Paulo Rafael Soares Oliveira Vieira

Open-World Active Learning in Self-Driving Cars

UNIVERSIDADE D COIMBRA
1 2 9 0

1 2  9 0

UNIVERSIDADE D
COIMBRA

Paulo Rafael Soares Oliveira Vieira

Open-World Active Learning in Self-Driving Cars

VOLUME 1

Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems, advised by Professor Luís Macedo and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September 2022

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

Paulo Rafael Soares Oliveira Vieira

Open-World Active Learning in Self-Driving Cars

Dissertation in the context of the Master in Informatics Engineering, specialization in Intelligent Systems, advised by Prof. Luís Macedo and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September 2022

Acknowledgements

I would like to thank my supervisor, Luís Macedo, for his continuous availability, assistance and suggestions throughout the development of this work, which was crucial for its success.

Also I would like to thank my family for the constant support given, motivating me through the toughest times, and my closest friends who accompanied me on this adventure.

At last but not least, I would like to thank all my colleagues and teachers from my graduation and master's degree that helped over the course of this last 5 years.

Abstract

Self-driving cars are placed in an environment with many different objects, where for each of these objects, the intelligent agent driving the car has to receive many example images. Having someone constantly labeling each object is an expensive and time-consuming process, and even in the unlikely event that all objects are labeled, there are some for which it is difficult to get examples, such as goods lost by a truck. In order to adjust to this environment and be able to perform the safest action for a set of conditions, causing the least damage, an agent of this type must be able to detect and learn these objects/classes that it has never seen, thus becoming an open-world learning model. It should be noted that sometimes the agent may be in contact with different unknown classes, and it then has to be able to identify that many classes. It should be noted that sometimes the agent may be exposed to different unknown classes at once, therefore it has to be able to identify that many classes.

In this project, we implemented a model capable of not only detecting samples belonging to unknown classes, but also identifying the number of classes hidden in these samples. This model consists in training four Convolutional Neural Network (CNN), in order to create a network capable of correctly classifying most samples, and then use them together with the OpenMax algorithm to detect samples belonging to unknown classes. Finally, the samples detected by OpenMax are used in the Density-based Spatial Clustering of Applications with Noise (DBSCAN) algorithm to obtain the number of unknown classes.

To measure the quality of the model, we calculated the percentage of each class per group created by the DBSCAN algorithm. This way, we can see whether each detected unknown class is well represented and if its samples can be used for training.

Our results show that, although we can get the number of groups equal to the true number of unknown classes, the samples from each group do not represent exactly one class, i.e. each group may contain more than one of the true unknown classes. For example, for one of the test data sets with 8 unknown classes, one of the groups contains 5 of these. Nevertheless, this method proved to be capable of solving, since in some situations it was able to distinguish some classes and group similar classes together.

Keywords

Open-World Artificial Intelligence, Active Learning, Unknown Classes Distinction

Resumo

Os carros autónomos estão inseridos num ambiente com muitos objetos diferentes, sendo que, para cada um destes objetos, o agente inteligente capaz de conduzir o carro tem de receber muitas imagens exemplo. Ter alguém constantemente a etiquetar cada objeto é um processo caro e demorado e mesmo na eventualidade de se conseguir etiquetar todos os objetos, existem alguns para os quais é dificil de conseguir exemplos, como é o caso de mercadoria perdida por um camião. Um agente deste tipo, para se ajustar a este ambiente e ser capaz de executar a ação mais segura para um conjunto de condições, provocando o menor dano possivel, tem de ser capaz de detetar e aprender estes objetos/classes que nunca viu, tornando-se assim num modelo de aprendizagem em mundo aberto. É de notar que por vezes o agente pode estar em contacto com diferentes classes desconhecidas, tendo este então de conseguir identificar essa quantidade de classes.

Neste projeto, implementamos um modelo capaz de não só detetar amostras pertencentes a classes desconhecidas como também detetar o número de classes escondidas nestas amostras. Este modelo consiste em treinar quatro Redes Neural Convolucional (RNCs), de modo a se conseguir criar uma rede capaz de classificar corretamente a maioria das amostras, e, posteriormente, usá-las em conjunto com o algoritmo OpenMax para se conseguir detetar amostras pertencentes a classes desconhecidas. Finalmente, as amostras detetadas pelo OpenMax são utilizadas no algoritmo Agrupamento Espacial Baseado em Densidade de Aplicações com Ruído (AEBDAR) para se obter o número de classes desconhecidas.

Para medir a qualidade do modelo, calculamos a percentagem de cada classe por grupo criado pelo algoritmo AEBDAR. Desta forma, conseguimos perceber se cada classe desconhecida detetada está bem representada e as suas amostras podem ser utilizadas para treino.

Os nossos resultados mostram que, embora se consiga obter o número de grupos igual ao verdadeiro número de classes desconhecidas, as amostras de cada grupo não representam exatamente uma classe apenas, isto é, cada grupo pode conter mais do que uma das verdadeiras classes desconhecidas. Por exemplos, para um dos conjuntos de dados de teste com 8 classes desconhecidas, um dos grupos contém 5 destas. De qualquer das formas, este método provou ser capaz de resolver, uma vez que, em algumas situações, conseguiu distinguir algumas classes e agrupou classes semelhantes.

Palavras-Chave

Inteligência Artificial em Mundo Aberto, Aprendizagem Activa, Distinção de Classes Desconhecidas

Contents

Acronyms

AI Artificial Intelligence.

API Application Programming Interface.

CAP Compacting Abating Probability Model.

CNN Convolutional Neural Network.

CV Class Verification.

DBSCAN Density-based Spatial Clustering of Applications with Noise.

DNN Deep Neural Network.

ML Machine Learning.

NNDR Nearest Neighbor Distance Ratio.

OCN Open Classification Network.

OSNN Open-Set Nearest Neighbor.

OvS 1-vs-Set Machines.

PCN Pairwise Classification Network.

SVM Support Vector Machines.

W-SVM Weibull-calibrated SVM.

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we introduce the subject of this dissertation. Firstly, we present an initial motivation, followed by the research question and how will it be addressed. Afterwards, we describe the main contributions of our thesis and close with the organisation of this document.

1.1 Motivation

Artificial Intelligence (AI) can either be a field of study or capacity. As a field of study, it can be described as the field concerned with understanding and building machines with some form of intelligence (it may or may not be human intelligence). As a capacity, it may be portrayed as the machines ability to exhibit intelligence in problem solution and performing tasks. Russel and Norvig [1] assume one category to define the intelligence exhibited by a machine, being this the success in terms of human performance versus the ideal concept of intelligence, to which they call rationality. The first one, success in terms of human performance , is an "empirical science, involving hypothesis and experimental" while the second, success in terms of rationality, is combining mathematics and engineering. The authors make this distinction because humans make mistakes and their reasoning takes into account their emotions.

The constant research and improvement of this subject throughout the years allows it to be applied today in all industries, from healthcare to marketing and from cars to surveillance and security, improving the efficiency of a task, automating repetitive work, and assisting in decision-making situations.

One of the main areas of AI is Pattern Recognition, which is the ability to recognize patterns in data, and potentially detect multiple classes, where patterns is defined as similarity between instances for a particular characteristic and a class is the group of instances similar in all characteristics. The example of two fruits, red apples and cherries, can be used to simulate the process done by a machine during pattern recognition. When only taking into consideration the color and shape of the fruits, it is possible to state that all instances are similar, existing just one big class, but when a third feature, size, is added, two different classes

emerge, big red apples and small red apples (cherries).

There are two main approaches to teach an artificially intelligent agent to perform this recognition. One is unsupervised learning [1], in which the agent receives data without any labels, unaware of the number of different classes in the environment. Therefore, it detects patterns by grouping instances according to their similarity with each other. The closer an instance is to a group, the higher chance it has to be a part of it. In opposition to the previous approach, the supervised learning approach considers that the learner receives a data set comprising labeled instances. Thus, the learner knows how many and which classes exist in the environment.

The latter approach has been quite effective in the scenarios in which it has been applied [2; 3], but a model trained with typical supervised learning lives in a closed world. Returning to the example of fruit with two classes, red apples and cherries, and three features, color, size and shape, with closed-world learning, the model perceives which feature values are inherent to apples and which to cherries. After its training, whichever of the fruit pieces it is, the model is able, to some extent, to distinguish between the two.

However, when the model is applied to the real world, some problems can arise with this learning when there is a lack of control. If a new unknown class is provided to the model, it is unable to perceive that it is facing an instance of that new class. Should this be a fruit similar to those provided during training, such as a green apple or a peach, where the greatest variation occurs in one or two of the features, although it can only state whether it is a red apple or a cherry, it may be able to state the most similar piece of fruit yet not by association. Otherwise, if that is a completely different fruit, such as a banana or a fig, the response given by the model seems random, being that it outputs one of the two trained classes with lots of uncertainty.

For a fruit classification problem, it is feasible to keep the learning and classification processes under control so that the model never receives instances of unknown classes. Nonetheless, there are real-world situations where assuming that models with closed-world learning will perform well is unrealistic since they are mostly exposed to dynamic systems with various uncertainties and need constant updating of their knowledge. Such models applied in dynamical systems can be commonly found in healthcare or automobiles.

Open-world artificial intelligence [4–6] can solve the problem that supervised learning models face in systems that are constantly changing or full of uncertainty. It allows the model the ability to introduce new classes into its knowledge, classes that were unseen during the training phase, without the need for retraining. In this way, the model learns incrementally.

A great example of open-world learning application is the classification of the elements that an autonomous car [7; 8] detects in its surroundings. This car is an intelligent vehicle capable of autonomously moving on the roads, and for that, it needs to recognize the various elements around it as the ones that can ever be in its field of view. To understand why open-world learning is applied in autonomous cars, it is necessary to attempt to name all the elements that it might

see. The two mandatory elements the agent has to recognize are the other cars and people outside so that the journey is safe and smooth, and no one is harmed. It also has to recognize buildings and trees so that, in a road without white lines (the guidelines for autonomous cars), the car is to see that it has to act when one of these is near. As traffic signs dictate the behavior that the driver should follow in a given region or street, it is crucial to make the agent know them too. Here the difficulty increases due to the variety of traffic signs. These might be about directions, warnings, information, and others, where each one comprises multiple symbols, for instance, in danger signs, there are symbols such as falling or fallen rocks, slippery road, or road narrows on both sides, and many others. In addition to all these noted elements and the more that can still be added, there are elements such as a lost tire or goods and even animals that can appear in front of the vehicle. This latter ones corresponds to the majority of situations with uncertainty in the environment. Since there is so much diversity in the elements needed for annotation and so many others that are rare, possibly discarded for model training, an open-world learning approach allows for simplification both in structure and in knowledge aggregation. Thus the model is able to detect elements it has never seen before and, if in an iterative learning phase, add them to its knowledge base or, if in an implementation phase (e.g., when it detects a lost box in the middle of the road), act accordingly by steering away or stopping.

