HAW HAMBURG

Bachelor Thesis

Hussam Kayed

Development of a method for deep learning based meteorological visibility conditions prediction

Faculty of Computer Science and Engineering
Department of Information and Electrical Engineering

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Hussam Kayed

Development of a method for deep learning based meteorological visibility conditions prediction

Bachelor Thesis based on the examination and study regulations
for the Bachelor of Engineering degree programme
Bachelor of Science Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg
Supervising examiner: Prof. Dr. Jörg Dahlkemper
Second examiner: Prof. Dr. Lutz Leutelt

Day of delivery: 22 Juni 2023

Hussam Kayed

Title of Thesis

Development of a method for deep learning based meteorological visibility conditions prediction

Keywords

CNN, VGG16, DenseNet201, Adam

Abstract

This paper discusses how a new way of meteorological and visibility conditions using deep learning methods is to be developed. The development strategy of the solution involves investigating previous research and related work and establishing a foundation of knowledge. In addition to that, a certain set of requirements will be set clear to establish the boundaries for the Convolutional Neural Networks (CNN) model to be created. Furthermore, the architectures and the development strategy will be analysed and the created model will be enhanced to increase its efficiency.

Hussam Kayed

Thema der Arbeit

Entwicklung einer Methode zur Vorhersage von meteorologischen Sichtbedingungen basierend auf Deep Learning

Stichworte

CNN, VGG16, DenseNet201, Adam

Kurzzusammenfassung

Dieses Papier diskutiert, wie eine neue Methode zur Vorhersage von meteorologischen und Sichtbedingungen mit Hilfe von Deep-Learning-Methoden entwickelt werden soll. Die Entwicklungsstrategie der Lösung beinhaltet die Untersuchung früherer Forschungen und verwandter Arbeiten und die Schaffung einer Wissensgrundlage. Zusätzlich dazu wird ein

bestimmter Satz von Anforderungen klar festgelegt, um die Grenzen für das zu erstellende Convolutional Neural Networks (CNN) Modell zu definieren. Darüber hinaus werden die Architekturen und die Entwicklungsstrategie analysiert und das erstellte Modell wird verbessert, um seine Effizienz zu steigern.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

In the past years, the most dangerous life threatening accidents which took place on the road involved a certain meteorological phenomenon which made it for fellow drivers all the more dangerous to be on the road at such a time. This meteorological phenomenon is fog and the reason it causes such accidents is due to the decreased visibility along the road. There has been several communication protocols established in modern vehicles in order to avoid accidents in such weather conditions such as V2V, I2V and I2I which would be installed in Road Side Units in the assistance of avoiding such accidents. In this thesis, a new approach is taken to analyze the change meteorological activities and detect whether it's safe to drive in the detected level of fogginess.

1.2 Fog Definition

Fog is a meteorological phenomenon characterized by the presence of minute water droplets or ice crystals suspended in the air near the Earth's surface, resulting in reduced visibility. This hydrometeor typically forms when the temperature and dew point converge, and sufficient condensation nuclei are available for water vapor to condense. As a natural occurrence with multifaceted implications, fog influences various aspects of human activities, ecological processes, and environmental quality. Some also may define fog as a cloud touching the ground, this also makes sense from the visibility point of view. The study of fog, its formation mechanisms, physical properties, and diverse impacts, contributes to a comprehensive understanding of atmospheric processes and informs effective forecasting, mitigation, and management strategies.

1.3 Goal

The goal of this thesis is to deliver a new approach through deep learning where one could use any webcam image in order to identify the level of fogginess and the visibility conditions for a normal human being. There are going to be different approaches investigation in which my approach will be compared to the existing approaches to solve this problem. The approach of how I came to choose the architecture will be also thoroughly investigated. In the end, the chosen architecture will be put into the test using a data set created specifically for this purpose.

1.4 Topic Structure

The following aspects of the topic are going to be discussed thoroughly.

- Fundamentals: In this chapter, the basics of machine learning will be clarified as well as investigating previous researches where the theories, implications, analysis and methods are to be examined, summarized and compared to one another and identify which gaps in the research exists to fully understand the approach followed.
- Conception: In this chapter, after taking all the research done in the previous chapter into consideration, a conception of this development approach is to be created where different solutions are to be put in trial and a decision will be made based on a specific results criteria.
- Development and Implementation: this is the chapter where the chosen architecture for the implementation will be used in order to derive the best solution possible to create a deep learning model to carry out the required functionality which is detection of the meteorological visibility conditions.
- Test and Evaluation: after the model was trained and implemented, the time comes for it to be tested against a whole new data set created solely for the purpose of testing the trained model. The accuracy of the prediction is going to be compared to the accuracy of the trained model.
- Conclusion: In this section, a consensus shall be reached and the previous chapters will be summarized to give an overview of the whole paper.

- Bibliography: This is the section where all the sources will be cited.

2 Fundamentals

In this chapter, the fundamentals of machine learning, neural networks, convolutional neural networks are going to be explained. The following explanation is essential where the latest technologies and state of the art research papers will be investigated and analyzed.

2.1 Machine Learning Basics

Machine learning is a subfield of artificial intelligence that focuses on the development of algorithms and statistical models, enabling computer systems to automatically learn and improve their performance in various tasks without explicit programming [31].

2.1.1 Machine Learning Strategies

In this part, different complex strategies will be explained. Three of which are Supervised Learning, Unsupervised Learning and Semi-supervised learning. What is essential for the scope of the paper is the supervised learning in which Neural Networks and Convolutional Neural Networks will be using it in the following sections.

2.1.2 Supervised Learning

In consonance with [30], Supervised Learning is a task where the machine maps an input to an output for the sake of learning a function that later can be applied to a similar input with the output being the prediction of the machine based on the input. This infers a function from labelled training data consisting of a set of training examples. In figure 2.1, one can see the exact workflow of supervised learning where the input is usually divided into two sets, **training data set** and **test data set**. The training data set is

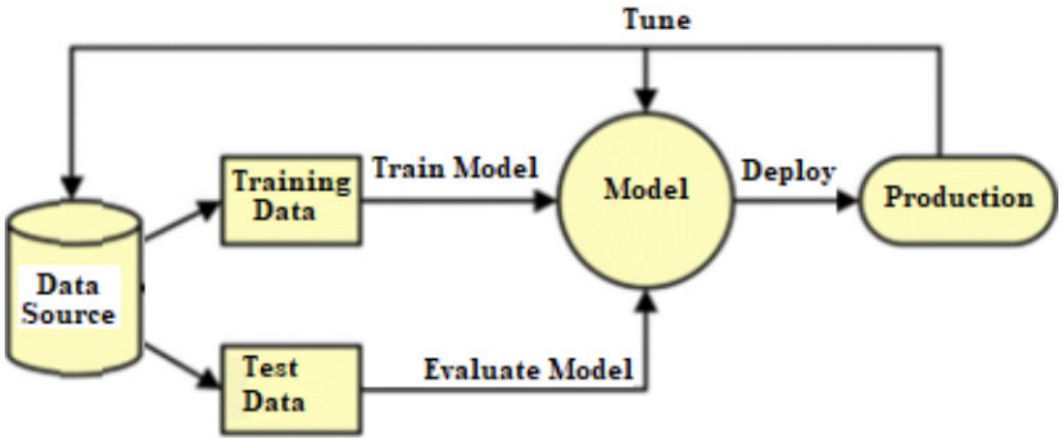


Figure 2.1: Supervised Learning Workflow [30]

used for training the model in order to have a model in the end which can be deployed into production. The test data set is used for the validation of the model i.e. that the model is trained using the training data set and tested against the test data set in order to make sure that the model's output predictions can be accurate enough or reach at least the same accuracy reached in the training phase. In case the accuracy of the model is not satisfying to be deployed, the hyper parameters of the model are fine tuned to increase the accuracy of the model, so the model is retrained on the same data set after fine tuning. The same process keeps happening over and over until the accuracy of the model is satisfying the requirements of the production phase.

K-Nearest Neighbours (KNN)

K-nearest neighbours (KNN) is a non-parametric, instance-based supervised learning algorithm used for classification and regression tasks. It is non-parametric because it does not make assumptions about the underlying data distribution, and instance-based because it does not explicitly learn a model but instead uses the training data set directly for predictions.

2.1.3 Unsupervised Learning

Unsupervised learning is a method for the machine to learn by itself, which means there is no supervision from a human using any input/output description or classification at

all. This implies that the algorithms discover and present new structures in the data by themselves. On introducing new data to the trained model, the previously learnt features are used to recognize and classify the data [18].

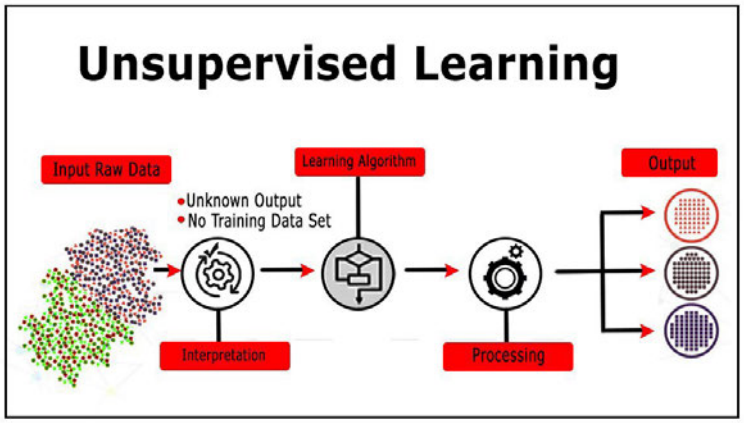


Figure 2.2: Unsupervised Learning Workflow [30]

In figure 2.2, one can see the full cycle of training where the input is fed to the machine with no classification. The machine starts interpreting the data based on patterns and previously learnt features if there's any using a learning algorithm which will be later examined. The machine classifies the input into different classes as many as specified in the learning algorithm. The prediction can always be refined based on how many times a model is trained or how big the raw input data set that is fed to the machine is. This sort of machine learning algorithm is used for clustering and feature reduction [30].

2.1.4 Semi-Supervised Learning

In this section, semi-supervised learning will be explained.

Semi-supervised learning (SSL) [8] is a machine learning paradigm that lies between supervised learning and unsupervised learning. It leverages both labeled and unlabeled data to train models, making it particularly useful when labeled data is scarce or cannot be obtained easily. Semi-supervised learning can improve model performance compared to using only labeled as in supervised learning or only unlabeled data as in unsupervised learning.

- **Data**: The data in SSL is a mix between labeled data where the input and the output are fed to the machine in pairs and unlabeled data which have only input

data. The labeled data is much less in the amount compared to the unlabeled data since labeled data is typically harder to acquire while unlabeled data are easily attainable.

- **Goal**: The goal of SSL [16] is to use the information in the unlabeled data to improve the model's performance on the labeled data, thereby achieving better generalization to unseen instances.
- **Assumptions**: There are two main assumptions, on which SSL relies which are:
  - Continuity Assumption [39]: this assumes that data points which are close together in feature space are most likely inhabiting similar traits, hence having similar outputs as well. Leveraging this assumption, the model can imply the outputs of unlabeled data points based on their proximity to labeled data points, if found nearby.
  - Cluster Assumption [16]: Clustering is already a technique used in unsupervised learning in K-Means Clustering. The same technique is used in SSL with added twist of having already labeled data in the clusters of the feature space, so the model doesn't have to learn anything from scratch yet it's able to classify the unlabeled data under the assumption of the existence of the unlabeled data within the same cluster as the labeled ones.
- **Algorithms**: There is a multitude of algorithms developed such as: **Self Training**.

2.2 Neural Networks

A neural network [15] is a computing system inspired by the biological neural networks constituting animal brains. These systems progressively improve their ability to perform tasks by considering examples, generally without task-specific programming. They are based on a collection of connected nodes or 'neurons', aka 'artificial neurons' or 'nodes.' Hence, neural networks are a variety of deep learning.

2.2.1 Deep Learning

Deep learning [15] is a sub field of machine learning that focuses on learning successive layers of increasingly meaningful representations from data. The term "deep" refers to the

fact that modern deep learning models can involve many layers of representation which are learned automatically from exposure to training data. This is in contrast to other machine learning approaches that typically only learn one or two layers of representation, and are therefore referred to as "shallow learning".

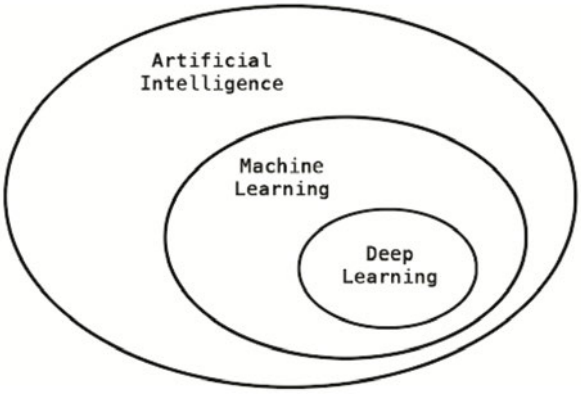


Figure 2.3: Classification of how machine learning is classified with respect to artificial intelligence and how deep learning is classified with respect to machine learning [10]

In deep learning, the goal is to build a model that can automatically learn to extract relevant features from raw data, such as images, sound, or text. This is accomplished by training a neural network composed of many interconnected nodes, or neurons, that pass information to each other in a way that allows the network to learn and generalize from the data. The neural network is typically composed of multiple layers, with each layer responsible for learning a different level of representation of the data.

Layers

The fundamental data structure in neural networks, as conformed in [10], is the layer. A layer is a module that processes data and can take one or more input tensors, and output one or more tensors. Some layers are stateless, meaning they do not have any memory, but more often layers have a state that consists of the layer's "weights". These weights are one or several tensors that are learned using stochastic gradient descent, and they contain the knowledge of the network. In other words, the weights represent the network's learned representation of the input data, and they are what allow the network to make accurate predictions on new data [10].

One of the simplest neural networks algorithms which illustrates how a neural network is layered is the *Perceptron* [32].

Perceptron

The perceptron was introduced by Frank Rosenblatt [38] in 1957. He proposed the Perceptron as an algorithm for supervised learning of binary classifiers which is, as the name implies, a type of model which is trained to classify data into one of two possible categories, commonly represented as binary labels (i.e. true or false, positive or negative, 0 or 1). This algorithm allows individual neurons to learn and process elements in the training set sequentially, one at a time.

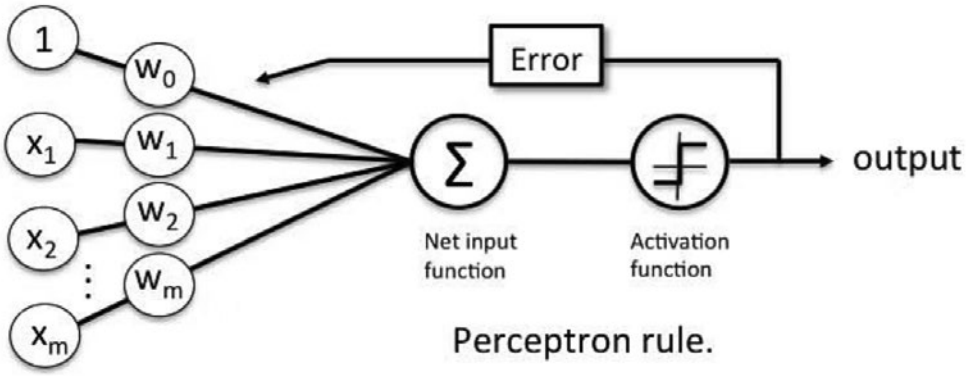


Figure 2.4: Clarification of the Percepton network [35]

In figure 2.4, the layers of the neural network are visually represented where:

1. Input Layer: The input layer consists of one or more input neurons, which receive input signals from the external world or from other layers of the neural network. Each input neuron is associated with a weight, which represents the strength of the connection between the input neuron and the output neuron.

2. Activation Function (hidden layer): The perceptron's output is determined by the activation function, which takes into account the weighted sum of the inputs and the bias term. Various activation functions can be used in perceptrons, including the step function, sigmoid function, and ReLU function.

3. Output Layer: The output of the perceptron is a single binary value, either 0 or 1 since it is considered a binary classifier, which indicates the class or category to which the input data belongs.

2.2.2 Data Augmentation

Augmentation is changing the nature of the element that is being used in an experiment. Similarly, data augmentation would be changing the nature of the data in order to suit the neural networks architecture and improve the performance. In some cases, it can be also considered as a way to normalize the data for the purpose of increasing the performance and reduce over-fitting.

Based on the type of data, there are different techniques of augmentation which can be carried out on the data:

- Images: images can be augmented in a manner of changing the characteristics of the image, some of which entail:
  - Rotation
  - Scaling
  - Flipping (changing the orientation to horizontal or vertical)
  - Translation (shifting the image)
  - Shearing
  - Brightness
  - Contrast
  - Adding noise (robustness tests)
  - Cropping (excluding irrelevant information)
  - Padding
- Text
- Audio

In essence, data augmentation is a crucial method for improving the performance of neural networks, especially when faced with scarce or skewed data. By creating a variety of new samples, data augmentation mitigates over-fitting, increases model robustness, and ultimately results in superior normalization [15].

2.3 Convolutional Neural Networks (CNN)

Convolutional networks [15], also known as convolutional neural networks or CNNs, are a specialized kind of neural network for processing data that has a known, grid-like topology. Examples include time-series data, which can be thought of as a 1D grid taking samples at regular time intervals, and image data, which can be thought of as a 2D grid of pixels. The name "convolutional neural network" indicates that the network employs a mathematical operation called **convolution**.

2.3.1 Convolution

Convolution [15] is a mathematical operation that is fundamental to many common image processing operators. In the context of a convolutional neural network (CNN), a convolution involves the application of a filter or kernel to an input image to produce a feature map that represents specific aspects of the image.

A convolution formula, in the manner of a 2D example, would look like the following:

Convolution operation in a 2D image is represented as follows [15]:

$$(I * K)(i, j) = \sum_{m} \sum_{n} I(i - m, j - n)K(m, n)$$

Where:

- I is the input image,
- K is the kernel or the filter,
- i and j are the spatial dimensions (coordinates) in the image,
- m and n iterate over the kernel dimensions,

- $I(i-m, j-n)K(m,n)$ is the element-wise multiplication of the image patch and the kernel,
- The result, $(I * K)(i,j)$, is the sum of these multiplications, which gives the convolution result at the location (i,j) in the output feature map.

