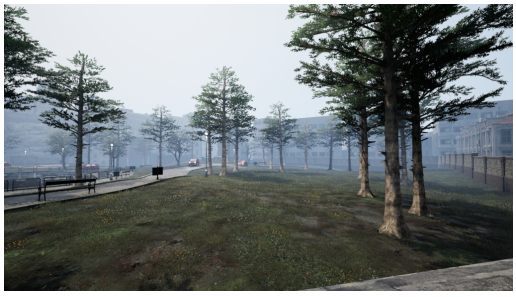


(b) Another generated image from UE5, also with no fog.

(c) Generated image from UE5, with light fog.

(d) Another generated image from UE5, also with light fog.

(e) Generated image from UE5, with medium fog.



(f) Another generated image from UE5, also with medium fog.

(g) Generated image from UE5, with dense fog.

(h) Another generated image from UE5, also with dense fog.

Figure A.1: More images generated from the UE5 rendering pipeline, took the picture from random places with random perspectives including the scenery of road, parking lot, buildings and parks, with 4 classes of visibility

A.2 Simulated images from Foggy_Cityscape

(a) An image in the middle of an alley, with light fog.

(b) Another image in the middle of an avenue, also with light fog.

Figure A.2: Example of an original image which was used in Foggy_Cityscape.

(a) The same picture after the processing, dense fog.

(b) Another image in the middle of an avenue, also with light fog.

Figure A.3: Example of images that are generated of Foggy_Cityscape

Master's Theses in Mathematical Sciences 2023:E48
ISSN 1404-6342

LUNFMS-3120-2023

Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/