1.2 Research Question

From the above considerations, open-world AI provides several advantages when it comes to the implementation of an agent in an environment full of uncertainties. Such an agent can adopt two types of behavior. One behavior rejects all instances corresponding to the unknown class while the other learns these instances by expanding its knowledge so that, at a later stage, when it is provided with more instances of this new class, the agent can already identify them as known.

Formalizing this idea, we obtain the following. Let $\mathcal{X} = \{x_1, x_2, x_3, ..., x_n\}$ be the training data instances and $\mathcal{Y} = \{y_1, y_2, y_3, ..., y_n\}$ be the labels of the instances $\mathcal{X}$, where n is the total number of training instances in the data set. In this case, the element x_i from $\mathcal{X}$ at position i is represented by the label y_i from $\mathcal{Y}$ at the same position i. The $\mathcal{Y}$ labels belong to the known classes set $\mathcal{C} = \{c_1, c_2, c_3, ..., c_k\}$, where k is the number of known classes.

The aim of open-world recognition is to create a model able to detect the elements of a test data set $\mathcal{T} = \{t_1, t_2, t_3, ..., t_m\}$, in which the set of labels is defined by $\mathcal{L} = \{l_1, l_2, l_3, ..., l_m\}$, that do not belong to the set $\mathcal{C}$ of known classes, should there be any, adding them to the rejected $\mathcal{R}$ set, while still correctly identifying the others based on its training with $\mathcal{X}$.

As described above, open-world learning goes a step further and learns the instances in $\mathcal{R}$. In other words, being r_i an instance of $\mathcal{R}$, where $r_i \notin \mathcal{C}$, a new class c_{k+1} is created and added to the known classes $\mathcal{C}$ set. The model is then trained with $\mathcal{R}$ and its labels $\{c_{k+1}, c_{k+1}, c_{k+1}, ..., c_{k+1}\}$.

In the literature [4; 5; 9], it is usually assumed that the instances in $\mathcal{R}$ belong all to the same unknown class. However, there might be a set $\mathcal{U} = \{u_1, u_2, u_3, ..., u_j\}$ of unknown classes to which the labels of $\mathcal{R}$ belong to, i.e., $\mathcal{R}$ might have instances corresponding to more than one unknown class.

1.3 Approach

To accomplish the steps formalized in the previous section, it is necessary to divide the work into two main stages which are the detection of instances belonging to an unknown class and the division of those instances into smaller sample classes, to later be included in the known set.

1.3.1 Implement a machine learning model

In order for a model to detect instances that do not belong to any of its known classes, it firstly needs to be trained with that known set of classes, so, as this work is targeted at self-driving cars that use images, it is required to implement a CNN capable of performing image classification on a custom data set.

Being this step the base for the two previously mentioned stages, where its performance affects their outcome, and as there are no restrictions or mandatory rules to follow while building a CNN for a given problem, it is important to consider multiple different networks, detailed in Subsection 4.4.1, in order to attain a good one.

1.3.2 Detection of instances belonging to an unknown class

To expand the CNNs to an open-world classification, their outcome must be treated in a way to acknowledge an extra (*unknown*) class. To accomplish this, due to reasons presented in Subsection 4.4.2, we chose to implement the method developed by Bendale and Boult [9] called OpenMax.

Although we can measure the performance of the CNNs individually and apply afterwards the best performing one to OpenMax, we ignore the influence that they have on each other. Thus, we apply the open-world method on each neural network and the ensemble with the best results advances onto the next stage. The quality measure of the ensemble is both the f-score and the number of correctly classified instances belonging to the unknown class.

1.3.3 Division of the unknown class into smaller classes

The last stage is about receiving the instances belonging to an unknown class and trying to find as many different classes as possible within those instances. To accomplish this, we opted for the DBSCAN algorithm since it creates the classes by

clustering the instances without requiring a prior knowledge about the number of classes in the data set provided.

To evaluate the quality of this stage, it is taken into account both the number of clusters found as well as the percentage of instances of each true class in every cluster.

1.4 Contribution

This work aims to investigate techniques capable of identifying multiple hidden classes in samples belonging to an unknown class. To this end, we built the hybrid model described in the previous subsection with two main purposes: finding as many clusters as there are hidden classes within the samples classified as belonging to an unknown class, with each cluster being associated mostly one class. This way, we explore deeper the goal of the work [10], looking for a model with better performance in identifying the amount of unknown classes discovered, based on another quality control variable (percentage of each class per cluster).

To do this, the model was tested with several parameters. With this, we concluded that, for a given combination of these, it is possible to find the true number of classes present in the set of samples belonging to the unknown class. However, it cannot fully isolate one class per cluster, clustering sometimes similar classes and even pairs of classes that have nothing to do with each other. Nonetheless, through this method, it is possible to understand that only identifying the number of unknown classes does not imply that the instances of each cluster can be used for active learning.

1.5 Structure

The structure of the document starts with Chapter 1 introducing the subject of this work, the motivation, research question and approach followed by the contributions. Subsequently, in Chapter 2, we provide the theoretical knowledge necessary to understand the concepts presented throughout the paper. Following this is Chapter 3, where we describe some of the State-of-the-Art approaches in open-world AI. Afterwards, in Chapter 4, we establish both the environment in which the work is developed as well as the chosen data set and its treatment, and the model architecture. Then, in Chapter 5, we describe the experimental process in detail, from the machine and environment specifications to class selection and model training and testing, continuing with Chapter 6.1, where we present the experimental results, analyse and compare them. Finally, in Chapter 7, is given a summary of the work developed along with a future work.

Chapter 2

Background

This chapter aims to clarify some concepts of artificial intelligence, some broader than others, discussed throughout the paper. To this end, the necessary theoretical knowledge is explored.

2.1 Artificial Intelligence

Since the topic is concerned with intelligent systems, an introduction to artificial intelligence is necessary. The term "artificial intelligence" was first officially used by John McCarthy in his workshop at Dartmouth in the summer of 1956 [11]. The workshop was based on the idea that all the characteristics of learning and intelligence could be described in such detail that it would be possible to simulate such behaviour in machines. In the workshop proposal, the authors stated the goal as "An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves".

But, before looking into the aim of this area, it is imperative to formally define the term "artificial intelligence", a task that is not easily done. Over the years, several authors have developed their definitions, among which we find Kurzweil [12], who defines it as "The art of creating machines that perform functions that require intelligence when performed by people", Bellman [13], who states "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning...", Winston [14], who claims "The study of the computations that make it possible to perceive, reason, and act.", Poole et al. [15] with "Computational Intelligence is the study of the design of intelligent agents", Nilsson [16] who wrote "AI ... is concerned with intelligent behavior in artifacts".

Russel and Norvig [17], based on various definitions to that time, built their idea of "artificial intelligence" by dividing the concept according to two main criteria. One criterion involves the process of representing the outcome, dividing it into either thought process or behaviour and the second criterion covers performance regarding whether the final result is close to the result obtained by the human

or the ideal/rational result. The rationality referred to is due to the reasons that explain why not all people get the same best marks in exams or are good at chess. Thus we have the four definitions, “thinking humanly”, where Bellman’s thought fits in, “thinking rationally”, framing Winston, “acting humanly”, corresponding to Kurzweil's idea, and finally Poole and Nilsson represented by the last definition “acting rationally”. Taking this into account, Russel and Norvig argue that the concept of artificial intelligence should be mostly focused on the last definition, “acting rationally”, since, ideally, an intelligent agent should always opt for the best action when faced with a problem.

With the term “artificial intelligence” introduced, we return to the goal of this field. As McCarthy indicated, this area seeks to make machines capable of demonstrating intelligence to solve problems and evolve. However, there are two types of AI: weak and strong. Weak AI is limited by functionality, giving the impression of intelligence. Using sophisticated algorithms, agents of this type manage to handle important problems, but are nevertheless limited to the problem for which they were developed. The current state of AI is of this type, with examples such as voice assistants [18] and recommendation systems [19]. Strong artificial intelligence, on the other hand, is the idea of agents capable of possessing their own intelligence by showing traces of consciousness and feelings. These agents are, for now, hypothetical, with one example of their existence being an agent that manages to fool the interrogator in the Turing test [20].

The objectives of this topic reveal that AI is a very broad area of study. An artificial intelligent agent may have as target the detection of an object in an image or the sentiment analysis of a sentence or dialogue. From this emerge the several fields that this area employs, from computer vision to natural language processing. In this chapter, there is a section dedicated to one of those fields, that being machine learning, since it is necessary that the machine learns to distinguish known objects and detect unknown objects through given data.

2.2 Machine Learning

The concept of Machine Learning (ML) was firstly introduced in the work of Arthur Samuel, where his idea was “A computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program” [21]. His approach was based on algorithms that measured the value of every move, at any time, to make the best move. A few years later, the author published a second paper where he generalized his idea by stating that “Programming computers to learn from experience should eventually eliminate the need for much of this detailed programming effort” [22]. Another researcher, Tom Mitchell, shares, to some extent, Arthur Samuel's ideas stating that ML is about finding the answer to “how to construct computer programs that automatically improve with experience?” [23].

With this in mind, it can be said that ML is the autonomous construction of a knowledge representation by an algorithm, giving the idea of human-like self-organised learning. The construction of the knowledge representation comes

from training the algorithm with a set of initial data, called training data set. Once trained, this representation must be as general as possible so that it may accurately classify or make predictions on a new batch of data never seen before while still being able to identify instances from the training data.

However, there are cases where the training performs well, but because it is trained on examples that do not fully represent the population, the model is unable to generalize and underperforms for unseen instances. Such an event is known as overfitting, which can be prevented in different ways, but as these are not the focus of this study, they will not be discussed.