The symbol $*$ represents the convolution operation. The process involves sliding the kernel K across the image I, and at each location (i,j), we calculate the sum of the element-wise multiplication of the image patch under the kernel and the kernel itself. This gives us a new value at the same location (i,j) in the output feature map.

The convolution operation captures the local dependencies in the original image. Depending on what the filter K looks like, it can detect different types of features in I, such as edges, corners, or other types of texture.

2.3.2 CNN Usage

Convolutiona Neural Networks are used for the following:

- **Feature Maps** [15]: Feature maps, also known as activation maps, are intermediate outputs in a Convolutional Neural Network (CNN) that represent the spatial arrangement of learned features at a given layer. They are generated through the application of convolution, activation functions, and pooling operations to the input data or to the outputs of previous layers in the network.

  In a CNN, the input data (usually an image) is passed through a series of layers, each designed to learn and extract increasingly complex and abstract features. These features can range from simple edges and textures to more complex structures [10] like object parts or entire objects. Feature maps serve as a way to visualize and understand the information retained and processed by the network at each layer.

  A feature map is obtained by applying a set of learnable filters to the input data, followed by an activation function, such as ReLU (Rectified Linear Unit). The filters are designed to detect specific patterns in the data, and as they slide (or convolve) across the input, they generate a response in the form of a feature map. The activation function introduces non-linearity into the network, allowing it to learn more complex features.

- **Pooling**: A pooling layer in a Convolutional Neural Network (CNN) is responsible for reducing the spatial dimensions of feature maps while preserving their essential information. Pooling layers help to make the network more computationally efficient and invariant to small transformations in the input data, such as translation, rotation, or scaling. This invariance contributes to the model's robustness and generalization capability when dealing with real-world data.

  Pooling layers are typically placed between consecutive convolutional layers in a CNN. The most common types of pooling operations are:

  1. **Max Pooling** : In max pooling, a sliding window (also called a pooling kernel) moves across the feature map, and the maximum value within the window is selected as the representative value for that region. This operation effectively down-samples the feature map, reducing its spatial dimensions while retaining the most prominent features.

  2. **Average Pooling**: In average pooling, the sliding window moves across the feature map, and the average value of all elements within the window is selected as the representative value for that region. Average pooling also down-samples the feature map, but it retains more information about the overall structure of the features compared to max pooling.

  3. **Global Pooling**: In global pooling, the entire feature map is considered as a single window, and a pooling operation (usually max or average) is applied to it. This operation reduces the spatial dimensions of the feature map to 1x1, effectively summarizing the entire feature map into a single value. Global pooling is often used before the final classification layer in a CNN.

  For the previously mentioned types of pooling, max pooling has been always the most favorable of all due to the fact that one can extract the highest value of the feature map when one uses it [15].

- **Flattening**: A flattening layer in a CNN serves as a bridge between the convolutional and fully connected (dense) layers of the network. Its primary purpose is to convert the multidimensional feature maps generated by the convolutional and pooling layers into a one-dimensional vector. This transformation enables the output of the convolutional layers to be fed into the fully connected layers for further processing, such as classification or regression tasks [15].

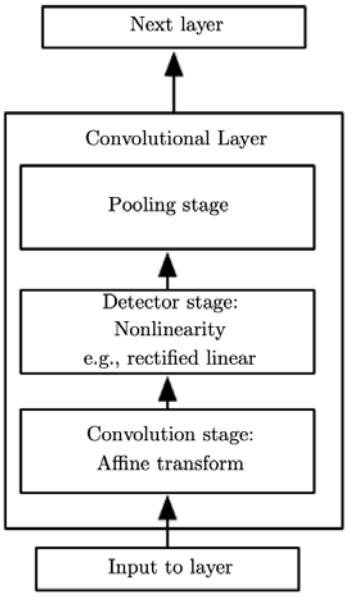


Figure 2.5: The components of a typical convolutional neural network layer with a generalisation of the convolutional layers, the activation function layer (ReLU) and the pooling stage with the next layer (a generic term for flattening layers and fully connected layers) [15]

In figure 2.6, the data are flowing from bottom to top where the first layer is the conventional input layer. This layer is usually resized based on the CNN architecture needs.

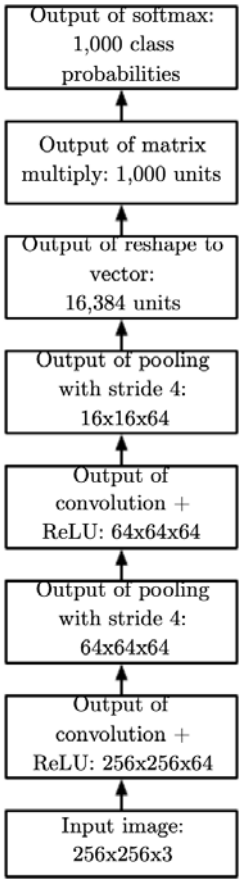


Figure 2.6: A convolutional network that processes a fixed image size. After alternating between convolution and pooling for a few layers, the tensor for the convolutional feature map is reshaped to flatten out the spatial dimensions [15]

2.4 State of The Art

In this section, the existing research on deep learning methods to detect meteorological visibility conditions will be analysed with a specific criteria of comparison.

- Approach: The approach will be analyzed, whether it was a neural networks or normal classification approach.
- Architecture: The chosen architecture will be specifically mentioned and explained.

- Training and Validation Accuracy: The accuracy of the training and validation data sets will be analysed and compared to those of the other researches.
- Results and Conclusions: The final results will be mentioned to understand how the research concluded.

At the end of this section, an overview of the CNN architectures used in previous researches and a distinction of what and machine learning classification approaches will be formed.

2.4.1 Approaches

The following approaches are used to identify meteorological visibility:

- Human visibility estimation based approaches: It is mentioned that only human experts are supposed to be making the estimations of how the visibility distance between objects in the fog. Normal humans tend to overestimate the distance between cars in the fog.
- Visiometer-Based Equipment Methods: they mentioned that these visiometers usually fall into two categories which are: Transmissometers or Scatterometers [4].
    1. Transmissometers: Transmissometers evaluate the average transmissivity (as shown in figure 2.7) of the atmosphere along a chosen, linear route. To accomplish this, they project a concentrated and collimated laser (with luminous flux energy ϕ_s) onto a narrow field of view (FOV) receiver at a specified measurement distance d. By analyzing the received luminous flux energy (ϕ_r), the device can determine the path extinction coefficient k [4].

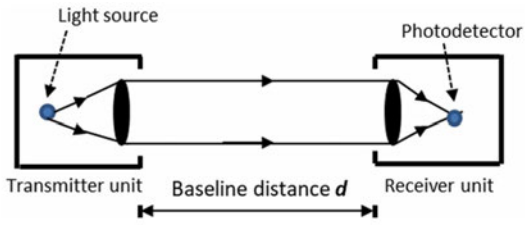


Figure 2.7: Transmissometer [4]

While transmissometers offer dependable readings, they frequently face criticism due to their substantial operating expenses, vulnerability to sunlight, and the intricate process of aligning the optical emitter and receiver.

2. Scatterometers: Optical scatterometers, also known as diffusometers, comprise a light source (commonly a laser) and a receiver, which are positioned at a specific angle and directed toward a shared region (referred to as the sample volume) [29].

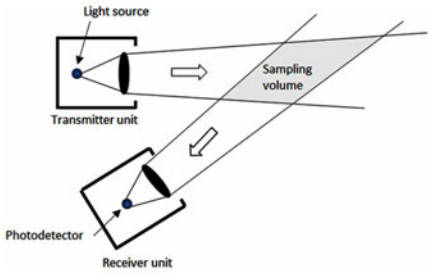


Figure 2.8: Forward Scatterometer [4]

While scatterometers are more cost-effective than transmissometers and do not necessitate alignment between the light source and detector, they come with several drawbacks, such as heightened sensitivity to fog non-uniformities, reduced accuracy in dense fog conditions, and significant sensitivity to motion also noted that backscatter fog detectors perform inadequately near water bodies, as reflections from the water surface can potentially lead to erroneous visibility measurements.

- Camera-Based Visibility Range Estimation Methods [4]: there has been an increasing interest in using affordable fixed cameras, already installed along major highways for traffic monitoring, security, and surveillance, to estimate visibility range. Cameras are easier to install onboard vehicles compared to optical fog sensors, but their use for visibility estimation in foggy conditions has been less explored, especially for onboard cameras without a reference image. A key challenge for camera-based methods is recovering lost depth information from a single image. A systematic review of existing literature identified gaps in classification and coverage of visibility distance estimation methods, leading to the development of a comprehensive taxonomy for daytime visibility distance estimation techniques in foggy conditions. When selecting the most appropriate visibility distance estimation

approach, researchers need to decide among various alternatives and constraints. These include for instance:

1. The choice of acting on the whole image or on a specific target
2. The availability of data with a reference visibility distance
3. The availability of a map distance
4. The availability of suitable mathematical model or the need to build a new one

Camera Estimations can be classified into two main categories as illustrated in figure 2.9:

1. Target based methods: These can be further categorized into those approaches that require camera calibration and those that do not.
2. Overall image-based methods: These can be further classified into two categories depending on the type of approach used to derive the mapping function between the scene's contrast and the visibility distance which are
   a) The correlation/linear regression approach
   b) The classification approach

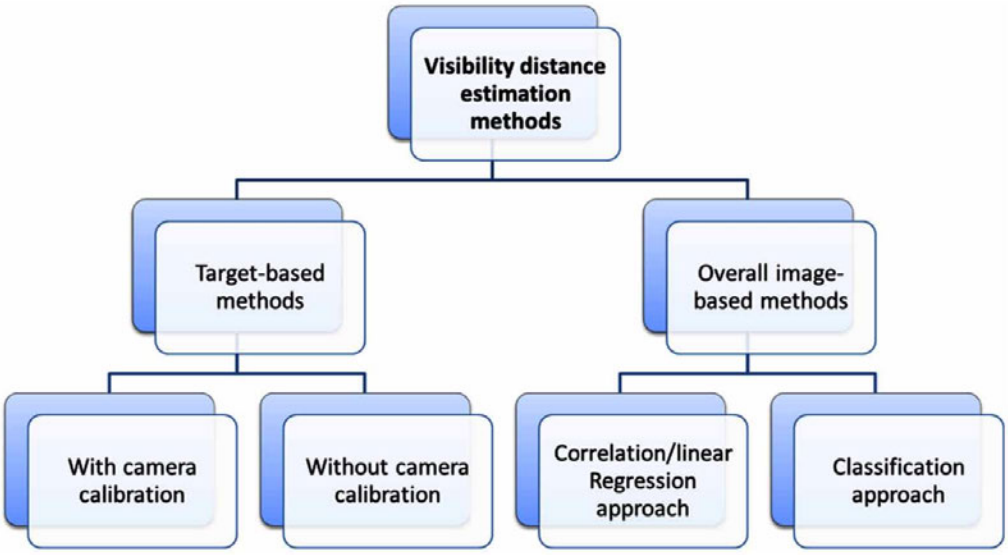


Figure 2.9: High-level classification of camera-based visibility distance estimation methods [4]

Expanding on that, the target-based approach for estimating visibility range has been applied to both fixed cameras with reference images and onboard cameras, such as the "RALPH" system. In other scenarios, edge detection techniques are common for fixed-camera applications, while other methods are used for onboard cameras to extract depth information from dynamic backgrounds. Target-based approaches rely on the relationship between image contrast and gradient, particularly in foggy conditions. Various aspects of these approaches differ, including edge detection techniques, descriptors, target points (region) of interest, and target localization techniques. Some examples of target localization techniques include object detection, which focuses on detecting known targets like road signs or road-sky intersections, and image segmentation.

Some of the above mentioned techniques were used to estimate the distance and five approaches have been proposed using those techniques (as shown in figure 2.10):

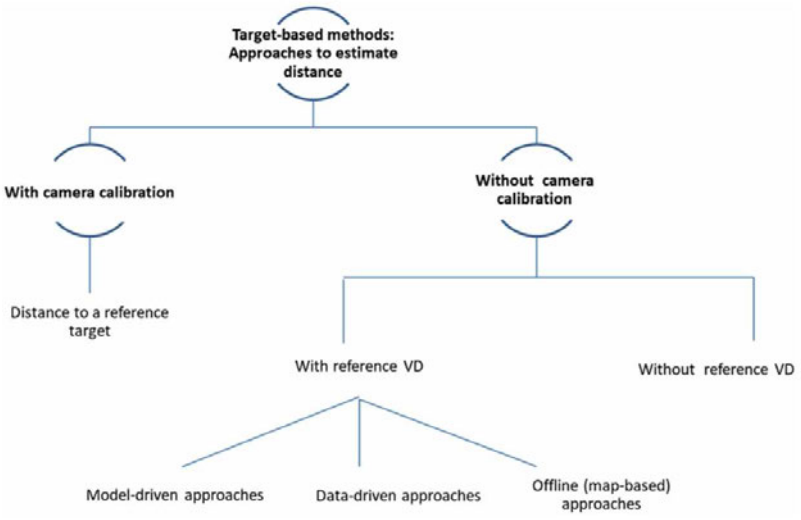


Figure 2.10: Distance estimation methods approaches based on target-based methods [4]

Distance estimation approaches based on Target-Based Methods:

- Case 1: **With Camera Calibration**
  In the case of methods requiring camera calibration, these approaches calculate distance to a reference axis or point. Examples include a horizontal line dividing the road and sky, a bandwidth axis, a vanishing point, or the farthest point with a contrast below 5%. The distance is then converted from image coordinates to real-world coordinates.

- Case 2: **Without Camera Calibration**
  For methods that do not rely on the availability of a reference visibility distance, Pomerleau (1997) suggested an approach to determine the relative visibility distance. This method involves measuring the scaled contrast attenuation between consistent road features at different distances in front of a moving vehicle.

- **Model-Driven Approaches**: model-driven approach involves identifying the most suitable probabilistic contrast distribution in a scene among a set of basic models (e.g., Uniform, Exponential, Rayleigh, Gaussian, or Weibull distributions) and estimating the associated parameters. This approach relies on fitting a model based

on the scene's contrast distribution. These are the main steps to be outlined in model-driven approaches [4]:

1. Choose appropriate mathematical distributions from hypothesized families of distributions, such as Uniform, Exponential, Rayleigh, Gaussian, or Weibull distributions.
2. Estimate the parameters using classical methods like moments matching or maximum likelihood.
3. Compute the errors to find the best-fitting distribution for the data.
4. Evaluate the quality of fit using classical methods such as R^2, F-test, and Root Mean Square Error (RMSE), which are based on the Sum of Squares Total (SST) and Sum of Squares Error (SSE).
5. Assess the goodness of fit using plots and statistics to estimate the discrepancy between observed and expected data.

In several studies, authors locate Lambertian surfaces, compute gradients using the Sobel filter, and fit the sum of contrast to visibility provided by a visibility meter. They found that an exponential distribution is the most suitable mathematical model. However, a key limitation of model-driven approaches is that the image's contrast distribution may not follow a known model, making these approaches inapplicable in some situations [4].

- **Data-Driven Approaches**: The data-driven (or non-parametric modeling) [4] approach involves applying learning algorithms to derive a function that best fits the computed scene's contrast data. This approach consists of two steps: (1) computing target indicators using classical methods like Sobel, homomorphic, or high-pass filters, and (2) correlating the contrast value with the visual range estimated by an additional reference meteorological sensor. This approach does not require accurate geometric camera calibration or a high-contrast reference object in the image. However, it needs an additional reference optical sensor and a model fitting or "learning phase" to estimate the mapping function between the computed scene's contrast and the visual range. Two main techniques for computing this mapping function are nonlinear regression and classification. The desired output type is the main difference between these approaches. Classification is used for problems involving discrete output values (e.g., estimating visibility range), while regression

is associated with problems involving continuous output values (e.g., estimating visibility distance).

  - **Nonlinear Regression**: The weighted intensity of the power spectrum (WIPS) as an indicator of the image's contrast and found it to be well-correlated with Subjective Visibility Assessment Values (SVAVs) given by test subjects.

  - **Classification**:A Scale Invariant Feature Transform (SIFT) key points of reference regions of interest (containing stationary objects with distinct edges and an approximate Lambertian surface) was used as targets. Using a combination of spatial features based on local contrast and frequency features based on the Fourier power spectrum, they constructed a feature vector. A Support Vector Regression (SVR) model was dynamically trained on samples with similar features to the estimated image. Their experimental evaluations showed promising results despite challenges in stability and accuracy when dealing with practical applications [4].

  - **Offline map-based method** [4]: This method was proposed for determining daytime visibility by preparing an offline map with numbered visible targets. They applied an edge detection algorithm to gray-scale images, finding better results with the Sobel operator compared to other operators. However, this approach is limited when the granularity of the visibility range is high, reducing its usefulness for highways.

  - **Drawbacks of target-based methods**: These methods have drawbacks such as the need for accurate geometric camera calibration in some approaches, reliance on high-contrast reference objects, and ineffectiveness in dealing with practical situations like occlusion problems, curved roads, and old lanes. Additional drawbacks include the need to develop appropriate target detection algorithms and the requirement for specific visibility markers or targets, which can be removed, displaced, or visually blocked in real-life settings. The cost of installing and maintaining these visibility markers should also be considered.

- **Classification Approaches**: Pavlic utilized the Fourier Transform as image features to distinguish between clear and foggy weather conditions based on a power spectrum approach. They performed a two-stage feature reduction consisting of (1) sampling the spectrum in the frequency domain using a filter bank of scaled and oriented Gabor filters, and (2) conducting feature selection and classification on the

Gabor feature vector using a linear classifier based on Fisher's Linear Discriminant Analysis (LDA). In [17], Hallowell proposed an algorithm applying edge extraction to the camera image using the Sobel algorithm, combining the raw image and edge detections to yield an overall composite. Image and edge global descriptors are calculated, each correlated to the visibility distance estimated by an additional sensor. A fuzzy logic scoring system was applied to integrate the estimates and derive four classes of visibility ranges. In [7], Chaabani uses an Artificial Neural Network (ANN) classifier to estimate the visibility range, while Chaabani proposes a deep learning method for feature extraction and an SVM classifier, providing better performance results. A key limitation of classification-based approaches is the scarcity of public databases with varying fog densities, crucial for training and validating the proposed visibility measurement methods.