As ML consists in the self-learning of a machine, how is it done? Well, there are many types of learning, but this section will only focus on the ones used in this work, such as supervised, unsupervised and active learning, or related to it, as it is the case of reinforcement and online learning.

2.2.1 Supervised learning

The term "supervised learning" comes from the idea of providing the agent some form of feedback during its learning process.This translates into providing both the input data and the corresponding labels for each instance in that data [24]. With this learning method, the model analyses the characteristics of the input instances and tries to correlate it with the desire output, this being the target labels. An illustrative example of supervised learning is a data set comprising 100 images, where one-half of these represent dogs and the remaining half represent cats. For each image provided to the model, a correspondence is made between the image content and the image label, which represents the animal included in the image.

As the papers that comprise the state-of-art of this topic, mentioned in Section 3, and as this thesis use supervised learning algorithms, they are going to be explored in this subsection. The algorithms are Support Vector Machines (SVM) and Deep Neural Network (DNN), particularly CNN.

Support Vector Machines

SVM [25] training technique consists in positioning all the instances of a data set in a k-dimension space, where k is the number of features in the data set, and finding the function (there may be more than one) that better separates the existing classes.

To explore this algorithm further, it will be applied to a simple example of apples. The example consists in separating green apples from red apples through the characteristics/features: ratio between green and red pigmentation and width. Displaying 15 randomly generated instances of apples on a two-dimensional plane, we obtain the Figure 2.1.

Note that it is possible to separate the two classes by a threshold, as long as it

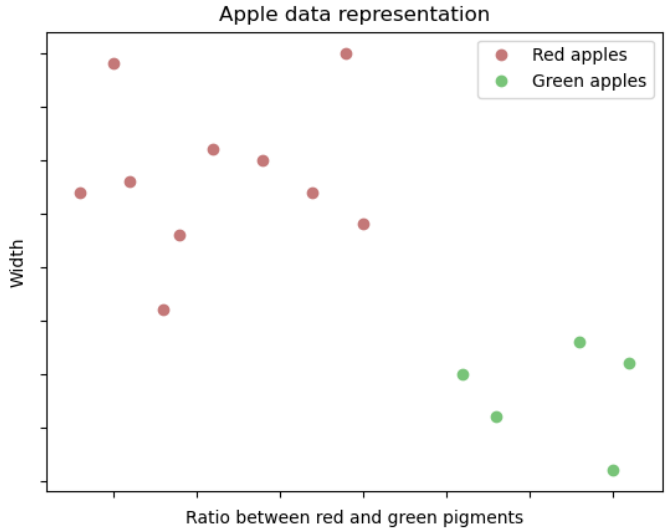


Figure 2.1: Apple data representation in two-dimensional plane

is between the instances at the extremity of each class, also known as support vectors. Thus, when a new point emerges on the left side of the threshold, it is classified as a red apple, otherwise it is classified as a green apple.

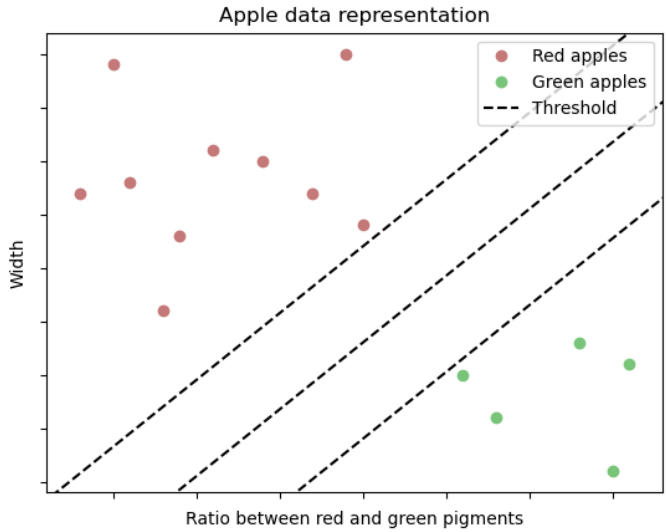


Figure 2.2: Apple data representation in two-dimensional plane with three possible thresholds that divide the two classes

Nevertheless, the choice of the threshold location requires attention. Assuming a new point near the red apple group, as shown in the Figure 2.3, it is more likely that this new instance belongs to the red apple class. However, using only the three thresholds displayed in the figure, if the threshold on the left is chosen, the instance is declared to belong to the green apple class.

To tackle this problem, we define the threshold where the distance between the support vectors is maximum. This distance is called the margin. Yet, this ap-

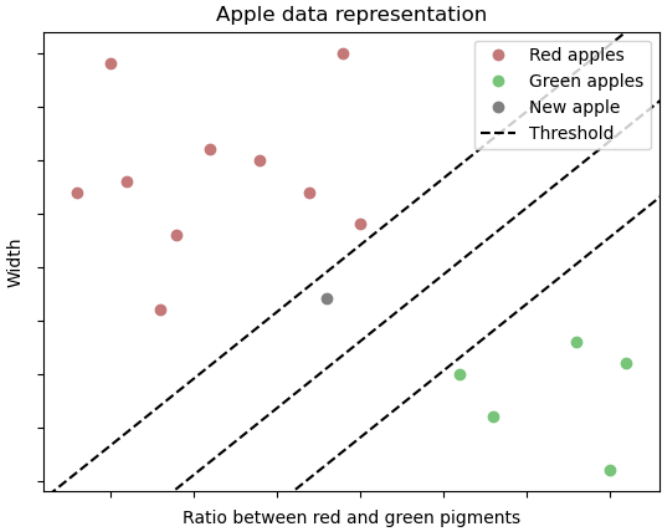


Figure 2.3: Apple data representation in two-dimensional plane with three possible thresholds that divide the two classes and a new instance

proach is sensible to outliers. Adding an instance of green apple near the group of red apples, as it can be seen in Figure 2.4, in addition to forming a threshold too close to the support vectors, it does not represent well the space of each of the classes, leading to misclassifications.

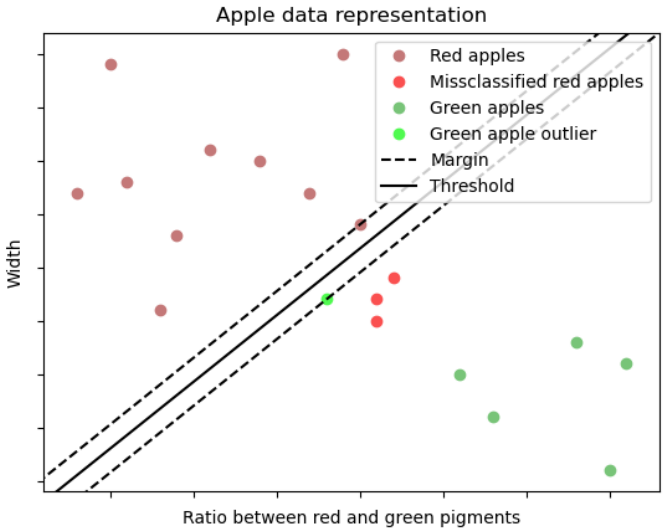


Figure 2.4: Apple data representation in two-dimensional plane with a threshold and misclassified instances

To decrease the large amount of unwanted outliers coming from threshold misplacement, instead of using the maximum margin strategy, one implements a soft margin, which allows misclassification during training.

To find a threshold with the desired soft margin, we first look for the pair of instances from different classes with the smallest distance, marking these two as

extremes of the respective classes. Then, for each of these extremes, we look for the neighbouring instance belonging to the same class and calculate the distance between them. The extreme-neighbour pair with the smallest distance is selected and the neighbour is considered the new extreme. Finally, the threshold is calculated using the maximum margin between the two extremes. These steps of extremes and threshold calculation are repeated until the desired ratio of misclassified instances is within the margins. An example of a final result is provided by Figure 2.5.

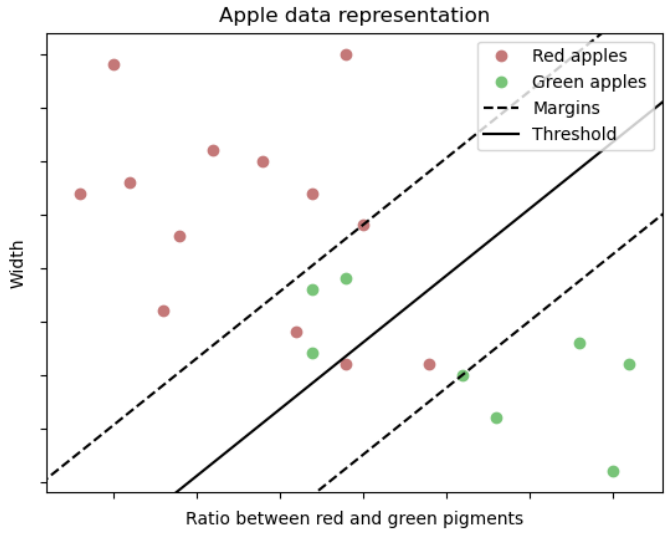


Figure 2.5: Apple data representation in two-dimensional plane with a threshold and soft margins

The threshold is, in this particular problem, a line but, when it comes to larger dimensions, it is a hyperplane described by a function called a kernel.

An SVM may also encounter problems that are not linearly separable. For that, the kernel trick is used. This trick consists in increasing the number of dimensions of the problem so that it can be linearly separable. Increasing the number of dimensions occurs by performing pairwise similarity comparisons on the original data, instead of transforming the data since it would be very costly to find the most appropriate transformation function. On the new data, the previous concepts are applied to obtain the kernel function.

Deep Neural Networks

To understand the architecture of CNNs, it is required an introduction to DNNs, which is a network of neurons arranged in layers, as shown in Figure 2.6.

Each neuron in a layer is connected to all neurons in the previous and following layers. The importance of these comes from the weights and biases associated with it. A neuron in a given layer has a specific weight and bias for their link with each neuron in the previous layer. It is through the expression 2.1, where n

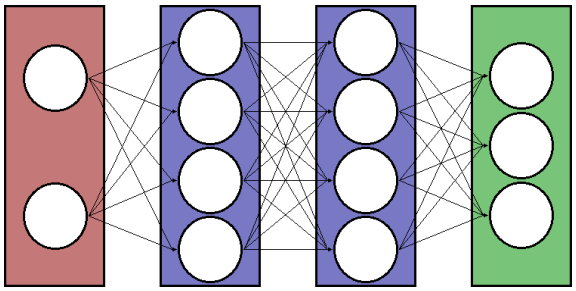


Figure 2.6: Deep Neural Network with four layers (red section is the input layer, blue sections are the hidden layers and green section is the output layer). Each circle represents a neuron

is the number of neurons in the previous layer, $\mathcal{X} = x_1, x_2, ..., x_n$ the value sent by each neuron in the previous layer, $\mathcal{W} = \omega_1, \omega_2, ..., \omega_n$ and $b = b_1, b_2, ..., b_n$ the weights and biases associated with each neuron in the previous layer, that one obtains the intrinsic value of the neuron for a given input instance.

$$f(\mathcal{X}) = \mathcal{W} \times \mathcal{X}^T + b \tag{2.1}$$

Since the expression above represents a linear function, the data undergoes only linear transformations throughout the DNN, preventing the detection of complex patterns as most data cannot be modeled by a linear function. To this end, an activation function is applied to the intrinsic value of the neuron. As its name suggests, this function aims to activate (outputting 1) or deactivate (outputting 0) the neuron. An example of such a function is the sigmoid function, which maps into real values within the interval [0,1]. The value derived by each neuron is forwarded to all neurons in the next layer, giving the name of forward propagation to this movement.

As stated earlier, weights and biases are important. This is because they are the core of improving knowledge representation by being adjusted during training to minimize the error between the final result and the true value (label).

Having covered the smallest particular of DNNs, we expand to the concept of layers. Even though the layers are composed of neurons, they do not necessarily have to be of the same size. The amount of neurons may vary between layers due to its position in the structure, which can be divided in three sections: input, hidden and output layers.

The input layer, corresponding to the first layer of the network, receives the input instances and, traditionally, feeds this data directly to the next layer without transforming it. This layer has two main constraints on the data: all instances must have the same number of features and must be encoded in real or integer numbers. Its number of neurons in depends on the number of features of the instances. For example, remembering the apples illustration, given in the SVM algorithm in Section 2.2.1, the number of neurons in the first layer would be two, and adding other features, such size and height, would increase it to 4 neurons.

All layers between the second and penultimate one refer to the midsection - hid-

den layers. Given the fixed number of input and output layers, the concept of "deep" neural networks derives from the great number of layers found in this section. However, both the number of layers, otherwise referred to as depth of the network, and the number of neurons in each layer, which may vary between layers, is arbitrary. An important note to be aware is that while increasing the depth and number of neurons generally yields good results, such as discovering more abstract patterns, it comes at the cost of increasing the complexity of the network.

The last section of the network is the output layer, which receives the outputs of the last hidden layer and, through an activation function, obtains the value of each class, with each class being represented by a neuron. The final result of the DNN is the class which neuron outputs the highest value. Using Figure 2.6 as an example and assuming that the neuron with the highest value in the last layer is the second from the top, the decision produced by the DNN is class 1, since the classification starts at 0.

Convolution Neural Networks

Having introduced the main concepts, it is now possible to delve deeper into concepts inherent to CNNs. This structure is a type of DNN designed to process data in a grid structure, such as images [26]. However, since images are only different arrangements of real/integer values, the following question arises: what is the need to create a specific network for processing this type of data?

As seen earlier, the value of each feature of an instance is sent to all neurons of the following layer, i.e. each neuron in the first hidden layer receives all the instance information. With an image as input, this process translates into each neuron of the hidden layer receiving all the pixels of the image to find the patterns in it. Yet, patterns in images are in regions so, the DNN should not be required to receive the whole image and evaluate its pixels equally. Thus, it was necessary to create a new method that could analyze an image by regions, leading to CNNs. To understand the functionalities of these neural networks, a 16x16 pixel image (256 pixels in total) will be used.

The analysis in CNN regions consists in connecting each neuron of the first hidden layer to a small region of the input image, which can be of size 5x5 as shown in Figure 2.7. Starting by assigning the first neuron of the region that is in the upper left corner of the image, the window is slid to the right a certain number of columns, usually 1, where this sliding value is given by the stride. When the region reaches the upper right corner, it moves down one or more lines, depending on the stride value, and begins to slide the region again from left to right. This whole process ends when the region reaches the bottom right corner.

Since the region in this example consists of 25 pixels, it is necessary to perform some kind of operation to transform it to just one value. This operation is dependent on the type of layer implemented. CNN offers two different types of layers: convolutional and pooling.

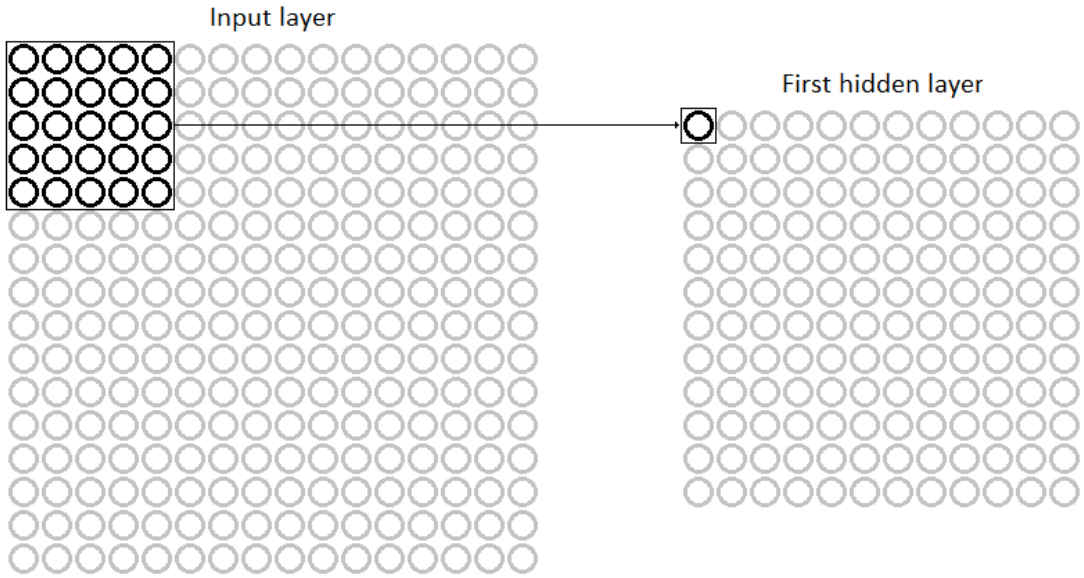


Figure 2.7: Mapping of a 5x5 region from the input layer to a neuron in the following layer

The convolution layer first creates a matrix with random values, the size of which is the same as the region, and then performs the convolution operation on each region obtained from the image. Expression 2.2 represents the convolution operation, where *region* is the region obtained from the image and *kernel* is the randomly generated matrix.

$$convolution = \sum_{n} region_i \times kernel_i \tag{2.2}$$

The result of the layer is a feature map. Usually, several different matrices are used so that it is possible to detect a feature in all regions, as can be seen in image 2.8.

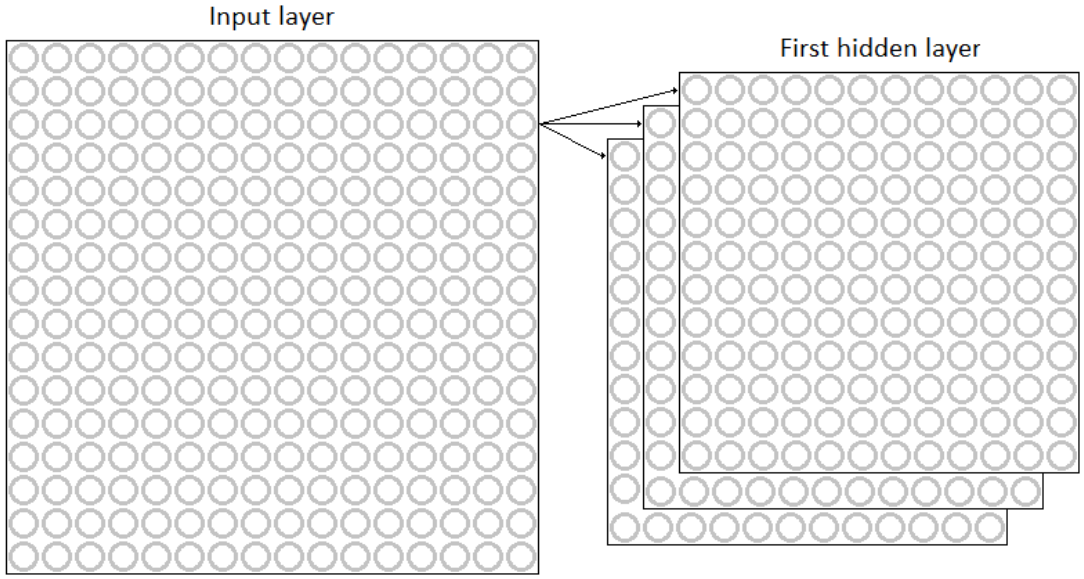


Figure 2.8: Feature maps resulting from 3 different kernels with size 5x5

When training the CNN, the kernels are adjusted to improve the accuracy of the model.

The pooling layer usually follows the convolutional layers in order to summarize each feature map. The operation performed by these layers consists in choosing the highest value of the region (max-pooling) or the average of the region (average-pooling). Note that the region for this layer is smaller than the region used to create the feature map.

An example of a complete CNN is shown in Figure 2.9.

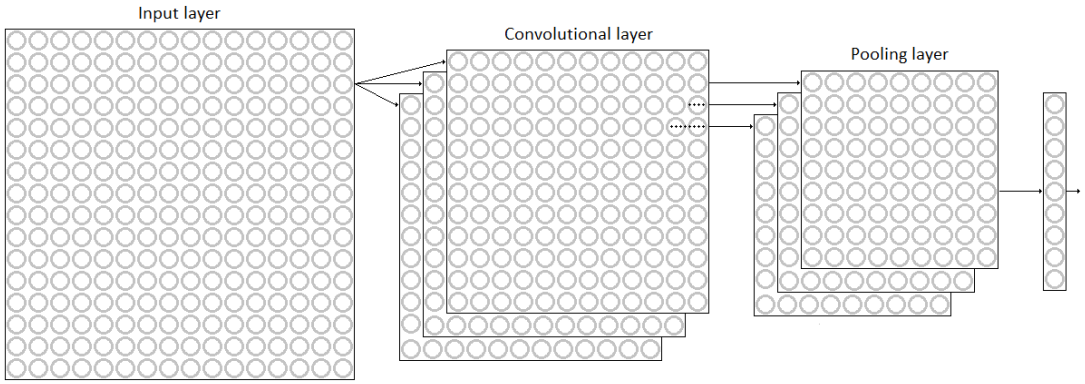


Figure 2.9: Example of a CNN

2.2.2 Unsupervised learning

In contrast to the previous learning method, an unsupervised learning model operates upon data without its labels. Therefore, the model has to discover patterns and relationships between instances of data without any sort of guidance [26]. Since there is an absence of a teacher, the output of this type of learning is not necessarily wrong, is just depends on what we are looking for.