  - Pavlic : Used the Fourier Transform to distinguish between clear and foggy weather conditions. Performed a two-stage feature reduction with Gabor filters and linear classifier based on Fisher's LDA.
  - Hallowell [17]: Proposed an algorithm combining edge extraction and global descriptors correlated to visibility distance, using a fuzzy logic scoring system to derive visibility ranges.
  - Chaabani [7]: Used an ANN classifier and a deep learning method for feature extraction and an SVM classifier, improving performance results. A key limitation is the scarcity of public databases with varying fog densities.

- **CNN Approaches**: numerous CNN approaches have been proposed in the matter of visibility detection
  - Lun Lo [28] proposed a new method for image classification where they used a multi-support vector regression in classifying the pictures after the feature extraction of the image done by the CNN where a pretrained VGG16 Neural Network was used.

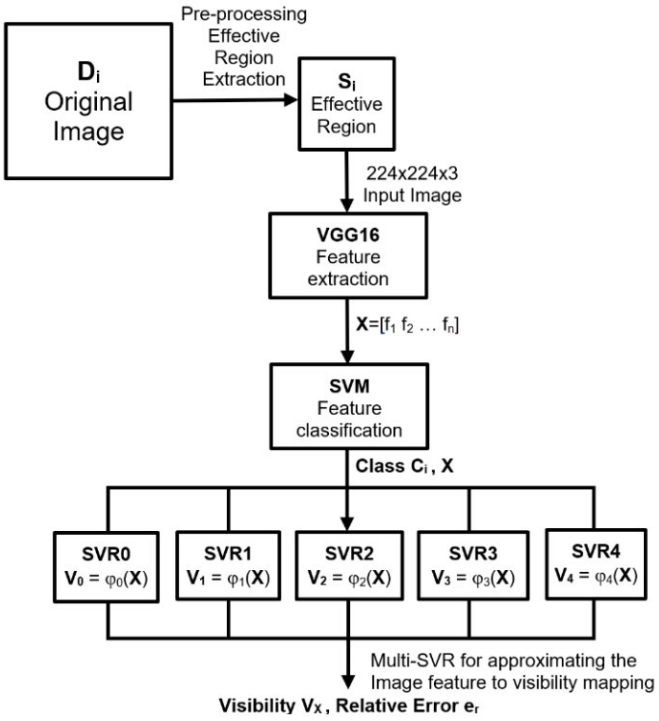


Figure 2.11: Proposed Multi-SVR System

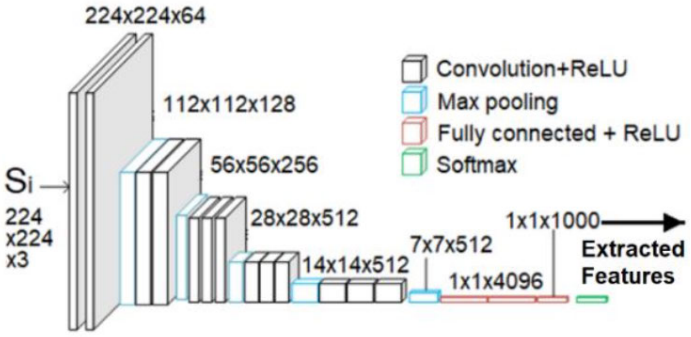


Figure 2.12: CNN: VGG16 Network Architecture

As shown in figure 2.11, the original images undergoes several stages where in the system depicted in figure 4.2, effective image regions are extracted based on landmark distances and fed into a VGG16 network to generate features. These features are classified by a Support Vector Machine (SVM) into distinct

visibility ranges. Then, Support Vector Regression (SVR) models estimate the actual visibility using the feature vectors.

The results of this algorithm is the following. The performance of the proposed algorithms was evaluated through simulation studies. The overall average relative percentage error e_r was 12.15%, and about 85% of the predicted values matched the expected values. The proposed algorithm demonstrated better performance compared to previous research, which had a prediction accuracy of 62%. The improved performance is attributed to the use of multi-SVR models, approximating the complex non-linear function by combining several local linear or simple non-linear function segments for different regions. This piece-wise strategy led to better results than a simple linear regression for this non-linear problem with multiple input variables.

2.4.2 Conclusion

The previously introduced researches covered some of the extremely essential points in visibility detection under foggy weather conditions. Nevertheless, there are open research directions which should be explored and gaps that prevent the advancement of the technology.

- Research Directions:

    1. The potential of exploring data driven approaches without relying on meteorological devices and learning phases needs to be investigated.

    2. Compared to classical artificial neural networks approaches, a deep learning approach might offer an interesting alternative to estimate visibility distance under both daytime as well as nighttime foggy weather conditions.

    3. The computer vision and image processing techniques that have been proposed to estimate visibility distance under foggy weather conditions open new opportunities to devise methods to restore images in the presence of fog, which can provide valuable assistance to drivers.

    4. Additional artificial intelligence algorithms should be explored to enhance the mining of digital imagery for visibility distance estimation in the presence of fog.

5. While earlier studies have mostly relied on the usage of high-resolution digital cameras, few studies have looked at leveraging existing classical CCTV cameras which are widely deployed on major highways for road surveillance and security purposes.

6. The usage of data fusion based on hybrid measurements might lead to more robust solutions especially when the errors in the estimates are at least partially independent.

- Gaps: There is a need to develop public databases of camera images with varying degrees of fog intensities and visibility levels. The scarcity of such databases makes it difficult to develop robust machine-learning based methods or to compare various visibility estimation methods.

3 Requirements

In this chapter, the requirements, which will be utilized to set the goal, are going to be defined based on the MoSCoW prioritization technique [11] where only two criterion will be used which are: Must-haves (where the model is not considered complete without any of these requirements) and Could-haves (where the model can still be considered complete even if none of them are fulfilled). The requirements are to be described throughout this chapter and fully listed by the end of it.

3.1 Input Data

The input data are to be used in the form of images of a resolution no less than 640 x 480, collected during daytime and classified into three categories which are the following:

- **Clear**: This is the category which, if analyzed, states that the weather in the picture is clear i.e. either sunny or cloudy. It states that the visibility conditions are extremely good which means there would be no visibility distortions at all ($>$10m).
- **Almost Clear**: This is the category which, if analyzed, states that the weather in the picture is almost clear i.e. there is a bit of fog in the weather but the visibility conditions are average where a motorist would be able to drive through the country but slowly and carefully (range between 1.5m and 10m).
- **Foggy**: This is the category which, if analyzed, states that the weather in the picture is foggy or heavily foggy i.e. the fog is covering the atmosphere completely which makes the visibility conditions bizarre making motorists unable to drive through such weather. Therefore it is always advised by weather forecasters in such circumstances to either drive slowly or not at all ($<$1.5m).

3.2 Output Data

Should the case require the model being fed a data set after being fully trained and validated, the output is expected to be one of the categories for each image in the data set: **Clear**, **Almost Clear** and **Foggy**.

3.3 Training Quality

The quality of the training is decided by the ***Accuracy*** of the training. It is described in [15] that the accuracy is the proportion of examples for which the model produces the correct expected output. It is also mentioned that a good indication on the performance measure can also be the error rate. In the case of this analysis, the error rate is expressed by the loss in each step of the training phase in a single epoch.

In this model, the accuracy is going to be taken into consideration as the official performance measure. The accuracy of the model is expected to be at least 90% for it to be considered a viable model for the application of visibility conditions detection.

3.4 Method of Data Collection

As mentioned in 3.1, the input data is a data set of images which have a resolution no less than 640 x 480. The data set is collected from a data set that was available on IEEE Dataport[1].

The data set is originally made up of 5 classes which express different categories of meteorological activities. These five classes are cloudy, sunny, foggy, rainy and snowy. Two of these classes are already well-classified and well-labelled. The other three classes' images are going to be used to increase further the images of the fore mentioned two classes and in the creation of the *Almost Clear* class.

[1]IEEE Dataport is a website where multiple data sets exist in various fields. The data set used can be accessed through the following link

3.5 System Requirements

It's required that the model is fully implemented on a linux-based environment with python being the programming language used for the development of the model. This means that the model will be available for operation on any linux-based environment. The architectures, libraries and packages used to do the image data augmentation, use pre-trained CNN architectures and assess the outcome should be publicly available through TensorFlow and Keras.

3.6 Requirements Table

The requirements have two criterion to be judged on:

1. Must Have: This means that it's essential to have this requirement to consider the model complete.
2. Could Have: This means that if the model doesn't contain this feature, it wouldn't decrease the quality of the end result.

The following are all the requirements listed for the model's acceptance.

ID	Must/Could Have	Requirement
1	Must	The model must have at least at accuracy 90% when training the model on the training set
2	Could	The model must have at least 90% accuracy when testing the model on the validation data set
3	Must	The output of the model must be one of the three classes: **Clear**(defining that the weather is clear), **Almost Clear** (defining that the weather contains some fog but the visibility is only lightly distorted), **Foggy** (defining that the weather contains fog or heavy fog and the visibility is heavily distorted)
4	Must	The images of the data set must be only day-time images
5	Must	The pre-trained CNN architectures and networks involved in the development must be publicly available on Keras
6	Must	On introducing a batch of images to the model, the output to be expected is categorized according to the 3 classes mentioned in requirement 3
7	Must	The used software for the development of the model must be open-source software
8	Could	The images used in the training and validation phases must have a resolution no less than 640 x 480 pixels (VGA)
9	Must	The development of the model must be done using python as a programming language and juypter notebooks for code formulation
10	Could	The time spent to train the model per epoch (iteration) shouldn't exceed 700 seconds

Table 3.1: Requirements Table with each requirement having an ID number, whether the requirement is a must or could be had by the end of the development and implementation phase and the description of each requirement

4 Conception

In this chapter, the achievement methodology of the proposed solution will be clarified through the pre-trained CNN architectures, four of which will be chosen which are publicly available on the keras [2]. The first step of the achievement methodology clarification is that the architectures will be discussed in the manner of their parameters, depth and working methodology.

The second step is explaining what additional hidden layers will be added and what they represent.

The third step is determining the suitable algorithm workflow to achieve the required functionality described in 3.1.

4.1 Chosen Architectures

A number of pre-trained deep learning architectures are made available publicly which fulfills requirement 7 in table 3.1. In the following table, The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset [2].

Depth refers to the topological depth of the network. This includes activation layers, batch normalization layers etc [2].

Model	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time(ms) per Inf. Step (CPU)	Time(ms) per Inf. Step (GPU)
Xception	0.790	0.945	22,910,480	126	109.4	8.1
VGG16	0.713	0.901	138,357,544	23	69.5	4.2
VGG19	0.713	0.900	143,667,240	26	84.8	4.4
ResNet50	0.749	0.921	25,636,712	107	58.2	4.6
ResNet101	0.764	0.928	44,707,176	209	89.6	5.2
ResNet152	0.766	0.931	60,419,944	311	127.4	6.5
ResNet50V2	0.760	0.930	25,613,800	103	45.6	4.4
ResNet101V2	0.772	0.938	44,675,560	205	72.7	5.4
ResNet152V2	0.780	0.942	60,380,648	307	107.5	6.6
InceptionV3	0.779	0.937	23,851,784	159	42.2	6.9
InceptionResNetV2	0.803	0.953	55,873,736	572	130.2	10.0
MobileNet	0.704	0.895	4,253,864	88	22.6	3.4
MobileNetV2	0.713	0.901	3,538,984	88	25.9	3.8
DenseNet121	0.750	0.923	8,062,504	121	77.1	5.4
DenseNet169	0.762	0.932	14,307,880	169	96.4	6.3
DenseNet201	0.773	0.936	20,242,984	201	127.2	6.7
NASNetMobile	0.744	0.919	5,326,716	389	27.0	6.7
NASNetLarge	0.825	0.960	88,949,818	533	344.5	20.0

Table 4.1: Comparison of Keras image classification models [2]

The architectures marked in green are the ones which are chosen as proposition for a solution for a number of reasons and that is:

1. They are publicly available on keras [2].

2. They provide top-1 and top-5 accuracy measurements which are reasonable to the requirements, see table 3.1 requirement 1.

3. All of these architectures have limited number of parameters available for training as to decrease the time spent in training per epoch, see table 3.1 requirement 10

There are several reasons other high-scoring in top-1 accuracy and top-5 accuracy architectures, such as: InceptionResNetV2 and NASNetLarge, are not considered:

1. **Computational Complexity:** InceptionResNetV2 and NASNetLarge are both very deep and complex models. They require a large amount of computational resources, both in terms of memory and processing power. This can be a limiting factor for their use in real-time applications or on devices with limited resources.

2. **Training Time:** Given their size and complexity, these models take a long time to train. In situations where rapid prototyping or frequent retraining is necessary, simpler models may be preferable.

3. **Dataset Requirements:** These models typically perform best when trained on large, diverse datasets. They have a high chance of overfitting or performing poorly on smaller, less varied datasets.

4. **Model Interpretability:** The complexity of these models can also make them more difficult to interpret compared to simpler models. This can be a disadvantage in applications where understanding the model's decision-making process is important.

4.2 Proposed Solution CNN Architectures Overview

In this section, the architectures marked in the table 4.1 will be explained in correlation with:

- Number of parameters: This is one of the most important aspects of a CNN architecture where it relates directly to the number of parameters that will be used in the training. Those parameters are classified into two types:
  - Trainable parameters: parameters which will be trained according to the dataset provided to them.

  - Non-trainable parameters: parameters which are already trained on the weights provided in the CNN architecture setup (initialization), i.e. they don't have to undergo a training phase unlike the trainable ones. It's worth mentioning, the training for these parameters can be unfrozen (allowed), which allows the training of the network's parameters for a second time without removing the weights.

- Depth of the architecture: Depth of a CNN architecture refers to how many hidden layers are between the input and the output layers of the network.
- Layers' Specification: This explains how the architecture is actually designed, how the layers are structured and how reasonable the layering in one block of the architecture is.

4.2.1 DenseNet201

DenseNet201 is a variant of DenseNet, a type of convolutional neural network, where 201 indicates the total number of layers within the network.

According to [23], a DenseNet block has three main components:

1. Batch Normalization (BatchNorm)
2. Rectified Linear Units (ReLU)
3. 3x3 Convolution (Conv)

In the DenseNet201, the layers are structured as follows: DenseNet201 specifically has four dense blocks, and each dense block is followed by a transition layer. The numbers of layers in the dense blocks are 6, 12, 48, and 32 respectively, which totals to 98 layers. Adding the 4 transition layers, 1 initial convolution layer, and 1 final dense layer with softmax activation gives a total of 104 layers. However, considering each layer as consisting of BatchNorm-ReLU-Conv trio, we count as 3 layers each. This means the effective "depth" is 201 layers. The exact number of the parameters can not be determined but according to [23], they are about 20 Million.

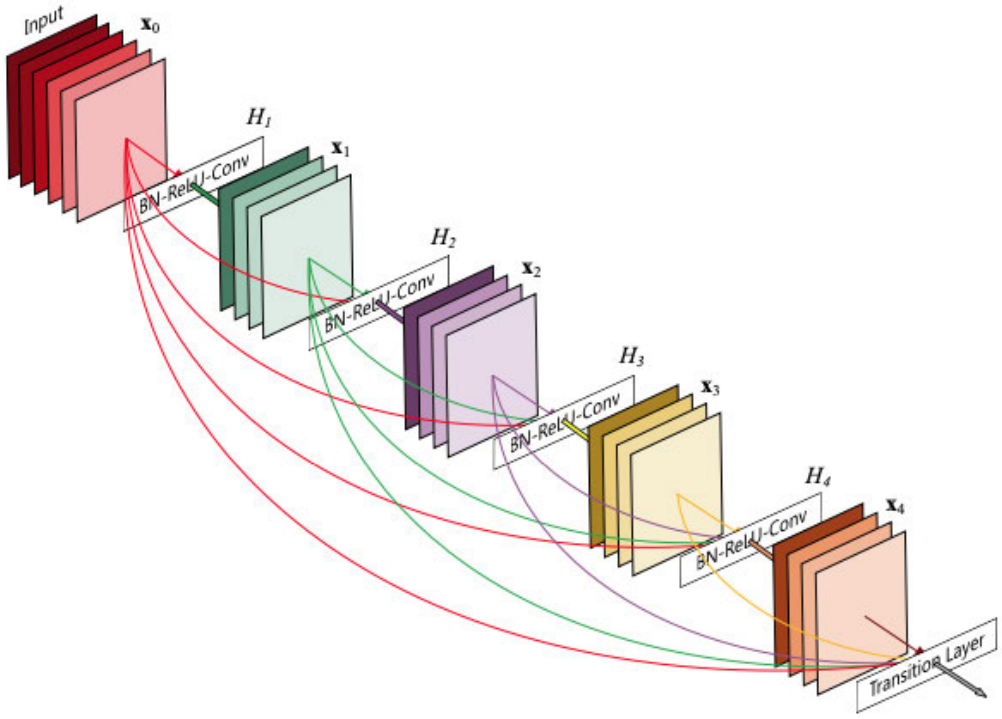


Figure 4.1: DenseNet Architecture flow [23]

4.2.2 VGG16

VGG16, as previously mentioned in the state of the art section 2.4.1, is a deep learning architecture well known for its usage in several applications where transfer learning and feature extraction is required [10].

- Parameters: The architecture contains 138.4 Million parameters, 40 Million of which are available for the training phase, 26,217,475 parameters of which are available to be trained on the new dataset whereas the rest is already weighted (pre-trained) where the network is weighted with weights of the **ImageNet** network
- Depth: The architecture is of a 16-layer depth.
- Layers' specification: The architecture includes the following layers [34]:

1. Input Layer: The input to the VGG16 network is a 224x224 RGB image, which passes through the network's layers in order.

2. Convolutional Layers (conv3-64, conv3-128, conv3-256, conv3-512): There are 13 convolutional layers in VGG16. They are grouped into five blocks by the size of the feature maps they produce. Each convolutional layer uses a filter size of 3x3, stride of 1 (move the filters one pixel(step) at a time) and same padding (adding extra pixels to the input images to prevent spatial reduction). Conv3 denotes the architecture's convolutional layer using 3x3 filters and the number following "Conv3-" denotes the number of filters being used in the layer. The depth of the feature map increases from 64 in the first two layers to 128, 256, and finally 512 in the subsequent layers [34].