Despite the various techniques used in unsupervised learning, only clustering will be discussed in this subsection since the work described in this report incorporates an algorithm of this nature.

Clustering aims at dividing the data into groups, where similar instances are in the same group and therefore closer together in space, while dissimilar instances are in different groups and further away from each other. As previously stated, this task is subjective, and therefore different forms of reasoning can be applied when building a clustering algorithm. However, due to the clustering algorithm implemented in our work, only density-based models are worth mentioning. These models search for areas of high density of data points and assigns them to the same cluster [27]. By doing so, it isolates the clusters with sparse areas and is even able to find some outliers.

Density-based Spatial Clustering of Applications with Noise

DBSCAN is the clustering algorithm used in this work and, as such, it is necessary to give an introduction to how it works. For this, the same example of apples used in the SVM algorithm in Section 2.2.1 will be employed, but with a larger number of samples.

The process starts by iterating over each instance and counting the number of neighbours. An instance is considered a neighbour of another if it is within a predetermined problem dependent distance ε. Taking the following Figure 2.10 as an example, the red point has 6 neighbours since the red point is neighbour to itself.

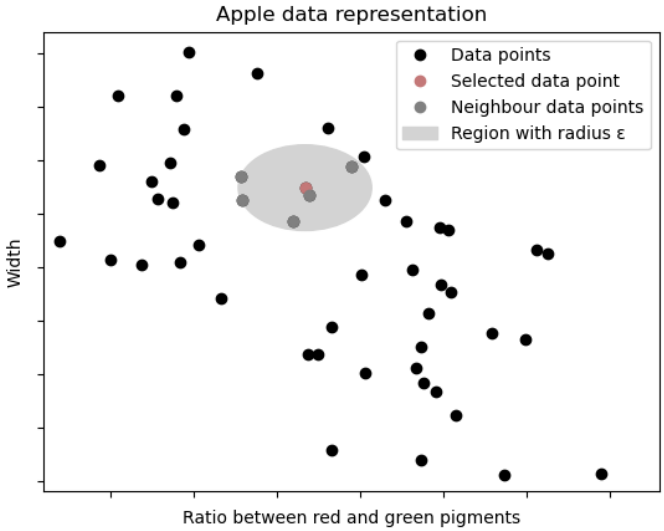


Figure 2.10: Apple data representation for DBSCAN with a selected data point and its neighbours

Then, each instance is marked as a core point in case it has at least a certain number of neighbours. The minimum number of neighbours necessary that makes an instance become a core point is also predetermined and problem dependent.

As soon as all core points are defined, a random core point and all of its core point neighbours are selected and marked as belonging to a cluster. This cluster continues to grow by iteratively adding neighbouring core points. Next, once all core points are added, the neighbouring non-core points are added to the cluster. It must be noted that a non-core point can only join the cluster and not extend further. A complete cluster is shown in Figure 2.11.

This algorithm creates clusters sequentially, i.e. when a non-core point is between two clusters, as exemplified in Figure 2.12, this instance belongs to the first cluster created of those two, and then cannot be considered for any other cluster.

Figure 2.13 is the final result of the DBSCAN for the apple problem, where the green cluster may represent the green apples and the red cluster the red apples.

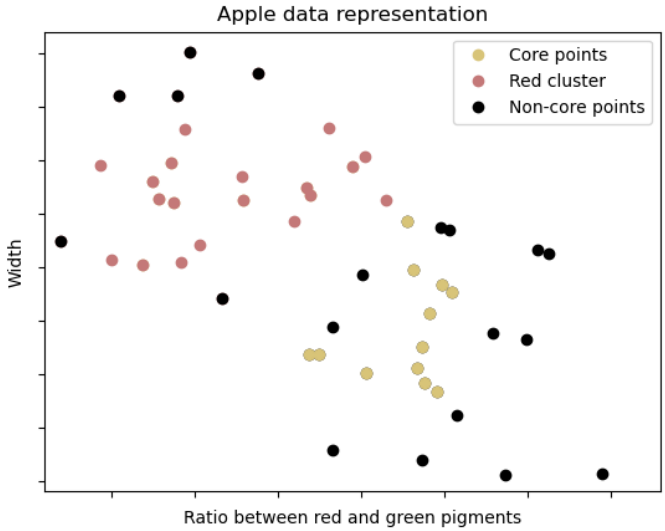


Figure 2.11: Apple data representation for DBSCAN with the cluster for red apples complete

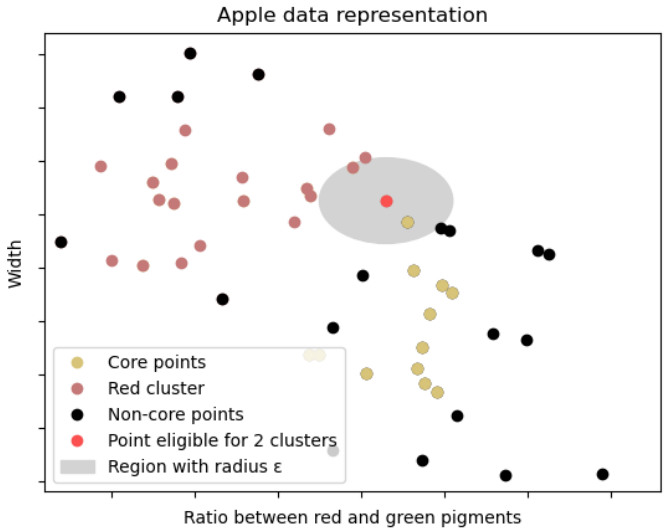


Figure 2.12: Apple data representation for DBSCAN with a point that could belong to two clusters (second cluster yet to be created)

2.2.3 Reinforcement learning

Although reinforcement learning is not used in this work, some researchers use it in an open-world context [28][29], hence being addressed in this chapter.

An agent that learns by reinforcement operates in an environment with a specific set of actions and must discover which one to perform in a certain situation according to a reward function. Just as for unsupervised learning, the agent is not told what to do, meaning there is not a guide that maps environment conditions to actions, but instead the agent has to figure out which action generates the high-

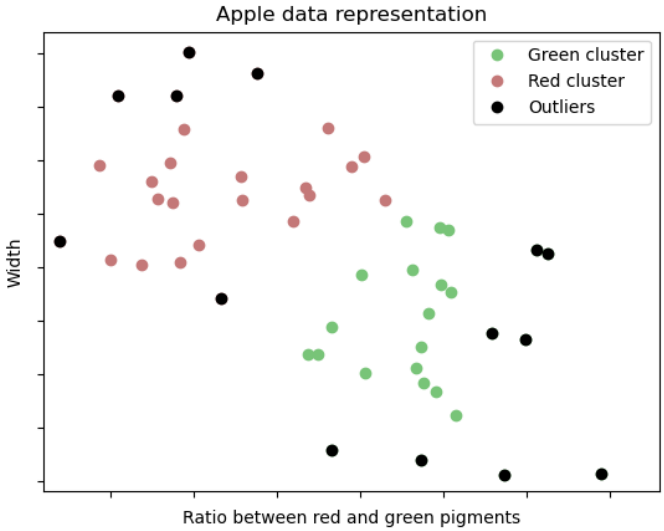


Figure 2.13: Final result of DBSCAN for the apple problem

est reward by trying them [30]. And since the agent is in an environment, it is not limited to just one event, i.e. a fixed data set. Its learning comes from constant interaction with this environment, where its experiences are loops between actions and feedbacks (result from the reward function) [26].

An example of this type of learning is the AWS DeepRacer [31]. This project is about virtually training a car to drive, with only the steering angle and throttle power as actions, and then applying the agent's knowledge to a real car and track.

2.2.4 Online learning

Like the previous learning method, online learning is featured because of multiple articles made on this topic that implement it [32] [33] [34].

Traditionally training an agent consists of using a fixed amount of data to optimize an equation. This can be called offline training since there is no new information coming into the system [35]. However, there are situations where new observations are provided over time or a data set too large to store in memory that needs to be split up and provided to the agent in parts as a stream of data [17].

Online learning is suited for this latter situations.The objective of this type of learning is to minimize regret, which is given by the difference between the achieved outcome and the best achievable result. A simple example of online learning is a shopping website that is constantly receiving and learning real-time user behaviour in order to provide personalised shopping recommendations.

2.2.5 Active learning

Considering again the supervised learning method, this is about collecting substantial amounts of data to train a model to perform predictions. The machine simply receives it with no free will in deciding what to learn, and thus this process is called passive learning. However, there are situations where either there is not much data to begin with or, from the vast collected data, there is only a few labels and increasing it would be expensive.

Active learning algorithms were created to deal with these scenarios. They are able to perform on a low amounts of training data or labels but, to do so, the model has to query a human during the learning process in order to clarify some "doubts" it might find along the way. This learning method seeks to achieve the same or better results than supervised learning with less data [36].

This is related to our work because the data set is comprised of many unlabeled data (treated as unknown) and when these unknown instances are divided into classes, there is a need to analyze and label each created class.

2.3 Open-World Artificial Intelligence

We now introduce the central topic of our work: open-world artificial intelligence.

A famous example that describes an open-world situation well is the Black Swan problem. This example dates back to the beginning of the 17^{th} century. Until then, in Europe, only white swans had been seen and a logical assumption to make at that time would be that all swans are white. Meanwhile, upon discovering and exploring an island, known today as Australia, a seemingly impossible event occurred as a black swan was spotted. With only one observation, the general knowledge collapsed and revealed the fragility of our knowledge.