3. Max-Pooling Layers: After each group of convolutional layers, there's a max-pooling layer, which reduces the dimensions of the feature maps 1 by half (e.g., from 224x224 to 112x112), effectively reducing the computation required by the network and helping to generalize the features learned.

4. Fully Connected Layers (FC-4096): After the last max pooling layer, the feature maps are flattened into a single dimension vector to be fed into fully connected layers. The VGG16 architecture includes three fully connected layers. The first two have 4096 nodes each, and they function to combine features to classify images [10].

5. Softmax Layer (FC-1000): The final layer is a fully connected layer with 1000 nodes (one for each class in the ImageNet database). This layer uses the softmax activation function to provide the output probabilities for each class [34].

6. ReLU and Dropout Layers: Each convolutional and fully connected layer is followed by a Rectified Linear Unit (ReLU) activation function, which introduces non-linearity into the network. The first two fully connected layers are also followed by dropout layers for regularization to prevent overfitting [34].

The way these layers are structured is that the preprocessing for the image only involves subtracting the mean RGB value, calculated based on the training data set, from every pixel. Then, the image is processed by several convolutional layers that use filters with a very small receptive field: 3 x 3, which is the smallest size capable of capturing notions like left/right, up/down, and center [34].

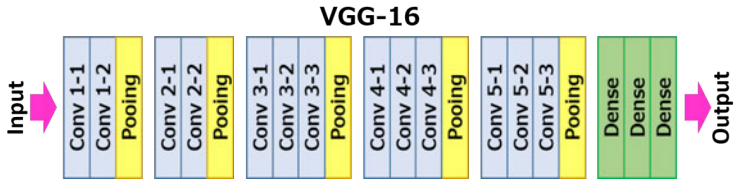


Figure 4.2: VGG16 architecture flow

4.2.3 ResNet152

The ResNet152 architecture is a deep residual network where the network is derived from the basic 34-layer ResNet network but with a key difference which is the bottleneck layers that are used in the construction of the model [19].

For the base 34-layer ResNet model, each residual block consists of two convolutional layers with 3x3 filters. However, for deeper ResNet models like the ResNet-152, each residual block is replaced with a "bottleneck" block.

A bottleneck block consists of three convolutional layers instead of two which are [19]:

1. A 1x1 convolution that reduces the dimension (number of channels): This layer performs a feature selection process by reducing the depth (number of channels) of the input. This is achieved by creating a weighted combination of the input channels. The number of 1x1 filters used equals the number of output channels desired. The input channels dimensionality is reduced to decrease the computational complexity of the next layer, which uses 3x3 filters.

2. A 3x3 convolution that processes the reduced-dimension input: This second layer performs the main amount of processing. Because the previous layer reduced the number of input channels, this layer can apply a relatively expensive 3x3 convolution while keeping computational costs under control. This layer's role is to process and transform the features selected by the previous layer.

3. A 1x1 convolution that restores the dimension to its original size: The job of the third layer is to restore the dimensionality back to the original depth, ready to be passed onto the next residual block. Similar to the first layer, it creates a weighted combination of the input channels, but this time it's increasing, not decreasing, the number of channels.

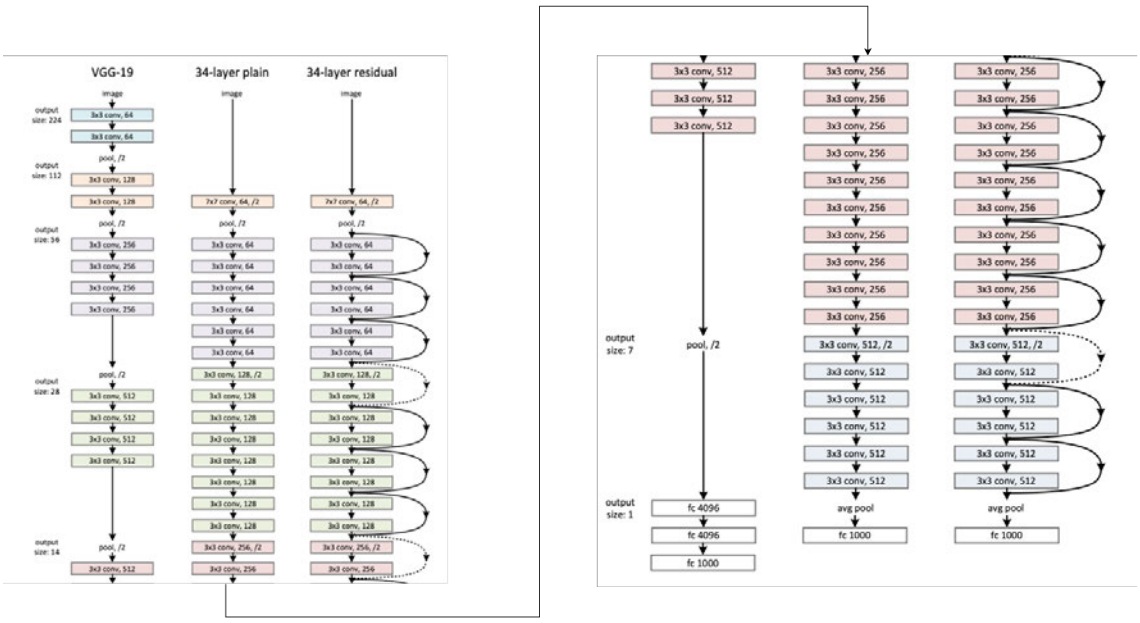


Figure 4.3: ResNet152 Architecture flow diagram [19]

4.2.4 Xception

The Xception architecture proposed by Chollet in [9], has a number of applications in the convolutional neural networks area due to the sophistication its architecture provides

The Xception architecture is specified as follows:

- Parameters: The architecture has a total of 22,910,480 parameters available for training.
- Depth: The architecture has 126 layers in total [9].
- Layers' Description: The architecture is divided into three main parts [9]:
  - **Entry Flow**: The input to the network is processed through two convolutional layers with 32 and 64 filters respectively, followed by a series of residual connections with depthwise separable convolutions.
  - **Middle Flow**: The middle section is composed of 8 identical modules. Each module is a residual connection with depthwise separable convolutions. These modules do not alter the dimensions of the feature maps.

– **Exit Flow**: The final section consists of a residual connection followed by a separable convolution layer, then a max pooling operation, another separable convolution layer, and finally a global average pooling operation leading to the final output layer.

Each of the separable convolutional layers is followed by batch normalization and then a rectified linear unit (ReLU) activation [9].

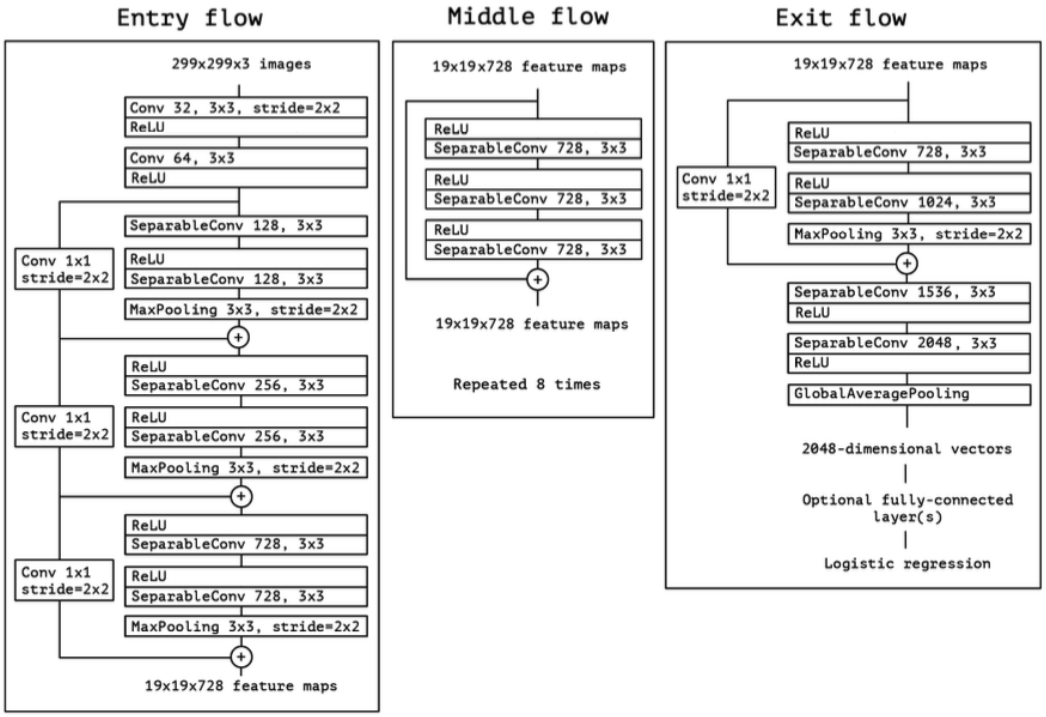


Figure 4.4: Xception Architecture flow [9]

4.2.5 Classification Layers

There are 4 prediction head layers used in the proposed solution to classify the output of the convolutional layers:

1. Flatten: This is a layer used to change the multi-dimensional vector from the CNN architecture to a one-dimensional vector

2. Dense(fully connected): This is the layer used to correspond the output the the labeled datasets, three of these layers are used with the activation function ReLU[1] and the final one is used with the softmax function in the output layer of the model. This activation function is used specifically for this model in order to solve the multi-class classification issue where the activation function turns the vector of real numbers produced by the model and turns them into a probability distribution

4.2.6 Optimizer

The optimizer's purpose is to adjust the weights of the network with the aim of reducing the loss function [24].

There are different available optimizer algorithms, some of which are:

1. **Adam** [24], which stands for Adaptive Moment Estimation, is an optimization algorithm that has been used extensively in training deep learning models, including convolutional neural networks. The Adam optimizer has several advantages which make it suitable for this purpose:

    a) Efficient computation: Adam requires minimal memory and is computationally efficient, which is particularly useful when training large models or dealing with large datasets.

    b) Adaptive learning rate: Adam automatically adjusts the learning rate for each parameter in the model based on estimates of the first and second moments of the gradients. This can be beneficial when dealing with complex datasets where some features may require faster or slower learning rates.

    c) Suitability for non-stationary objectives: Adam is effective even when the objective function isn't stationary, making it suitable for scenarios such as online and machine learning settings.

    d) Robustness to noisy gradient estimates: This makes Adam particularly effective in scenarios with large amounts of data and parameters.

[1] ReLU: short for Rectified Linear Unit

2. **SGD** [6], which stands for Stochastic Gradient Descent, is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression.

   The principle behind SGD is straightforward [6]. Given a set of training examples, traditional gradient descent would calculate the gradient of the loss function for each example in the training set, sum these gradients, and then update the model's parameters. While this approach is fine for small datasets, the computational cost becomes prohibitive for larger datasets.

3. **RMSprop**, which stands for Root Mean Square Propagation, is an optimization algorithm developed by Geoff Hinton, introduced by him in his course [21]. The algorithm builds upon Rprop (resilient backpropagation) and scales the learning rate by dividing it by an exponentially decaying average of squared gradients. The key idea behind RMSprop is to resolve the problem of diminishing learning rates in the Adagrad optimization algorithm. RMSprop accomplishes this by using a moving average of squared gradients. It utilizes the magnitude of the recent gradient descents to normalize the gradient.

4. **Adagrad** [13] is a first-order gradient-based optimization algorithm designed to handle sparse data. The key idea behind Adagrad is to adaptively scale the learning rate for each parameter in the model, which is particularly beneficial when dealing with sparse features. With Adagrad, features that are rare but informative get a higher learning rate, and common but less informative features get a lower learning rate.

   One limitation of Adagrad is its accumulation of the squared gradients in the denominator: since every added term is positive, the accumulated sum keeps growing during training.

The previously mentioned optimizers are all efficient for the functionality and can be used. However, according to previous research and literature [10], Adam is the one widely used for image processing applications. Therefore, Adam is the optimizer to be used.

4.3 Chosen Optimizer and Architectures

The optimizer and architectures chosen for the development and implementation is the following:

- Architectures:
    1. VGG16
    2. DenseNet201
    3. Xception
    4. ResNet152
- Optimizer: Adam

The following chapter will show how these architectures are used in the implementation and further developed.

5 Development and Implementation

In this section, the approach of development is going to be clarified with the detailed steps of each phase.

1. An overview of the development steps will be disclosed.
2. The environment preparation will take place where all the imported libraries will be explained in a manner of the version that was imported and the use of the library in the code.
3. The data collection phase will be explained where the source of the data will be clarified with how the data is structured and how it's changed in order to achieve the required functionality, see table 3.1, requirement 3.
4. The data augmentation process will be described to show what exactly in the image is being augmented (changed) and how an example image would look like after this augmentation.
5. The training phase will be demonstrated with chosen architectures and optimizer from the previous section 4.1 4.2.6.
6. In order to totally narrow down the architecture choice to one single solution architecture, Keras tuner will be used in order to detect which hyper-parameters are the ones to be used to improve the training accuracy and retrain them to have the higher accuracy required, see table 3.1 requirement 1 and 2.

5.1 Overview of The Development Steps

Training the model requires different stages of dataset preparation and model development. In the figure 5.1, the main steps of the development are mentioned which are:

- Data Preprocessing: This is the step where requirements number 5 and 6 in table 3.1 are fulfilled where the images will be augmented and altered based on the needs of the architectures.
    1. Data Retrieval and Data Preparation and Transformation.
    2. Data Split: This is the step where the dataset is to be divided into three parts, the first part is the training dataset, the second one is the validation dataset and the third one is the testing dataset.
- Training:
    1. Model Initialization: This is the step where the model made publicly available using keras [2] is initialized with specific properties such as: weights, top level input layer inclusion and trainability of the parameters.
    2. Output Layers Initialization: This is the step where output layers are initialized based on the classes defined on the labeled dataset provided as input.
    3. Training the model with proper batch size.
- Validation and Testing: These are the final steps to conclude development of the trained model.
    1. Overfitting Detection: This is the step where the output graphs of model training and validation phases' accuracy will be compared to reach a consensus whether the model is overly fit.
    2. Prediction on an arbitrary dataset: This is the step where the final testing phase will be done through which an arbitrary dataset will be used in order to predict the classes of the images

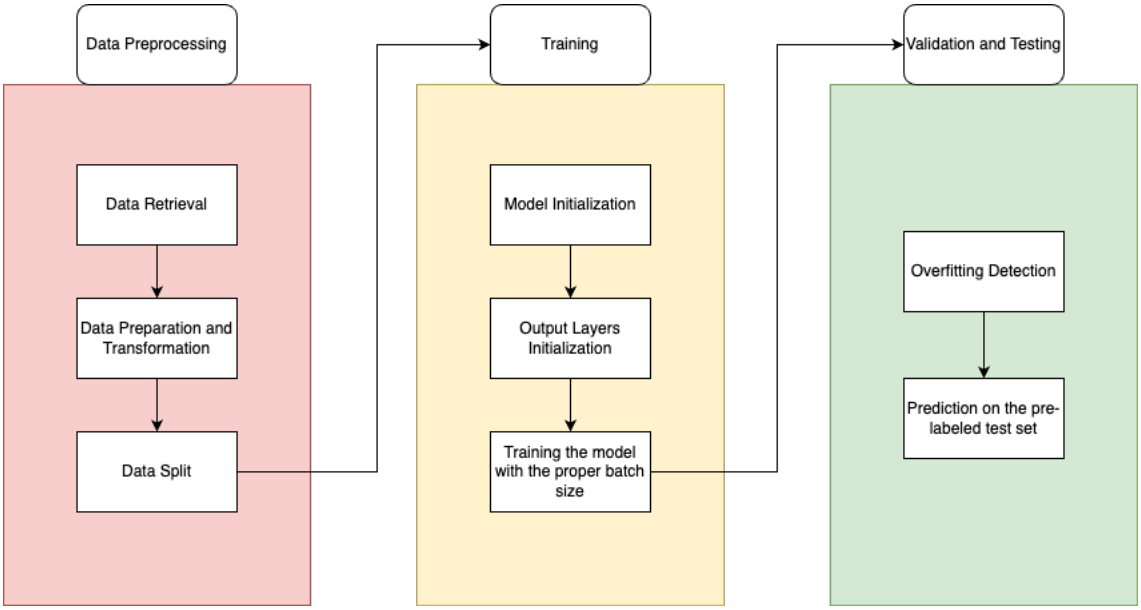


Figure 5.1: Development Approach

5.2 Environment Preparation

In order to prepare the development environment, the following libraries [9] [10] [15] are imported.

Library	Latest Version	Use
Keras	2.6.0	High-level neural networks API, running on top of TensorFlow
os	Standard library in Python	Provides functions for interacting with the operating system
TensorFlow	2.6.0	Open-source platform for machine learning
ImageDataGenerator (from Keras)	2.6.0	Generates batches of tensor image data with real-time data augmentation
image (from Keras)	2.6.0	Provides tools for image data preprocessing
NumPy	1.21.2	Fundamental package for scientific computing with Python
Matplotlib	3.4.3	Python plotting library which produces publication quality figures
PIL (Pillow)	8.3.2	Adds image processing capabilities to Python interpreter

Table 5.1: Python Libraries Imported and their Uses

```
1 import keras, os
2 from keras.models import Sequential
3 from keras.layers import Conv2D, MaxPool2D, Flatten, Dense
4 from keras.preprocessing.image import ImageDataGenerator
5 from keras.preprocessing import image
6 import numpy as np
7 import matplotlib.pyplot as plt
8 import tensorflow as tf
9 from PIL import ImageFile
```

5.3 Data Collection

The data collection might be the most challenging part in this thesis due to the fact that there is almost no free images providing any information on weather fogginess or clearness. The main idea is to have three sets of pre-labeled images, each of which contains 1000 images for the training phase. For the validation data set, a set of 200 images will be created under each pre-labeled class. For the testing of the compiled model, a set of 200 images will be created under each pre-labeled class.

5.3.1 Dataset

The Five Class Weather Image Dataset, found on IEEE Dataport [14], is an extensive collection of images that are categorized into five distinct weather conditions: cloudy, sunny, foggy, rainy and snowy. The dataset provides a valuable resource for training and testing machine learning and deep learning models, particularly in the area of weather classification and prediction.