This can be seen when introducing, for example, supervised learning agents to open-world environments. Since they only know what they have seen during training (white swans), they are unable to understand that a sample belonging to a new never seen class (black swans) is in fact unknown and try match the sample to the known class whose examples are most similar. To add the new class, the whole knowledge has to be deconstructed and a new one has to be built from scratch. This leads to a new field of research: anomaly/novelty/outlier detection [37][38][39][40].

However, these unexpected events can also be seen differently. Despite knowing that these events exist and are bound to happen, they could be treated as if we did not knew. This is the case of the Long Tail problem. It occurs when the data set is not well balanced and the frequency of elements per class decreases substantially.

Considering, the 2 classes *17* and *9* from Figure 2.14, it is possible to see that the difference in samples between the two classes is high. This is detrimental when training a deep neural network model, since it gets biased towards cars

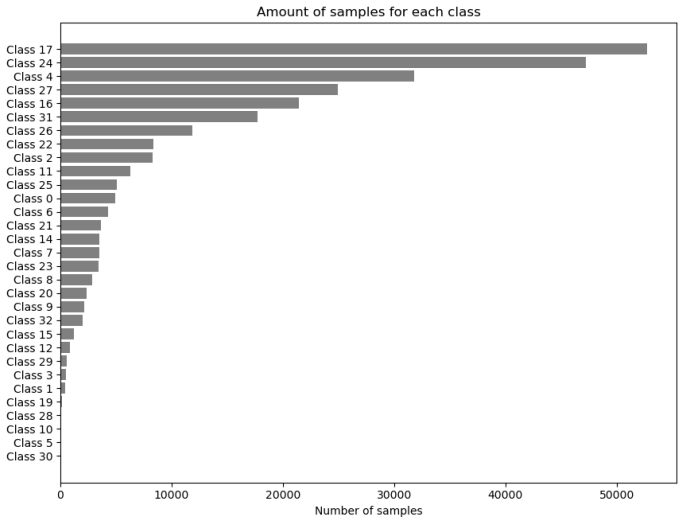


Figure 2.14: Number of samples for each class following a Long Tail distribution

[41]. Note that the problem necessarily stems from missing observations, but rather labeled observations. Often, labeling them is a costly and time-consuming process, especially in a data set with 10,000 or more samples per class, as happens in the current example.

From the many already implemented solutions [42][43][44][45], a classical one is the re-sampling method which is about randomly repeating instances from low sampling level classes and randomly discarding instances from high sampling level classes.

Yet, open-world artificial intelligence differs from the above mentioned situations once this approach learns the unknown samples, adding them to its knowledge. This is a continuous process, similar to active learning.

Chapter 3

Literature review on open-world learning

Numerous methods integrate the open-world component into a model requiring supervised learning, so in this chapter, it will be conducted a comprehensive review of some of the current literature on this topic. Both the problems and solutions are described. Other works related to open-world learning may be found in Parmar and Chouhan [46].

3.1 One versus Set Machines

In the early days of open-world learning exploration, SVM for just one class (1-class SVM) were able, albeit in a rather rudimentary way, to deal with outliers or anomalies in the data. As it is a problem of multiple classes, this approach consists in having n SVMs, where, as suggested by the name, each SVM is trained with only one class, referred to as the positive class. Since there is no negative class to help find the plane, the origin of the space is used as only instance from the negative class. On this basis, its training process involves finding the plane, in a k-dimensional space, being k the number of features of the dataset, that separates the positive class from the negative one. Given that the solution is an SVM, the plane, in a simple solution, must intersect the support vectors, which are the data instances from the positive class closest to the origin of the space, keeping the remaining instances of the positive class in the positive region of the space opposite to the one where the negative instance is included.

However, 1-class SVMs have severe shortcomings in terms of generalization and specialization, as the space is infinite. The reason is that if an instance is located far away from the positive class, as long as it is on the positive side of the plane, it is considered as an instance of a known class (overgeneralization). Since the changes on the plane have to be minor, so that most positive instances are on the right side and not many negative instances are accepted, there is a specialization problem (it can never be specific enough).

In order to minimize this open-space risk, [47] improved the 1-class SVM to what

they called 1-vs-Set Machines (OvS), by adding a new Ω plane parallel to the plane created by the 1-class SVM, refered to as base near plane, limiting the positive class. Hence a serious reduction of the volume is obtained since the space referring to the positive class ceased to be half-space, but only the space between the two margins. It is possible to notice that this improved approach allows managing the generalization and specialization by moving the two planes further (generalize) or closer (specialize) from each other.

Once the far plane is set to include all instances of the positive class, supporting generalisation means distancing the margins, which results in a negligible impact as these instances are still accepted and only the volume of the positive class space increases. Nevertheless, this notion is necessary to tackle the problem of overspecialisation. The overspecialisation comes from trying to decrease the number of false positives. By doing so, it approximates the two straight lines in such a way that only a small percentage of instances of the positive class are accepted. By generalising the first margin (approximating it to the closest negative instance contained in the negative region), it allows the second margin to be brought closer without losing volume. To also mitigate overgeneralisation and overspecialization, a ν value that trades-off accuracy for smoothness is set.

To evaluate the performance of their work, the authors used two data sets, Caltech 256 [48] and Labeled Faces in the Wild (LFW) [49]. For the first data set, the authors established 7.2% of the classes as known, achieving 33.33% of accuracy for the positive classes, correctly labeling 10 instances out of 30, and 79.86% of accuracy for the negative classes, matching in 2300 instances out of 2880. As for the second data set, Scheirer et al. compared their model with one of the best performing algorithms on LFW [50], using ROC curves, attaining the results in Figure 3.1.

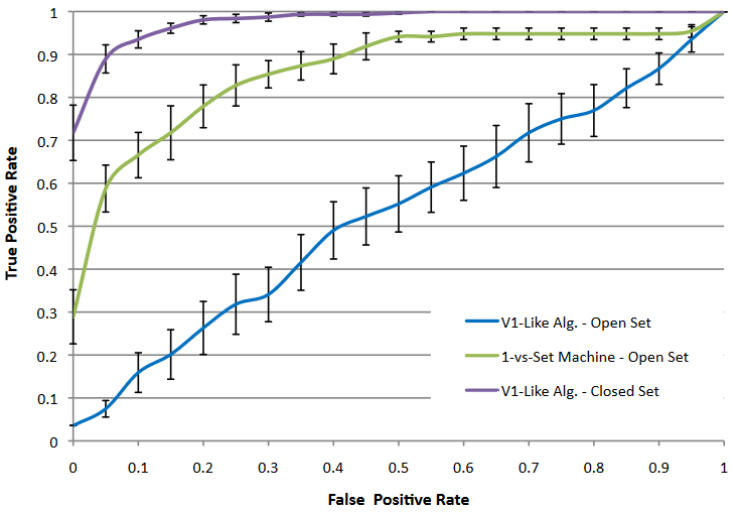


Figure 3.1: ROC curves comparing the accuracy of the 1-vs-Set Machine approach to a multiclass SVM approach using V1-like features [50]. Each point represents mean accuracy and the error bars reflect standard error

As the multi-class SVM approach using V1-like features is one of the best for this specific data set, its closed-world nature (purple line in Figure 3.1) outperforms the OvS. Meanwhile, its open-world aspect falls significantly short.

3.2 Weibull-calibrated SVM

Despite OvS machines being able to significantly reduce the space of the positive class, the space is still infinite between the two parallel planes. To overcome this issue, the same authors developed a Compacting Abating Probability Model (CAP) [4] which is capable of isolating the positive class in a finite manner and ensure the probability that a test data instance falls into a certain class decreases the further away it is from the sample of this class.

The CAP model is composed of two parts. The first part consists of a finite abating bound. According to the authors, an abating bound is "a non-negative finite square integrable continuous decreasing function", where its decreasing characteristic allows the value of the function to be lower the further two points are from each other, these being a test instance and a class training sample. As this function is used to isolate the positive class, it is used as the kernel function of a SVM. The second part is the calibration of the kernel in order to obtain probabilities. To this end, the authors use the training data to define a "monotonically decreasing probability distribution", such as an exponential or Weibull distribution, and use the probabilistic distribution on the outputs of kernel. Meanwhile, most of the probabilities are non-zero even though they are very small. By applying a minimum threshold to accept a test instance, this issue is dissolved, where rising it lowers the open-space risk.

However, the probability achieved by applying the CAP model only takes into account the probability of belonging to an unknown class. To overcome this problem, Scheirer et al. implemented a complementary method only when the instance is accepted as a known class, i.e. when the probability of a given instance x_i belonging to a class c is higher than the threshold established in the CAP model. The complementary method is a binary SVM, from which are attained both the probability of x_i belonging to the known class c (P^+) and the probability of not belonging to that class (P^-). These probabilities are transformed into scores using two Weibull cumulative distribution functions, one for each type of probability (P^+ and P^-), in the attempt to minimize the impact of the overall distribution of the data on the final result. Thus, two different sets of values are obtained, which are P_η, representing the scores of being accepted as instances of a known classes, and P_ψ, the scores of being unknown instances.

The authors complete their method with the application of the equation 3.1, which retrieves the most likely known class for the instance x_i.

$$c = \arg\max_{c \in \mathcal{C}} P_{\eta,c}(x_i) \times P_{\psi,c}(x_i) \tag{3.1}$$

In order to prove that their new approach is better than their previous OvS [47], they tested with four data sets, such as Caltech-265 [48], ImageNet [51], LETTER [52], MNIST [53], and compared the F-scores between other open-world approaches and multi-class algorithms. Scheirer et al. took a step further in their evaluation of the different approaches by comparing their performance in worlds widely and poorly opened. The expression 3.2 describes the openness of

the world.

$$openess = \sqrt{2 \times T_c / (R_c + E_c)} \tag{3.2}$$

For the Caltech-265 and ImageNet data sets, the Weibull-calibrated SVM (W-SVM) has shown a 20% to 26% improvement in F-measure, in relation to the previous OvS approach. For the remaining LETTER and MNIST data sets, it is clear in Figure 3.2 that W-SVM outperforms the others in unknown instances detection.