Each image in the dataset is associated with a specific weather condition, offering an opportunity to develop and validate algorithms capable of automated weather recognition based on visual cues. The dataset is designed to be challenging due to the variety of images and the complexity of distinguishing between some weather types.

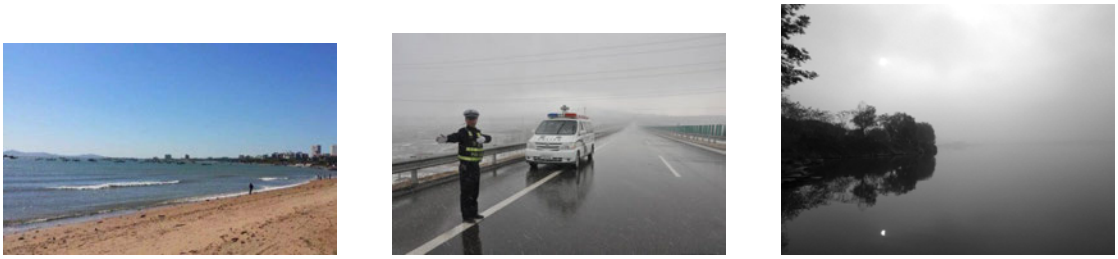
The data is stored in the JPEG format, which is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography. This format makes it easy to work with the dataset using standard image processing tools and libraries.

The images are collected from different sources ensuring a good variability in terms of the scenery, location, time of day, and season, making the dataset versatile for different use-cases. It can be used for various applications including weather forecasting, climate study, training of autonomous vehicles, augmented reality, and other fields where weather recognition can play a crucial role.

5.3.2 Classification Description

The dataset provides a 2/3 of the classification of the dataset required to acquire the full functionality. What is provided is the **Clear** dataset through the sunny and the cloudy classes, and the **Foggy** dataset through the foggy class. What is left is deriving the third dataset, which is the **Almost Clear** images, from the two classes snowy, cloudy and foggy. The reason for doing this is those classes provide an exact degree of weather and fogginess where one can derive a state in which one is able to drive a car while able to see the road clearly but not anything farther than that, i.e. only the oncoming and outgoing cars on the same road would be visible yet any buildings, sky scrapers or normally far sighted structures to the point where one is wouldn't be visible as compared to normal weather conditions when they normally would.

The following is an example of the proposed images for the datasets where each is a representation of how the rest of the images in each of them would look like.



(a) Clear (b) Almost Clear (c) Foggy

Figure 5.2: Examples of the classes in the dataset [14]

In figure 5.2, 5.2a shows an example of the **Clear** dataset where all the features of a place are visible without any visual enhancement. 5.2b shows an example of the **Almost Clear** dataset where the features of the scene are 70% visible and the fog isn't thick where a distance between different cars is easily perceivable for the drivers. 5.2c shows an example of the foggy dataset where the fog is thick in which an outgoing car is barely visible to the human eye.

The following is the exact division of the dataset:

- Training:
    1. Clear: 1000 images

2. Almost Clear: 1000 images
3. Foggy: 1000 images

- Validation:
  1. Clear: 200 images
  2. Almost Clear: 200 images
  3. Foggy: 200 images
- Testing: 522 images

5.3.3 Image Augmentation

The augmentation [37] of an image is a necessary step to reduce the possibility for overfitting , the following code snippet shows the specific properties which are to be augmented in an image.

Listing 5.1: Image data augmentation using ImageDataGenerator

```
trainingGen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

The editing of the images is done by a function that is provided by TensorFlow [3] called `ImageDataGenerator()`. A lot of arguments are editable in an image, some of which are necessary for the model training which are [1]:

- rescale=1./255: This rescales pixel values in the image. Images are made up of pixels with values between 0 and 255. Neural networks work better with small input values, so the images are rescaled to have values between 0 and 1.
- rotation_range=40: This randomly rotates the image within a range of 40 degrees.

- width_shift_range=0.2: This randomly shifts the image horizontally by a factor of 0.2.
- height_shift_range=0.2: This randomly shifts the image vertically by a factor of 0.2.
- shear_range=0.2: This applies shear transformations to the image, which are a type of distortion where the image is skewed in such a way that lines parallel to the bottom of the image remain parallel, but lines parallel to the sides may become non-intersecting (not parallel).
- zoom_range=0.2: This randomly zooms in and out of the image by 20%.
- horizontal_flip=True: This randomly flips the image horizontally.
- fill_mode='nearest': If the rotation, shifts, or zooming alter the image size, this fills in newly created pixels. The method of fill is the 'nearest' existing pixel value.

The main idea behind the previous augmentation steps is to provide a kind of a normalization form that is applied to all the images which consequently puts the whole dataset in a matter of fairness, i.e. all the images' pixels are scaled under the same number, all images are horizontally flipped, all images are completely filled with the nearest neighboring pixels in case the shifting led to loss of parts of the image, ... etc.

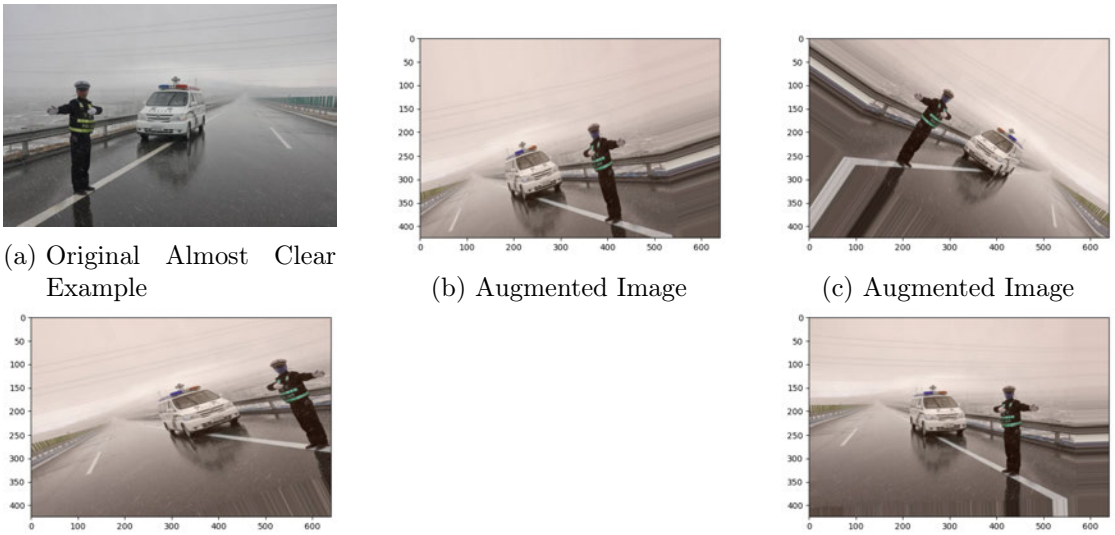
Listing 5.2: Load and preprocess images from a directory

```
1    training_data = trainingGen.flow_from_directory(
2    directory=TRAINING_PATH,
3    batch_size = BATCH_SIZE,
4    target_size = (224, 224),
5    class_mode = 'categorical',
6    shuffle=True)
```

The previous code snippet is executed to import all the code using `flow_from_directory()` function call where it uses the following parameters:

- `directory`: This is the path to the target directory. It should contain one subdirectory per class. Any PNG, JPG, BMP, PPM, or TIF images inside each of the subdirectories directory tree will be included in the generator.

- `batch_size`: Size of the batches of data. If your training set size is not a multiple of batch size, it will impact the number of steps per epoch.
- `target_size`: The dimensions to which all images found will be resized. For the VGG16 model, it typically is set to (224, 224).
- `class_mode`: Determines the type of label arrays that are returned. For multi-class classification problems, it is set to 'categorical'.
- `shuffle`: Whether to shuffle the data. If set to True, the data will be randomly shuffled at each epoch.



(a) Original Almost Clear Example

(b) Augmented Image

(c) Augmented Image

(d) Augmented Image

(e) Augmented Image

Figure 5.3: Example of the output of image augmentation

5.4 Implementation

In this section, the implementation aspects of the model will be comprehensively displayed and tackled. In the conception of the architectures, 4 architectures were proposed as a solution for the issue of detection of visibility conditions under foggy weather. Narrowing down the solution architectures comes down to the investigation of the performance of such architectures. Hence such performance detection can be recognized from previous researches.

5.4.1 Architecture Shortlisting

The selection of the architectures can be shortlisted based on the suitability of the architecture to the application it's used for.

The architecture's complexity is judged based on three main factors [22].

1. Depth: how many layers an architecture has. Deeper networks can model more complex functions and patterns in the data due to their increased hierarchy of representations. However, as the network gets deeper, it may start to fit not just the underlying patterns, but also the noise and outliers in the training data, leading to overfitting[20].

2. Parameters: how many parameters could be trained in the architecture. More parameters mean the model can fit a wider range of functions. But it also increases the risk of overfitting, particularly when the amount of training data is limited [27].

3. Architecture Design: how the architecture is designed in a manner of distribution in the neurons in each layer which corresponds directly to the width of the network (number of filters) and the choice and configuration of layers and the type of connections (feedback, skip-connections in ResNet [19]). These factors can impact the model's capacity and increase the risk of overfitting.

In the architecture selection, three architectures out of the selected four were deep architectures with 50 layers in depth or more. The proposed idea is to shortlist these architectures into one only, as a representation of high number of layers' architectures. With that being mentioned, DenseNet201 offers the highest number of layers (201 layers) in the total chosen architectures. Hence, DenseNet201 and VGG16 are going to be used for the comparison to which architecture solves the problem best.

Later on, using the feature maps, one will be able to detect where in the architectures the overfitting occurs

5.4.2 Implementing CNN Model

The implementation of the CNN model to be trained and analysed will be clarified in this section.

What is left in the code is to start initializing the Keras model to be trained. Using VGG16 as an example, in code snippet 5.4.2, the model is initialized where it's imported using the TensorFlow library to import the pre-trained network directly from the library using the statement `tf.keras.applications.VGG16()` where `tf` represents TensorFlow.

Listing 5.3: Loading the pre-trained VGG16 model and freezing its layers

```
1    VGG = tf.keras.applications.VGG16(
2    input_shape=(224, 224, 3),
3    include_top = False,
4    weights='imagenet')
5
6    VGG.trainable = True #unfreeze training pre-trained parameters
```

The following parameters are used in the initialization of the model.

- `input_shape`: This is used to specify the input shape of the top layer which will be ready to receive the previously augmented images. In case of VGG16, the standard input has to be of size at least `(224, 224, 3)` where:
  1. 224 x 224 represents the height and the width of the image
  2. 3 represents the number of channels in the image because VGG16 is typically trained on RGB images.
- `include_top`: This is a flag which indicates, if false, a new input will be coming in to be trained, if true, the architecture will be used using the same top layer which was provided during training the parameters on the weights.
- `weights`: This represents the weights provided for the model. If left out, then the base model without any pre-training is used. In this case, the `Imagenet` [12] weights are the ones used in this model.

The previous code snippet was merely the initialization of the model, the following one explains the layers add after an image has been processed through the network and how the output is to be presented.

Listing 5.4: Create a new sequential model with the pre-trained VGG16 model and additional layers

```
model = keras.Sequential([
    VGG,
    keras.layers.Flatten(),
    keras.layers.Dense(units = 512, activation = 'relu'),
    keras.layers.Dense(units = 256, activation = 'relu'),
    keras.layers.Dense(units = 3, activation = 'softmax')
])
```

The previous piece of code shows the addition of the created model in 5.4.2 to a sequential model where:

- `model = keras.Sequential([...])`: The model is defined to be a sequential model. Sequential model is a linear stack of layers. This means that we can create a full multi-process model by calling the constructor for a new Sequential object and then just keep calling `.add()` to add more layers.
- `VGG`: This is the pre-trained VGG16 model that was defined earlier. This model is serving as the base model, or the convolutional base, for our new model. It's included as the first layer of the new model.
- `keras.layers.Flatten()`: The Flatten layer [15] is used to convert the final feature maps into a single one-dimensional vector. This flattening step is needed so that we can make use of fully connected (dense) layers after some convolutional/maxpool layers. It combines all the found local features of the previous convolutional layers.
- `keras.layers.Dense(units = 512, activation = 'relu'),`: This line of code is adding a densely connected (also known as fully connected) layer to the model. A dense layer [15] is a layer in a neural network that's fully connected. In other words, all the neurons in one layer are connected to all other neurons in the next layer. In this layer, it has 512 neurons and uses 'relu' as the activation function. 'relu' stands for Rectified Linear Activation [25]. Although it is two linear pieces, it has been proven to work well in neural networks.
- `keras.layers.Dense(units = 256, activation = 'relu')`: This is another dense layer, this time with 256 neurons. Again, 'relu' is used as the activation function.

- `keras.layers.Dense(units = 3, activation = 'softmax')`: This is the final layer, or output layer, of our model. This dense layer has 3 neurons, which should be equal to the number of our target classes. The activation function is 'softmax', which is typically used in the output layer of a multi-class classification problem. Softmax [15] converts a real vector to a vector of categorical probabilities. The softmax function can be mathematically represented by the following equation:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{5.1}$$

  In the above equation:

  - $\sigma(\mathbf{z})_j$ denotes the j^{th} element of the output of the softmax function.
  - $\mathbf{z} = [z_1, z_2, ..., z_K]$ is the input vector to the softmax function, where K is the number of classes or elements in the vector. Each z_i denotes the raw score or 'logit' for each class.
  - The numerator e^{z_j} represents the exponential of the logit for the j^{th} class. This ensures that each output of the softmax function is positive.
  - The denominator $\sum_{k=1}^{K} e^{z_k}$ is the sum of the exponentials of all the logits. This ensures that the sum of the output vector of the softmax function is 1, so it can be interpreted as a probability distribution over the K classes.

The choice of 512 and 256 units for the two dense layers are usually based on trial and error or prior experience. They may also be chosen based on the complexity of the problem or the amount of data one has. Certain tweaks might be needed in order to get better results. Definite answers will come once the hyperparameters are tuned and the best combination of parameters to be trained is to enhance the model's accuracy.

The `compile` function in Keras is used to configure the learning process of the model. In the given c ode snippet:

Listing 5.5: Compiling the model with the Adam optimizer, categorical cross entropy loss, and accuracy metric.

```
model.compile(
    optimizer = 'adam',
    loss = keras.losses.categorical_crossentropy,
    metrics = ['accuracy'])
```

- `optimizer`: This argument is used to specify the optimization algorithm that will be used to update the network weights. The `'adam'` optimizer is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. It is known for its efficiency and low memory requirement.
- `loss`: This argument specifies the loss function that the model will try to minimize. The `categorical_crossentropy` loss function is often used in multi-class classification tasks. It calculates the cross-entropy loss between the true labels and the predicted labels.
- `metrics`: This argument is used to specify the list of metrics to be evaluated by the model during training and testing. The `'accuracy'` metric calculates the proportion of correct predictions over the total number of predictions.

After the compilation of the model, it will start the training process when the `fit` function is called. During training, after each epoch, it will print the computed value of loss and the selected metrics (in this case, accuracy) for the training set. If a validation set is provided in the `fit` function, it will also compute and print these values for the validation set.

The following block of code is where the training of the model actually happens. The `model.fit()` function is used to train the model for a fixed number of epochs (iterations on a dataset).

Listing 5.6: Train the model on the training data with validation on the test data.

```
1 CnnModelHistory = model.fit(x = training_data,
2     validation_data = test_data,
3     epochs = 20,
4     batch_size = BATCH_SIZE,
5     validation_steps = test_data.samples // BATCH_SIZE,
6     steps_per_epoch = training_data.samples // BATCH_SIZE,
7     )
```

Here's an explanation of its parameters:

- `x`: This is the input data, which in this case is a generator yielding tuples of (input, target) for training data.

- `validation_data`: This is the data on which to evaluate the loss and any model metrics at the end of each epoch. Here it's another generator for the test data.
- `epochs`: Number of epochs to train the model. An epoch is an iteration over the entire 'x' and 'y' data provided. In this case, the model will be trained for 20 epochs.
- `batch_size`: Number of samples per gradient update. It's not explicitly used in this method when 'x' is a generator, but it is used to define the generators (training and validation data).
- `steps_per_epoch`: Total number of steps (batches of samples) before declaring one epoch finished and starting the next epoch. When training with input tensors such as TensorFlow data tensors, the default `None` is equal to the number of samples in your dataset divided by the batch size, or 'training_data.samples // BATCH_SIZE'.
- `validation_steps`: Only relevant if `validation_data` is provided and is a generator. This defines the total number of steps (batches of samples) to yield from 'validation_data' generator before stopping at the end of every epoch. It's set to 'test_data.samples // BATCH_SIZE' in this case.

The `fit()` function returns a 'History' object, in this case `CnnModelHistory`. Its `History.history` attribute is a record of training loss values and metrics values at successive epochs, as well as validation loss values and validation metrics values (if applicable).

5.4.3 Feature Maps

Discovering what features are extracted from the images is beneficial to understand in what stage of the convolution the most data is extracted.

Figure 5.4 is an example of foggy images to be considered for feature extraction.

Figure 5.4: Example for feature maps usage

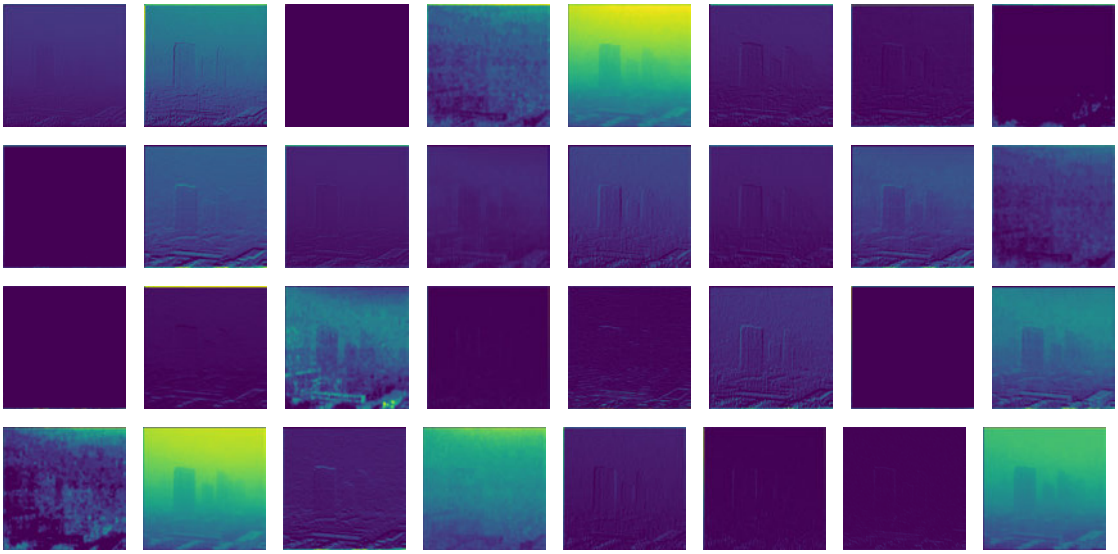


Figure 5.5: Visualization of feature maps block 1 VGG16

Figure 5.5 shows the output of the convolution from the first block of filters (feature maps). The first block extracts a lot of features from the images specifically:

- The extraction of the sky features during fog
- The extraction of the buildings

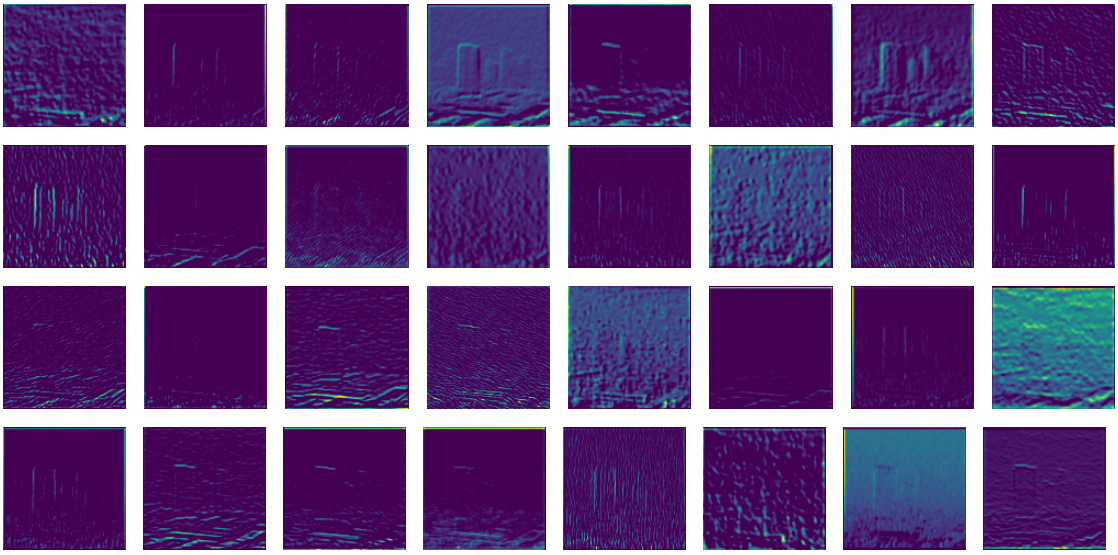


Figure 5.6: Visualization of feature maps block 2 VGG16

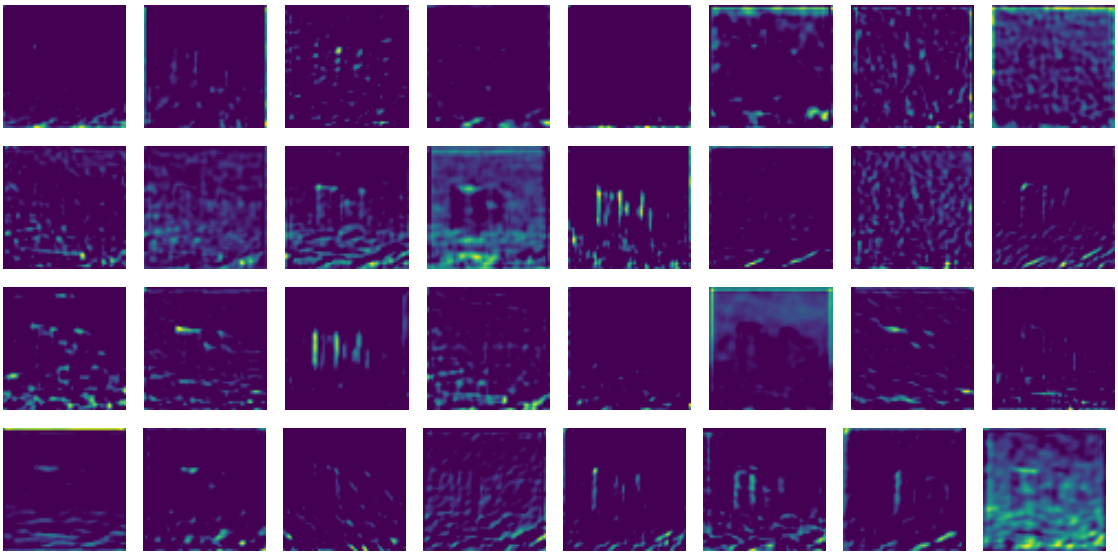


Figure 5.7: Visualization of feature maps block 3 VGG16

Figures 5.6 and 5.7 extract more of the distinct features of the images such as the edges of the buildings.

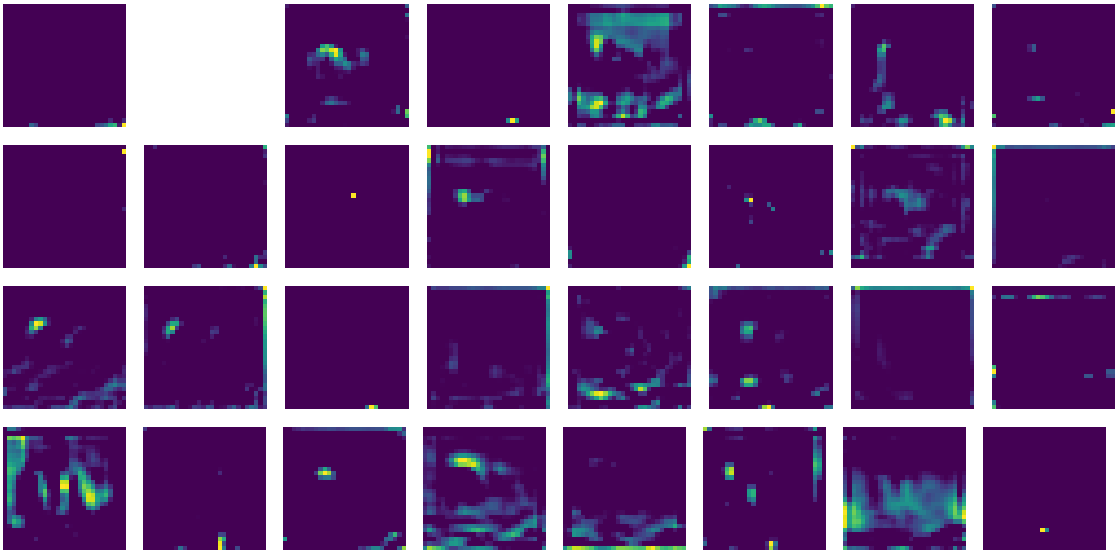


Figure 5.8: Visualization of feature maps block 4 VGG16

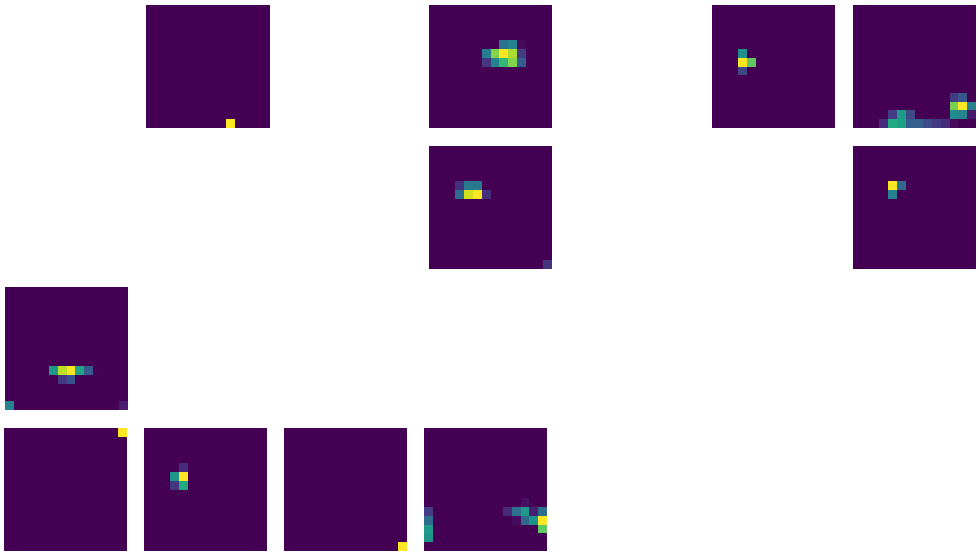


Figure 5.9: Visualization of feature maps block 5 VGG16

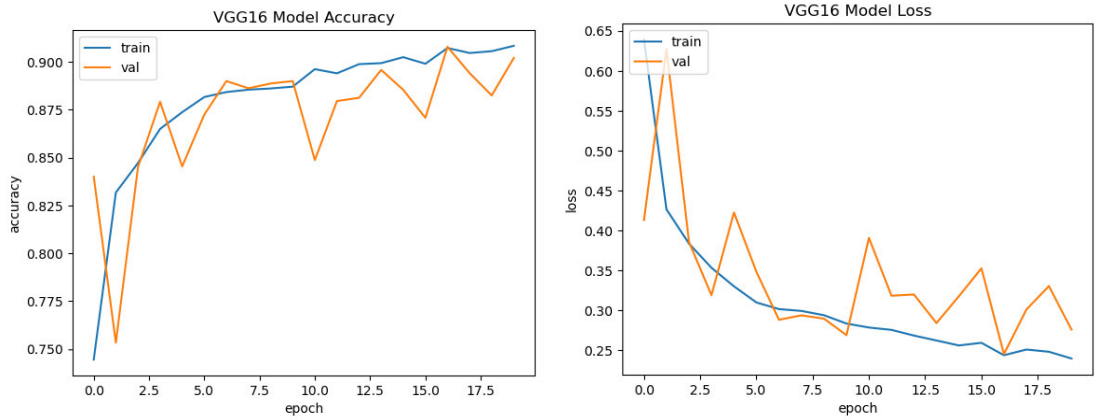
Figures 5.8 and 5.9 show an extremely important feature in which some of the filters don't detect any features, hence producing empty (white) images. The reason for this is that at such point, the filters have extracted all the features and there is nothing left to learn. This is always a good point for the images to stop descending down the

architecture layers, hence the deeper architectures than VGG16 are prone to overfitting faster.

All of the previous images were examples of the feature maps output. For the whole collection, check out A.7

5.5 Training Results

In this section, the training results will be evaluated based on the requirements of the accuracy defined in 3.1.



(a) VGG16 trained model accuracy

(b) VGG16 trained model loss

Figure 5.10: VGG16 trained model results over 20 epochs

Figure 5.10a shows the output accuracy of the model and 5.10b the loss that decreased along the training process. The model showed convergence when it came to the difference between the training and validation hence the model wasn't either underfit or overfit.

Figure A.9a shows the output accuracy of the model and A.9a the loss that decreased along the training process. The model showed divergence too early in the training phase between the training and the validation which was early signs of prospective overfitting to occur in the model.

Architecture	**Accuracy**
DenseNet201	95.8%
VGG16	90.2%

Table 5.2: Training Accuracy Results over 20 epochs

In table 5.2, though the DenseNet201 showed a higher accuracy than the VGG16, the divergence in figure A.9a at the end showed that the accuracy result is this high due to overfitting.

5.6 Hyperparameter Tuning

Hyperparameters [15] are parameters that are set prior to the start of the learning process. Unlike the parameters of the model (weights and biases), which are learned from the data during training, hyperparameters are not learned and must be set manually. These can include learning rate, number of hidden layers, number of neurons in each layer, batch size, number of epochs, and so on. The choice of hyperparameters can significantly affect the learning process and the performance of the model.

The process of tuning the hyperparameters is done specifically to raise the model's efficiency. The algorithm used for the search of hyperparameters is the random search [5] to optimize them.

5.6.1 Finding Hyperparameters

`RandomSearch` [5] tuner performs a random search over the hyperparameter space, where the hyperparameters are defined in the `build_model` function. It randomly samples hyperparameter combinations, resulting in a broad exploration of the parameter space.

Listing 5.7: Keras Tuner RandomSearch Usage

```
1 tuner = RandomSearch(
2     build_model,
3     objective='val_accuracy',
4     max_trials=5,
```

```
5       executions_per_trial=3,
6       directory='output',
7       project_name='VGG16')
```

The parameters in the code snippet 5.7 are defined as follows [36]:

- `build_model`: This is a function that returns a compiled model. It should take a single argument, `hp`, which is used to sample hyperparameters.
- `max_trials`: This is the maximum number of different hyperparameter combinations to test. The tuner iteratively selects a new set of hyperparameters, which is how many different sets it decides to test.
- `executions_per_trial`: This is the number of models to train per trial. This can help mitigate the impact of the randomness inherent in neural network training. The final score for a trial is the average of the scores from each execution.

The output of this function is the tuner object which is used to start the search for the hyperparameters. Code snippet 5.8 shows how the search starts which an initial `epochs` count of 10.

Listing 5.8: Searching for the optimal hyperparameters

```
1 tuner.search(training_data,
2                epochs=10,
3                validation_data=test_data)
4
5 # Get the optimal hyperparameters
6 best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]
```

- The `tuner.search()` function is where the hyperparameter tuning process is performed. It tries different sets of hyperparameters to train the model and uses the validation data to evaluate the performance of each model. The performance measure here is the accuracy on the validation set. The `epochs` parameter specifies the number of times the entire training dataset is used to train each model during the hyperparameter search process.

- Once the search is done, we use `tuner.get_best_hyperparameters()` to retrieve the best hyperparameters found during the search. Here, `num_trials=1` indicates that we want the hyperparameters that produced the best model. If `num_trials` was set to a number greater than 1, it would have returned that many sets of hyperparameters, ordered by their performance.

5.6.2 Enhancement Results

The original model has been already adjusted after the hyper parameter tuning and the fully connected (FC) layers were consequently modified.

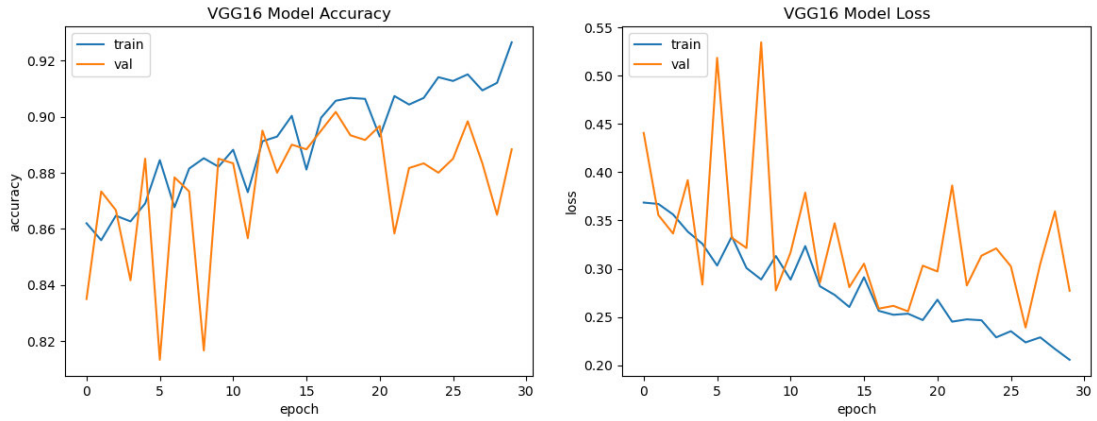
Listing 5.9: Changes done in FC layers after hyperparameter tuning