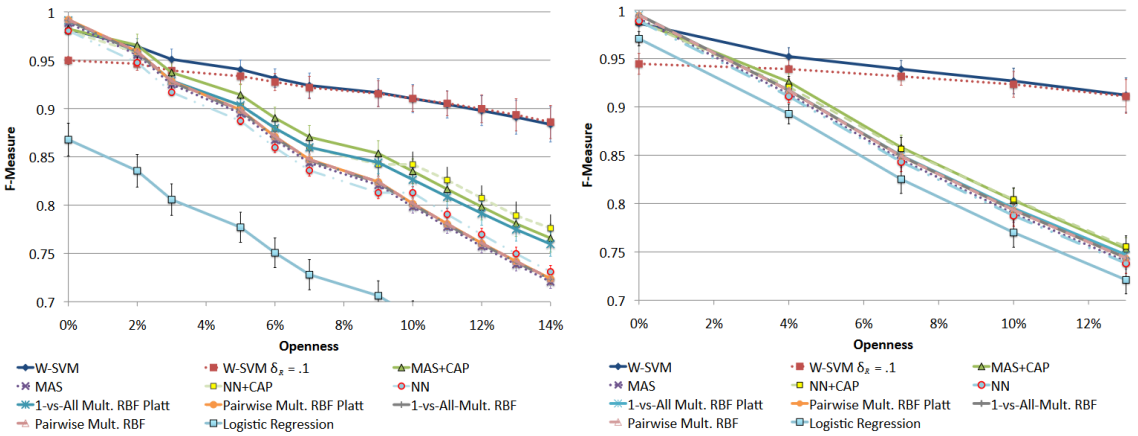


Figure 3.2: F-scores of the several approaches tested for LETTER (left plot) and MNIST (right plot) [4]

3.3 OpenMax

DNN are not prepared to embrace the open-world AI since they require the labels of the training data set. Even though a solution such as thresholding the values of its last layer is possible, some researchers showed that certain images are able to mislead [54] [55]. With this in mind, Bendale and Boult proposed a new method to be appended as the last layer of a DNN, obtaining the class of an instance in an open-world scenario called OpenMax [9].

In the first stage, the DNN is trained normally. Then, the training images are sent to DNN again, but this time for classification, and the output of the penultimate layer, to which the authors gave the name activation vector, is used to calculate the mean of each class for each neuron. Next, the test data set is used in the DNN where the OpenMax method is applied.

The new method starts by receiving the activation vector and calculating its distance to the mean of each class. As there is no need to compare all classes, since only the closest classes are the most likely to correspond to a given instance, the authors decided to limit the comparisons. The cumulative Weibull distribution function is then applied to the top classes of an instance, and the new class "unknown" is added, which corresponds to the remaining of the subtraction between

one and the sum of the scores of the top classes. Finally, test instances are declared as unknown whenever the highest scoring class is the "unknown" class or the highest scoring class is known but its score does not meet a certain threshold.

To verify that their new proposed method enables open-world AI in DNN, the authors compared the performance of OpenMax against a SoftMax function with threshold, so it is able to detect unknown instances, and a Softmax without thresholds. All these three layers were appended to the AlexNet DNN provided by the Caffe software package [56]. The data set used for the tests was ImageNet Large Scale Visual Recognition Competition 2012 (ILSVRC) [57].

As it is illustrated in the following Figure 3.3, the OpenMax outperforms the Softmax with threshold method, with an improvement in accuracy of 4.3%. OpenMax also revealed an improvement of 12.3% over a SoftMax without threshold.

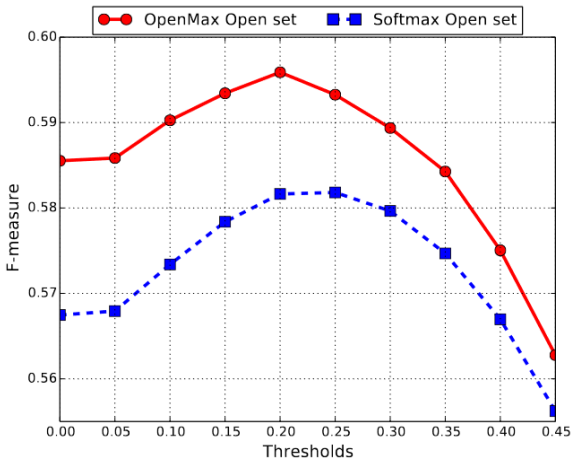


Figure 3.3: F-scores of OpenMax and SoftMax with threshold on ILSVRC 2012 data set [9]

3.4 Open-set Nearest Neighbor

Open-Set Nearest Neighbor (OSNN) [5] is an extension of the existing closed-world Nearest Neighbor classifier, adapted to identify unknown class instances from test samples. Instead of calculating the similarity scores for all classes and applying a threshold on the highest similarity score (score of the most similar class), as the traditional method does, this new method gets the two closest neighbors, through the Euclidean distance measure, from distinct classes, calculates the ratio of similarity between them and applies a threshold on it. This new approach is characterized by the disregard for the "open" scenario, for the regions of space where unknown classes may appear as well as for the number of unknown classes that may appear in the test samples. It also has the ability to create a boundary around the positive region, limiting the open space risk. This ability is what allows the method to find and reject unknown classes.

Unlike the previous open-world approaches described, OSNN is inherently a multi-class solution, meaning that while other solutions have intersections be-

tween the positive and the negative classes, where the more classes the problem has, the worse they perform, the OSNN is not affected by it, needing at least two known classes.

The authors built two different methods of the OSNN approach. These are the Class Verification (CV) and Nearest Neighbor Distance Ratio (NNDR) and, even though they differ in the methodoly, the major difference is in the ability to find unknowns, where the first one is not able to declare a test instance far from the two closest training instances as unknown, while the latter succeeds.

The first method, CV, is rather simple. It selects the two nearest neighbors of the test instance, and if they have the same class, that class is assigned to the test instance, otherwise the test instance is declared as unknown and discarded.

The second NNDR method is more elaborated with a problem-dependent variable. This one also selects the two closest neighbors but these have to belong to distinct classes. Being s the test sample, and t and u its neighbors (where t will always be closer to s, or at the same distance, than u), the NNDR obtains the distance between the test instance and each of the neighbors and calculates the distance ratio according to the following expression:

$$R = \frac{d(s,t)}{d(s,u)} \tag{3.3}$$

From this, the label of the test sample can only be the same as the label of the neighbor t, or be unknown. The decision is made by a positive threshold value less than 1. If R is smaller than the threshold, then the test sample belongs to the same class as t, otherwise its class unknown.

The threshold is the boundary characteristic, being the problem-dependent variable due to its decision component. As such, the authors developed a mechanism to find the best threshold automatically, avoiding its manual discovery, keeping the solution simple. To accomplish this, from the data that would be for training T, they divide the number of known classes into two. One part remains known and is split into two, where the first half is the reduced training data F and the second half is inserted into the validation data V. The remaining part becomes unknown and is also inserted into V. With F and V defined, the threshold is calculated in the following way. First, they assign the interval from 0.5 to 1.0 to the threshold, partitioning it into ten equal parts. Then, all threshold values are tested against V and the threshold i, where i is the i-th threshold from the interval, that achieves the highest accuracy is selected. As the threshold can be more specific, they take the thresholds $(i-1)$-th and $(i+1)$-th and calculate the following average values.

$$\begin{aligned} mean_A &= \frac{threshold_{i-1} + threshold_i}{2} \\ mean_B &= \frac{threshold_{i+1} + threshold_i}{2} \end{aligned} \tag{3.4}$$

The two values $mean_A$ and $mean_B$ are taken as the extremes of the new threshold interval and the whole procedure is repeated. This cyclical process is performed four times and the final threshold is set as the best one for the given problem.

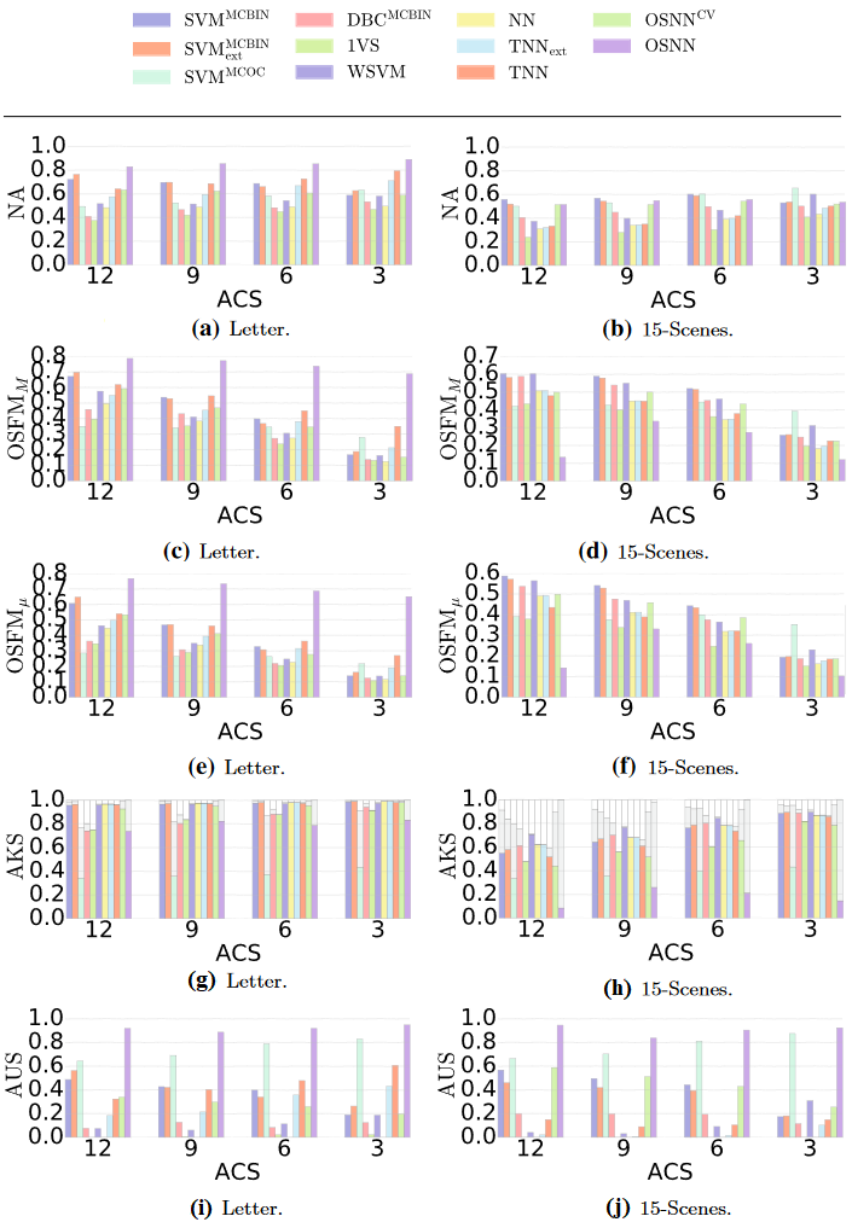


Figure 3.4: Results of the several approaches tested for one of the best data sets of OSNN (LETTER) and one of the worst data sets of OSNN (15-Scenes) on all measure methods [5]

To identify the potential of OSNN, six data sets were used such as 15-Scenes [58], LETTER [52], Auslan [59], Caltech-256 [48], ALOI [60] and Ukbench [61]. Its performance was compared with other works, including the OvS and W-SVM. The authors took into consideration different measures, for instance, the normalized accuracy (NA), the macro ($OSFM_M$) and micro-averaging ($OSFM_\mu$) open-set f-measures, and the accuracies of the known (AKS) and unknown samples (AUS).