```
model = keras.Sequential([
    VGG,
    keras.layers.Flatten(),
    keras.layers.Dense(units = 768, activation = 'relu'),
    keras.layers.Dense(units = 3, activation = 'softmax')
])
```

The output of the hyperparameter search has concluded in the following changes:

- `Dense(units = 512, activation = 'relu')` → `Dense(units = 768, activation = 'relu')`
- `Dense(units = 256, activation = 'relu')` → `Dense(units = 768, activation = 'relu')`
- Optimizer learning rate has been fixed to be 0.0001

Since both `Dense` layers have been adjusted to have 768 units, having two of them would be redundant for the model and increase the learning time, so only one layer will be used as shown in 5.9



(a) VGG16 trained model accuracy

(b) VGG16 trained model loss

Figure 5.11: VGG16 enhanced hyperparameter trained model results over 30 epochs

Figure 5.11 shows the results of the hyperparameter tuning on VGG16 where the accuracy of the model during training has increased to 93.1% which is a significant increase in comparison to that before the hyperparameter tuning.

Architecture	**Accuracy Improvement**
DenseNet201	95.8% → 96.2%
VGG16	90.2% → 93.1%

Table 5.3: Training Accuracy Results over 30 epochs

Table 5.3 shows the comparison between DenseNet201 and VGG16 in the accuracy improvement where the change in DenseNet201 was quite minimal but still shows in figure A.9 the same between the training accuracy and the validation accuracy noticed before.

6 Test and Evaluation

In this section, the developed model in section 5 will be evaluated according to the accuracy resulted in the training process.

6.1 Predictions and Testing

In this section, the testing methods will be put down to be used on the two previous models and evaluate finally which model is better.

6.1.1 Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of data for which the true values are known [33]. A basic confusion matrix would look like this:

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positives (TP)	False Negatives (FN)
Actual Negative	False Positives (FP)	True Negatives (TN)

Table 6.1: Typical Confusion Matrix

Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). The terms mean the following:

- **True Positives (TP)**: These are cases in which we predicted yes (or the positive class), and the actual output was also yes.
- **True Negatives (TN)**: Predicted no, and the actual output was no.

- **False Positives (FP)**: Predicted yes, but the actual output was no. Also known as "Type I error".
- **False Negatives (FN)**: Predicted no, but the actual output was yes. Also known as "Type II error".

The Confusion Matrix forms the basis for the other types of model performance metrics, such as precision, recall, F-score, and support.

	almost_-clear	**clear**	**foggy**
almost_-clear	118	3	1
clear	22	175	3
foggy	38	3	159

Table 6.2: Confusion matrix VGG16

6.1.2 Classification Report

The classification report provides data about the predictions done on the testing set:

- **Precision** [26] is the ability of a classifier not to label an instance positive that is actually negative. For a given class, it is defined as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{6.1}$$

- **Recall** [26] (also known as sensitivity) is the ability of a classifier to find all positive instances. For a given class, it is defined as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{6.2}$$

- The **F1-score** [26] is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. It is defined as:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6.3}$$

- **Support** [26] is the number of actual occurrences of the class in the dataset. It does not change between models but instead diagnoses the evaluation process.

	Precision	**Recall**	**F1-Score**	**Support**
almost_-clear	0.92	0.93	0.92	122
clear	0.93	0.98	0.96	200
foggy	0.97	0.92	0.95	200
accuracy			0.94	522
macro avg	0.94	0.94	0.94	522
weighted avg	0.95	0.94	0.94	522

Table 6.3: Classification Report VGG16

Table 6.3 shows the accuracy of 94% of the enhanced VGG16 model in the predictions on the testing set. In table A.2, the accuracy of the enhanced DenseNet201 is 89% which proves that the VGG16 is a better model due to the lower depth of the model than that of DenseNet201.

6.2 Requirements Evaluation

The following table shows the requirements of the thesis and whether they were fulfilled.

Achieved?	Must/Could Have	Requirement
Yes	Must	The model must have at least at accuracy 90% when training the model on the training set
Yes	Could	The model must have at least 90% accuracy when testing the model on the validation data set
Yes	Must	The output of the model must be one of the three classes: **Clear**(defining that the weather is clear), **Almost Clear** (defining that the weather contains some fog but the visibility is only lightly distorted), **Foggy** (defining that the weather contains fog or heavy fog and the visibility is heavily distorted)
Yes	Must	The images of the data set must be only day-time images
Yes	Must	The pre-trained CNN architectures and networks involved in the development must be publicly available on Keras
Yes	Must	On introducing a batch of images to the model, the output to be expected is categorized according to the 3 classes mentioned in requirement 3
Yes	Must	The used software for the development of the model must be open-source software
No	Could	The images used in the training and validation phases must have a resolution no less than 640 x 480 pixels (VGA)
Yes	Must	The development of the model must be done using python as a programming language and juypter notebooks for code formulation
No	Could	The time spent to train the model per epoch (iteration) shouldn't exceed 700 seconds

Table 6.4: Requirements table showing which were fulfilled

6.3 Challenges and Outcome

To sum up, for the application of detecting fog in the meteorological activities, VGG16 can be used as a good choice for such applications.

The Challenge which was quite an issue in this thesis was the dataset. There is almost no free webcam images databases that are available to be accessed and use for such purposes.

7 Conclusion

In this paper, the target was to introduce a new method of predicting the visibility during certain meteorological activities.

In chapter 2, the paper started off by showing the fundamentals of machine learning and deep learning. The basic concept behind supervised, unsupervised and semi-supervised was briefly explained as well as neural networks where the first ever neural network "The Perceptron" was explained, the concept behind data augmentation was analysed and the convolutional neural networks. Following that, a brief overview of previously developed methods of meteorological activities detection was created which showed the state of the art of the technologies in the field.

In chapter 3, the requirements of the model were analysed in a manner of the type of input and output to be expected, the specification of the classes in the input, the expected training quality of the model to be created, how the data is to be collected to compose the three required sets and the system requirements of the machine to train and develop the model on.

In chapter 4, the available architectures on keras [2] were introduced and four of which were picked as a proposition for the solution. These four architectures' working methodology was explained as well as the optimizers that were suitable for usage. In the end, only one optimizer (Adam) was chosen for development with the four architectures being: VGG16, DenseNet201, Xception and ResNet152.

In chapter 5, the implementation overview was set and the model was created. The output of the feature maps was investigated and it was cleared up that the deeper the images go through the layers of the model, the more the model is prone to overfitting and neglecting some of the features (i.e features went undetected). Hence, the VGG16 architecture was more favorable to be the solution for the problem due to the simplicity of the dataset as well. The VGG16 was compared against DenseNet201 and the result was that the DenseNet201 model always showed signs of early overfitting. In addition to

that, the hyperparameters of both models were enhanced, yet the overfitting signs of the DenseNet201 model didn't disappear.

In chapter 6, the two enhanced models were tested to come to a result of a prediction accuracy of 94% using VGG16 and 89% using DenseNet201 which showed that the VGG16 is the suitable solution to achieve the required functionality and proves that the simplicity of the dataset rendered DenseNet201 the unfavorable architecture. In the end, the requirements of the thesis established in chapter 3 was compared against what was achieved.

Bibliography

[1] Imagedatagenerator. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator. Accessed: 2023-06.

[2] Applications - keras documentation, 2023. Accessed: 2023-05-21.

[3] Tensorflow, 2023. Accessed: June 11, 2023.

[4] Fayez M. Alazab, Abdul-Hussain S. Abdullah, Mohammed Anbar, Mohammed Saeed Jawad, and Aqeel Al-Sarray. *A Survey of Approaches for Estimating Meteorological Visibility Distance Under Foggy Weather Conditions*. IGI Global, 2019.

[5] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[6] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[7] Azar Chaabani, Faouzi Kamoun, Hichem Bargaoui, Fatma Outay, and Ansar-Ul-Haque Yasar. A neural network approach to visibility range estimation under foggy weather conditions. *Procedia Computer Science*, 113:466–471, 2017.

[8] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2006.

[9] François Chollet. Xception: Deep learning with depthwise separable convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

[10] François Chollet. *Deep Learning with Python*. Manning Publications Co., 2018.

[11] Agile Business Consortium. Moscow prioritization. Last accessed 10 April 2023.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[13] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *Journal of Machine Learning Research*, volume 12, pages 2121–2159, 2011.

[14] Lin Gao. Five class weather image dataset, 2019.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[16] Mohamed Farouk Abdel Hady and Friedhelm Schwenker. *Semi-supervised Learning*, pages 215–239. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[17] Stuart Hallowell, Jeff Hallett, and Glenn Cooney. An automated visibility detection algorithm utilizing camera imagery. In *International Snow Science Workshop*, pages 9–14, 2007.

[18] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer New York, 2009.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2016.

[21] Geoff Hinton. Neural networks for machine learning - lecture 6.5 - rmsprop: Divide the gradient by a running average of its recent magnitude. https://www.coursera.org/lecture/neural-networks/lecture-6-5-rmsprop-divide-the-gradient-by-a-running-average-of-its-recent-magnitude-ACpTQ, 2012. COURSERA: Neural Networks for Machine Learning.

[22] M. Hossari, M. John, S. Egoh, G. Machucho, K. McGuinness, and N. E. O'Connor. Weather conditions classification using convolutional neural networks. In *2020 31st Irish Signals and Systems Conference (ISSC)*, pages 1–6, 2020.

[23] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1(2):3, 2017.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[26] Scikit learn developers. Precision, recall, f-measure and support. https://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-f-measure-metrics, 2023. Accessed: 2023-06-21.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 2001.

[28] Wai-Lun Lo, Meimei Zhu, and Hong Fu. Meteorology visibility estimation by using multi-support vector regression method. *Journal of Advances in Information Technology*, pages 40–47, 2020.

[29] David G Long and David B Arnold. Seasat scatterometer: Results from the 1996 nscat in situ analysis. *IEEE Transactions on Geoscience and Remote Sensing*, 34(3):733–745, 1996.

[30] Batta Mahesh and Vadlamani Venkata Ravi Prasad. Machine learning algorithms - a review. *International Journal of Engineering and Advanced Technology (IJEAT)*, 10(1):231–243, 2020.

[31] T. M. Mitchell. *Machine Learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

[32] Neuroelectrics. Artificial neural networks: The rosenblatt perceptron, August 2016.

[33] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*, 2011.

[34] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[35] Simplilearn. Perceptron: The simplest neural network, 2021.

[36] TensorFlow. Keras tuner, 2023.

[37] Sachin Mehta Wang, Mohammad Rastegari Marvasti, and Murali Annavaram. Eespnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*. IEEE, 2018.

[38] Wikipedia. Frank rosenblatt. https://de.wikipedia.org/wiki/Frank_Rosenblatt, accessed May 2023.

[39] X. Zhu and A. B. Goldberg. *Introduction to Semi-Supervised Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2009.

A Appendix

A.1 VGG16 Feature Maps

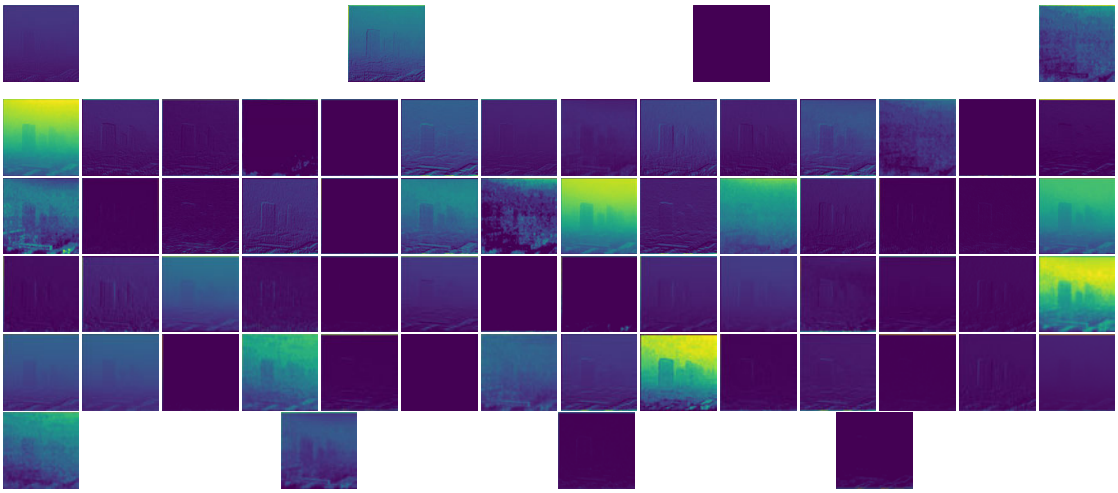


Figure A.1: VGG16 Convolutional block 1

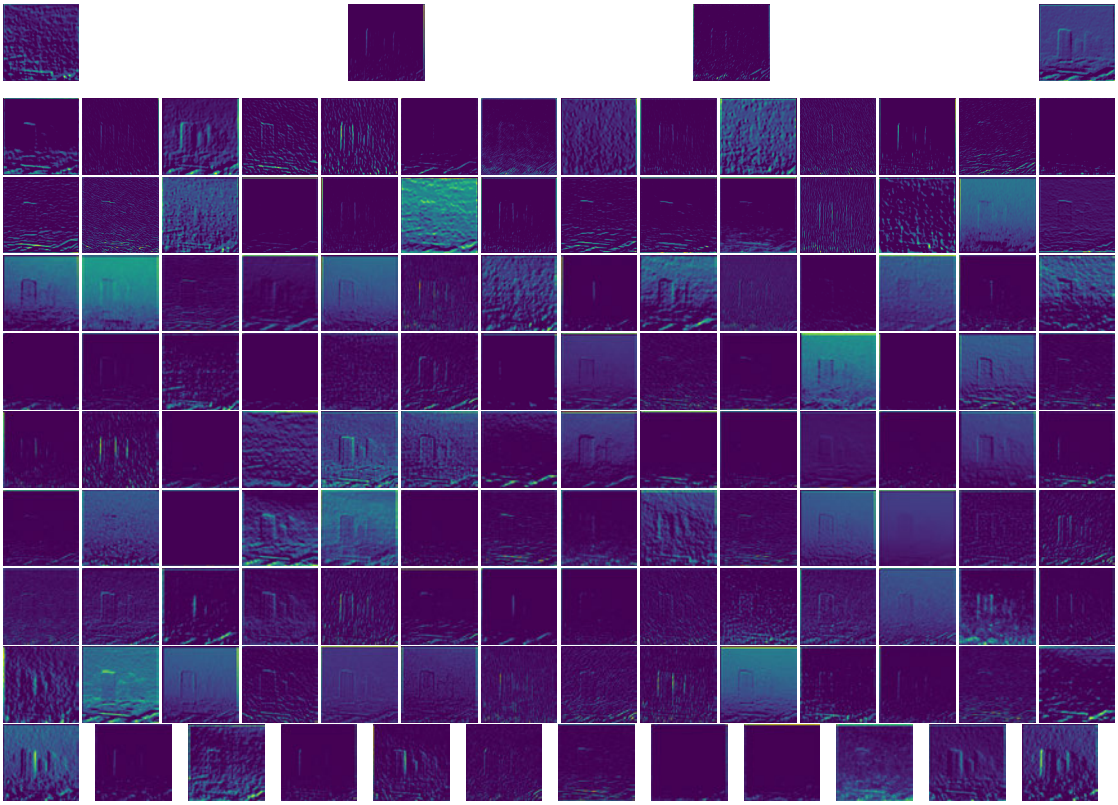


Figure A.2: VGG16 Convolutional block 2

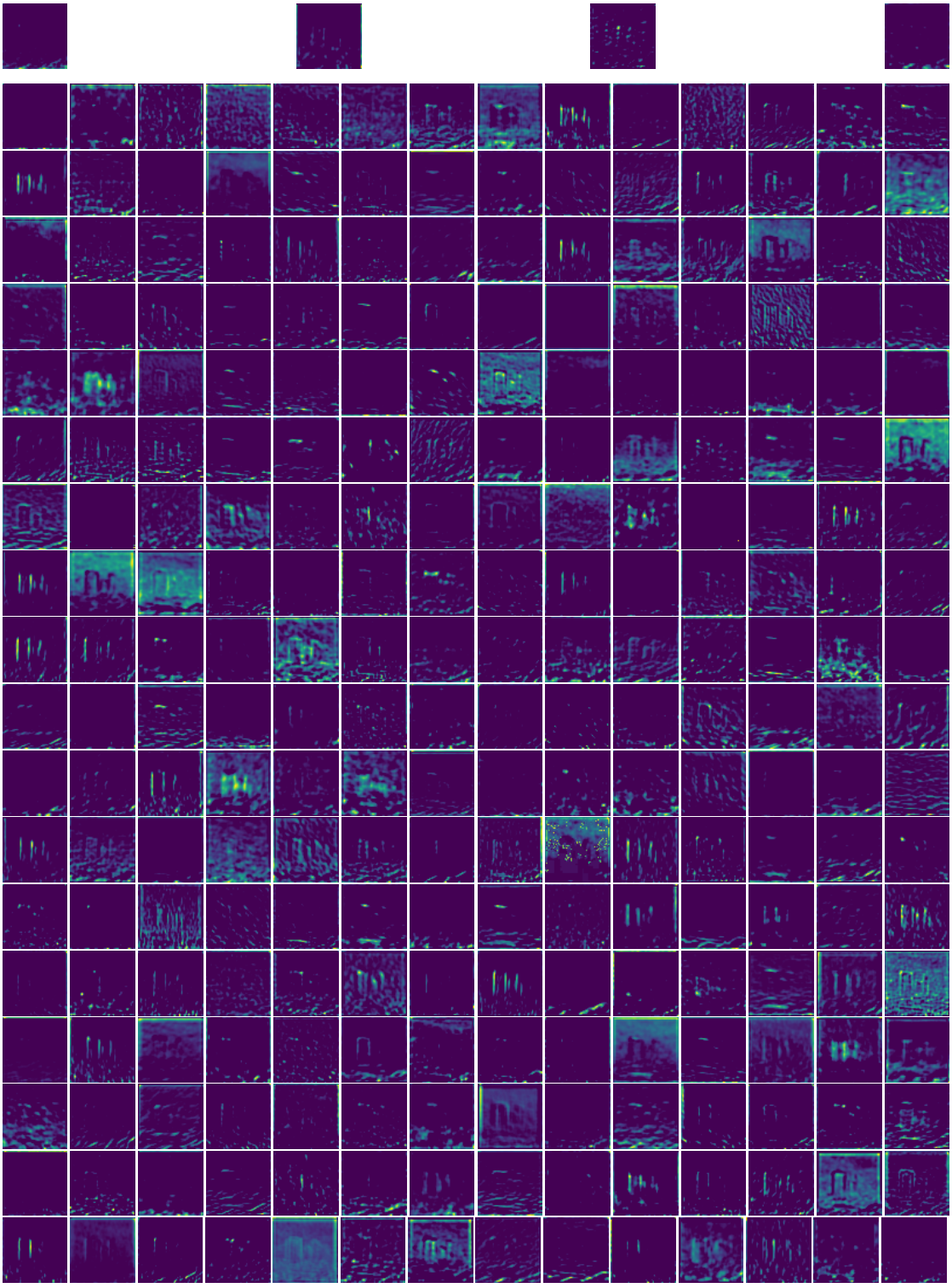


Figure A.3: VGG16 Convolutional block 3

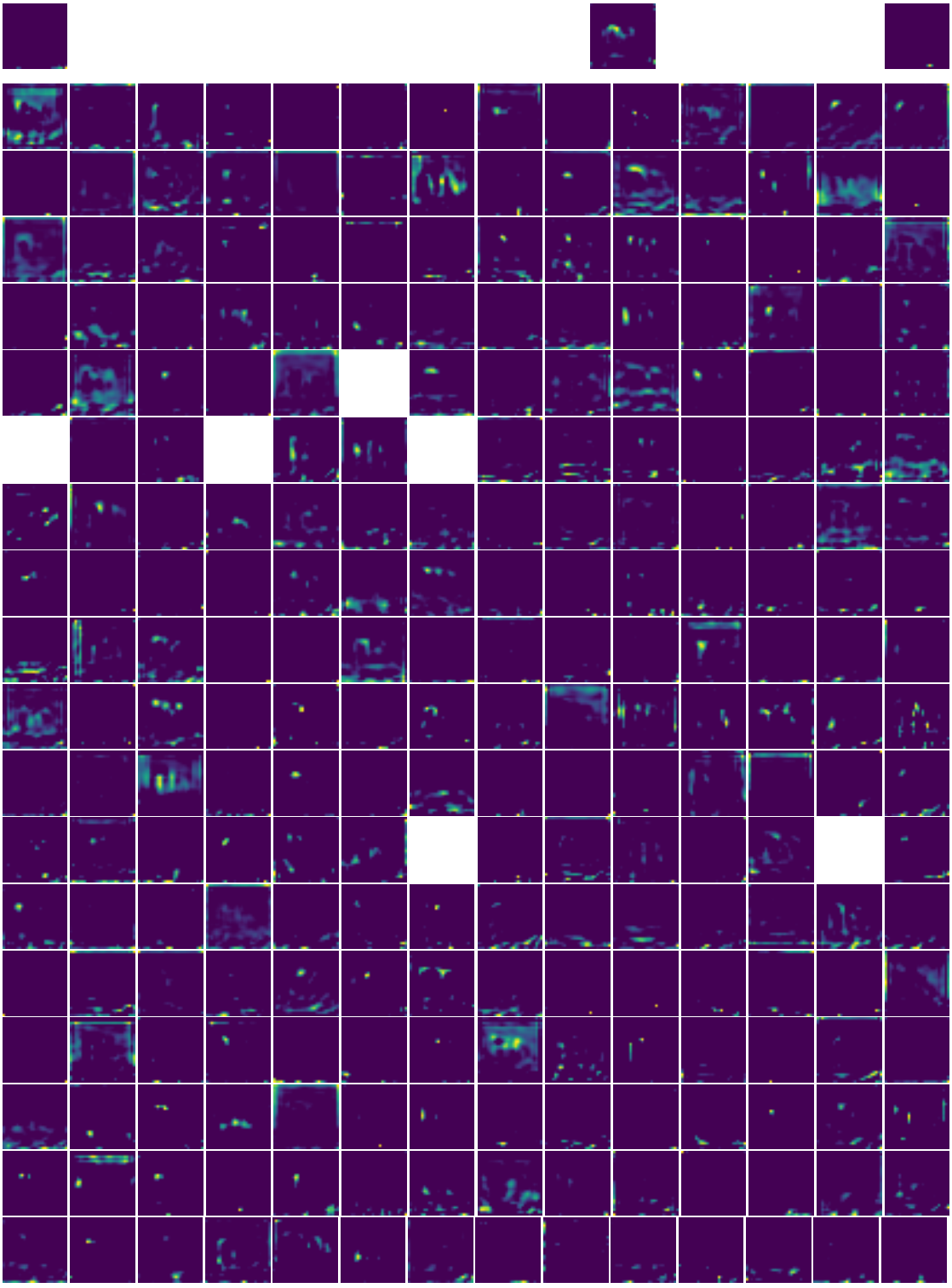


Figure A.4: VGG16 Convolutional block 4 [0 - 255]

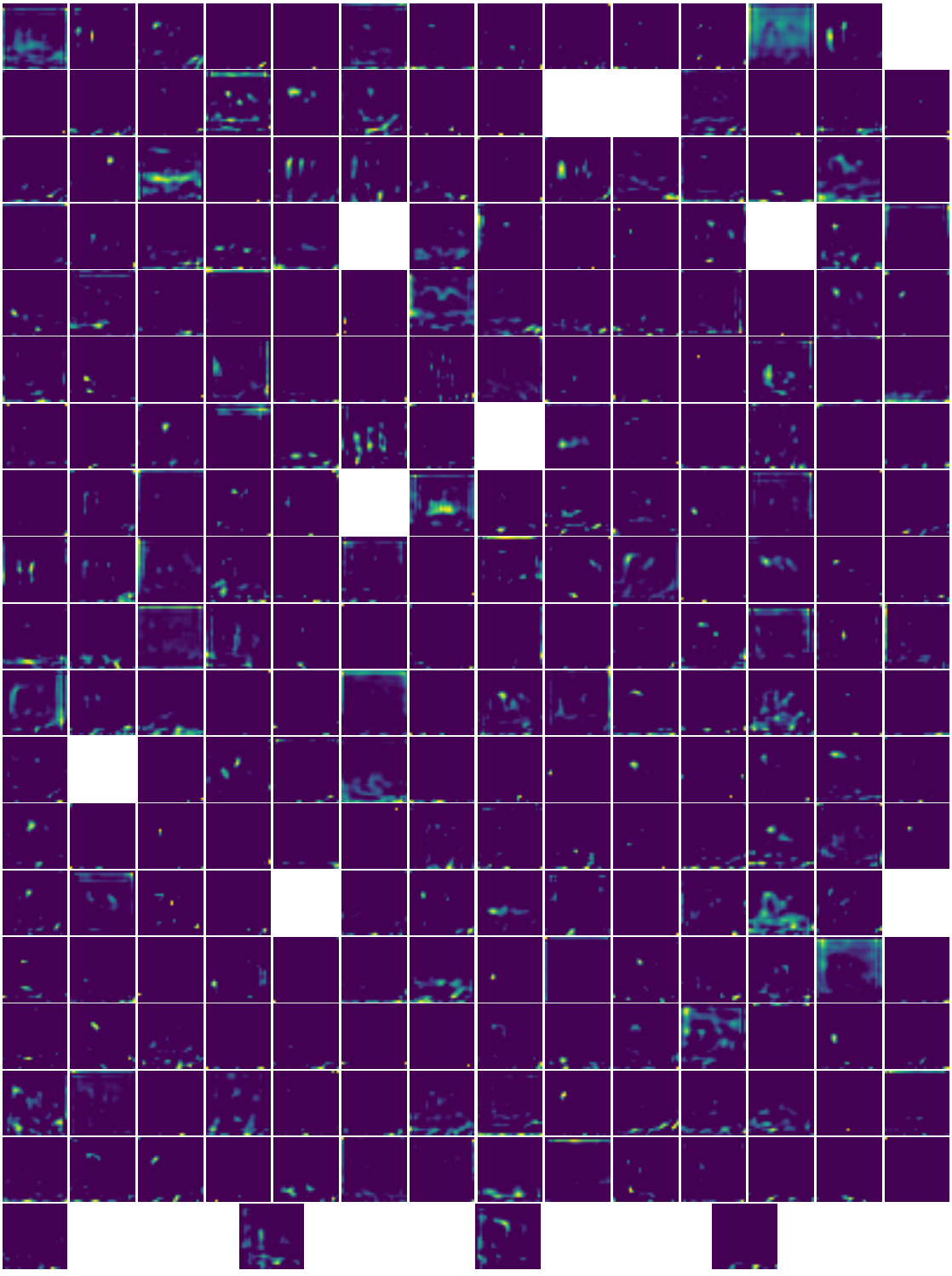


Figure A.5: VGG16 Convolutional block 4 [256 - 511]

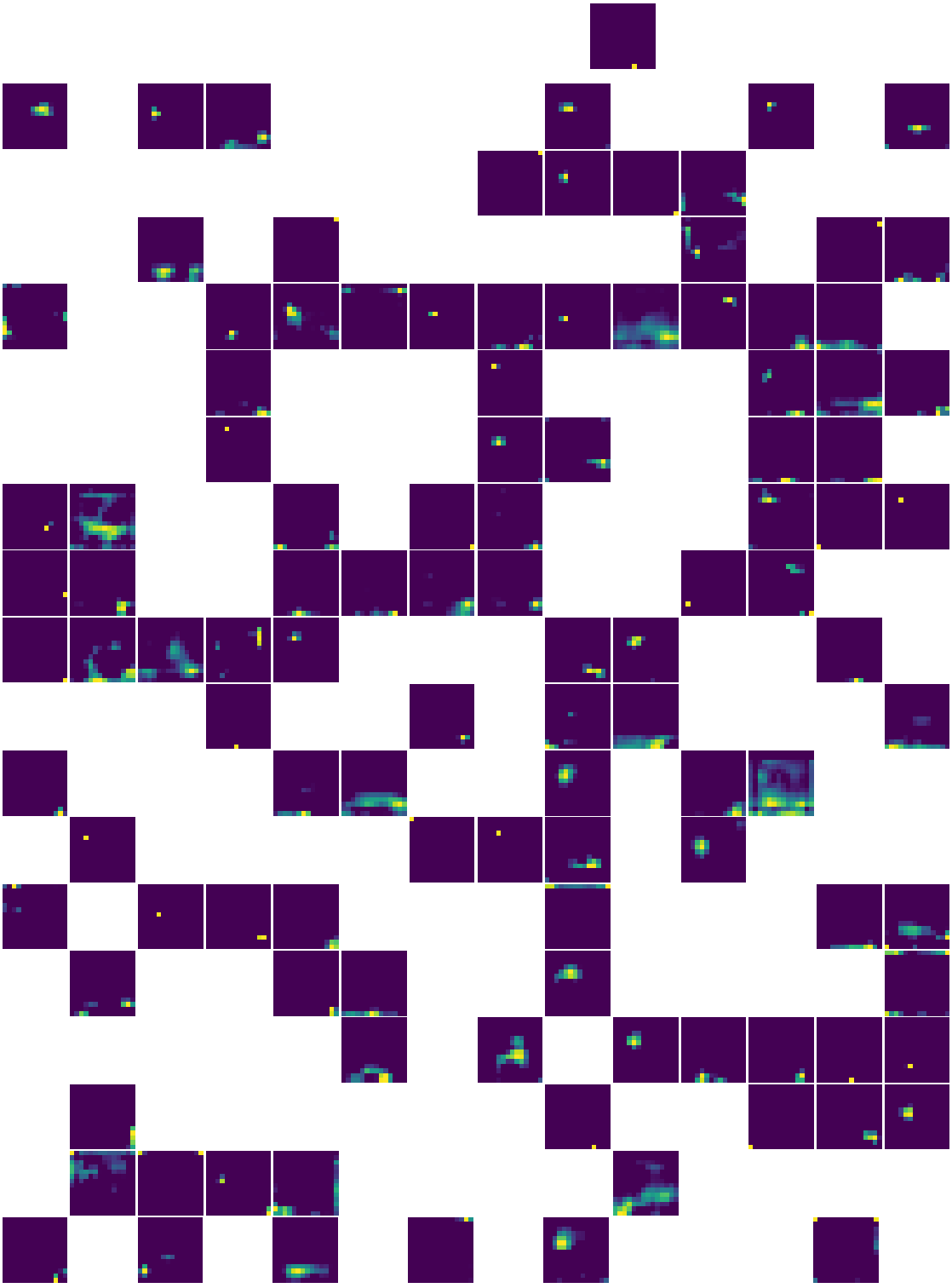


Figure A.6: VGG16 Convolutional block 5 [0 - 255]

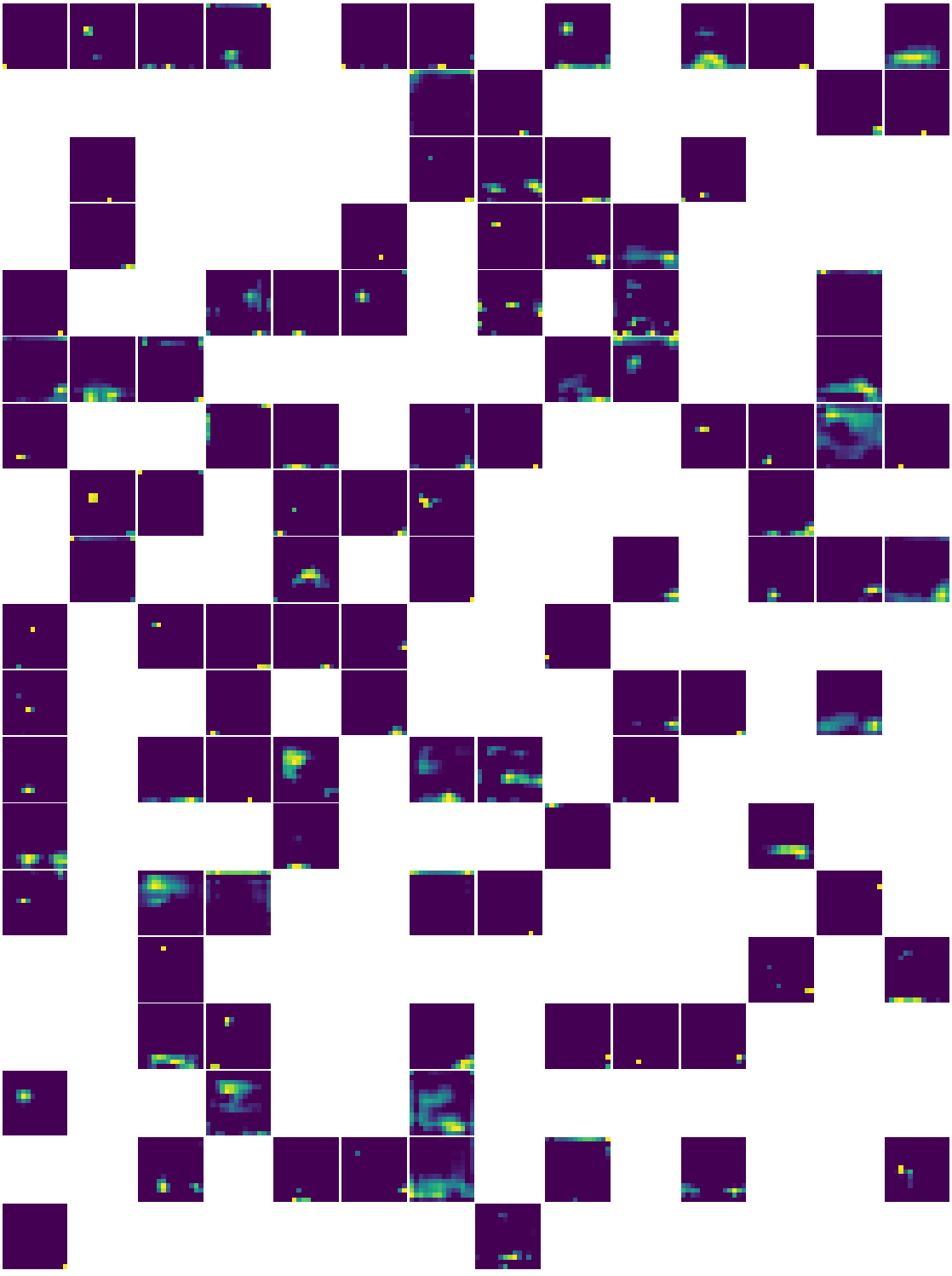
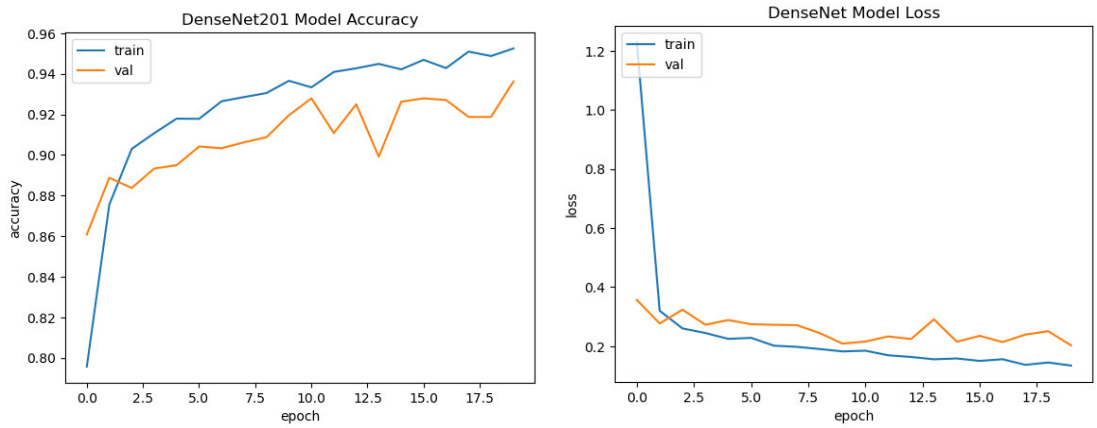


Figure A.7: VGG16 Convolutional block 5 [256 - 511]

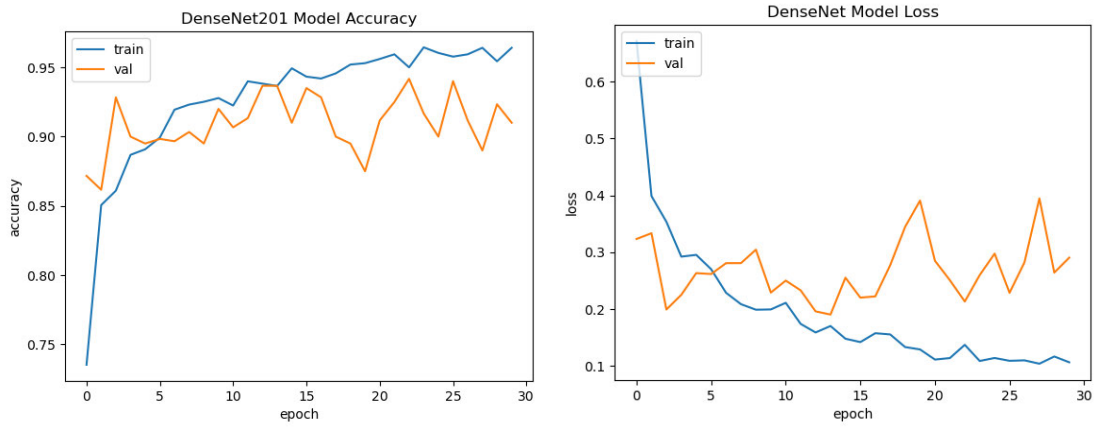
A.2 DenseNet201 Training Results



(a) DenseNet201 trained model accuracy (b) DenseNet201 trained model loss

Figure A.8: DenseNet201 trained model results over 20 epochs

A.3 DenseNet201 Training Results Hyperparameter Tuned



(a) DenseNet201 trained model accuracy (b) DenseNet201 trained model loss

Figure A.9: DenseNet201 trained model results over 30 epochs with hyperparameters tuned

A.4 DenseNet201 Confusion Matrix and Classification Reports

	almost_-clear	**clear**	**foggy**
almost_-clear	113	7	2
clear	0	199	1
foggy	26	19	155

Table A.1: Confusion matrix DenseNet201

	Precision	**Recall**	**F1-Score**	**Support**
almost_-clear	0.81	0.93	0.87	122
clear	0.88	0.99	0.94	200
foggy	0.98	0.78	0.87	200
accuracy			0.89	522
macro avg	0.89	0.90	0.89	522
weighted avg	0.90	0.89	0.89	522

Table A.2: Classification Report DenseNet201

Declaration

I declare that this Bachelor Thesis has been completed by myself independently without outside help and only the defined sources and study aids were used.

City Date Signature