For all the measure methods, excluding the accuracy of the known samples, OSNN outperformed all the other approaches except for the 15-Scenes and Caltech-256.

If a more detailed analysis is required, the full results can be found in [5].

3.5 Neural-Network-Based Representations

Hassen and Chan [6] try a different approach on the open-world problem. Instead of trying to develop a model or a set of them, they go for data representation, transforming it in order to find outliers in an easier way. To this end, they use a DNN to learn the training data representation and adjust it. The data representation must have two properties. The first property is to have all instances of a class close together and the second property is to have different classes far apart from each other. Converting the training data to respect these properties allows you to isolate a class by having all its instances close to the mean/center, and to have the means of two different classes as far apart as possible.

The learning and transformation of the data is done in the layers of the neural network. The layers can be either convolutional or fully connected ending with a loss function, named by the authors as "ii_loss", given by the following expression.

$$ii\_loss = intra\_spread - inter\_separation \tag{3.5}$$

By minimizing this loss function, the intra-class distance is minimized while the inter-class distance is maximized.

With all this set, even though the aim of the work is to have a low classification error, it is not stated in the properties of the solution. For this, the authors set a third property declaring a low classification error in training. To this end, they combined the "ii_loss" function with a cross entropy loss function and trained with the intention to minimize both loss functions.

As the models finish the train, the model is stored in order to transform the test data. The averages of each of the known classes are also stored since these are be used to detect outliers.

Their solution classifies the test instances as one of the known classes if it is sufficiently near the class mean, or as unknown if it too far from any of the known classes means. To be declared as an unknown, there are two steps needed. First, an outlier score is calculated, which corresponds to the distance of the test instance to the center of the closest class. Next, a threshold is applied to check whether the instance should be accepted or not. The threshold is, like many other solutions, problem dependent. The authors train the models as if 1% of the training data is outliers and so, the minimum distance is to closest considered outlier is set as the threshold. This is set global to all the classes instead of being class dependent since, according to the authors, it holds the best results.

In order to test their approach, the authors compared their multiple versions with the state-of-art OpenMax [9]. To this end, Hassen and Chan used three data sets, such as the Microsoft Malware Challenge data set [62], the Android Genome

Project data set [63] and MNIST data set [53]. As measure methods, they used the precision, recall and F-score.

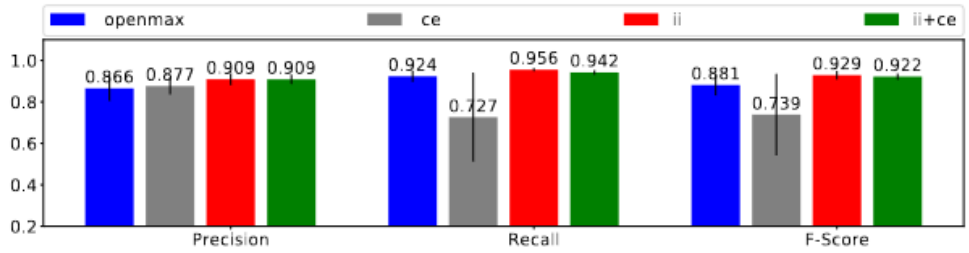


Figure 3.5: Results of the several approaches tested for the MNIST data set (*ce* is the use of the cross entropy only, *ii* is the use of the ii loss function only, and *ii+ce* is the use of both) [6]

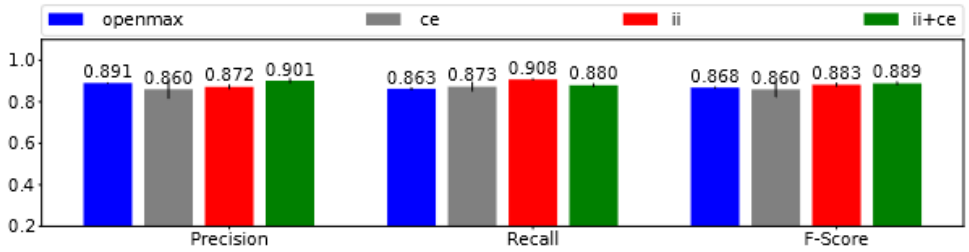


Figure 3.6: Results of the several approaches tested for the Microsoft Malware Challenge data set (ce is the use of the cross entropy only, ii is the use of the ii loss function only, and ii+ce is the use of both) [6]

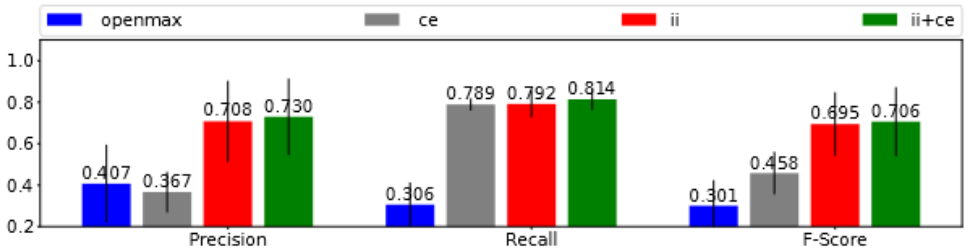


Figure 3.7: Results of the several approaches tested for the Android Genome Project data set (ce is the use of the cross entropy only, ii is the use of the ii loss function only, and ii+ce is the use of both) [6]

As it can be seen by the Figures 3.5, 3.6 and 3.7, the neural-network-based representations allow better results in all data sets, particularly in the Android Genome Project data set. Even though the variant that combines the ii loss function with cross entropy is not always the best one, it outperforms OpenMax in every situation.

3.6 Unseen Class Distinction

Diverting from the search for unknown class instances in the test data, some researchers set out to analyze all the rejected instances, the ones that do not belong

to any known class, and find how many unknown distinct classes are there [10]. To do so, they divided the problem in four parts.

The first part is the acceptance and rejection of test instances. For this, the authors build an Open Classification Network (OCN) which consists of a CNN followed by a fully-connected layer and a 1-vs-rest layer of sigmoid activation functions. The latter layer is able to reject instances by applying a threshold on its output, where an instance that does not have a probability output value higher than 50% is rejected.

The second part is the comparison of the classes of two test instances. Here, the authors focused on training a model, Pairwise Classification Network (PCN), that takes two data instances and learns their intra-class or inter-class relation. The model is equivalent to the one used to reject instances, where each instance requires a CNN and a fully-connected layer. The outputs of the two-fully connected layers are used to predict whether the instances are from the same class or not, based on the distance between them, through one last activation sigmoid function.

As training both the OCN and PCN with nothing but data from known classes is highly prone to overfit, their third part consists of an auto-encoder trained with some unlabeled data instances. This is trained simultaneously with the two other previous models where the aim of the training is to decrease the joint loss of the three parts described.

The last part is the main objective of their work. For this fourth part, the authors use an hierarchical clustering, where, from the rejected data instances, it is used the distances obtained from the PCN to form the clusters. The hierarchical clustering used is agglomerative, i.e., it starts with several different clusters and it will merge the closest pair iteratively until one big cluster of clusters is formed. The distance between clusters is the maximum distance between two instances of each cluster. On this basis, a dendrogram is created and the number of cluster is obtained through it. Since some clusters may originate from mislabelling of OCN, the authors assigned a threshold, where only the clusters with a distance lower than the threshold are recognized as unknown classes. The threshold is problem dependent and so, Shu et al. added an extra validation step, which consists of four classes that are fed to the hierarchical cluster and the threshold value that holds four final unknown classes is set to be the best value to that given problem.

Towards the end of their work, the authors Shu et al. assessed the capabilities of their model. This was organised in two parts. Firstly, they evaluated the rejection capability of unknown instances by comparing it to the OpenMax [9] approach with different Weibull tail sizes (20 and 1000), through the values of precision and recall, macro F-score of all classes, and F-score of the rejection class. These approaches were applied on the MNIST [53] and EMNIST [64] data sets. As it can be seen in Tables 3.1 and 3.2, the OCN has a better performance on detecting unknown instances on both data sets.

Secondly, they assessed the ability to find the number of unknown classes among the rejected instances. By using the same data sets as for the first assessment, the results obtained are presented in the following Table 3.3, where it is shown that

Algorithms	$(m+1)$ classes	Rejection class		
	Macro F-score	Precision	Recall	F-score
OCN	0.914	0.920	0.824	0.869
OpenMax (w = 20)	0.678	0.955	0.026	0.051
OpenMax (w = 1000)	0.684	0.956	0.043	0.083

Table 3.1: Results of the OCN and OpenMax for MNIST data set

Algorithms	$(m+1)$ classes	Rejection class		
	Macro F-score	Precision	Recall	F-score
OCN	0.832	0.664	0.47	0.554
OpenMax (w = 20)	0.789	0.786	0.07	0.13
OpenMax (w = 1000)	0.803	0.725	0.239	0.359

Table 3.2: Results of the OCN and OpenMax for EMNIST data set

the number of classes/clusters found by the OCN is close to the ground truth number of unknown classes.

Algorithms	Ground truth	OCN	
	# of Clusters	# of Clusters	NMI
MNIST	4	6	0.320
EMNIST	10	14	0.500

Table 3.3: Results of the clustering of the rejected instances (# of C is the number of clusters and NMI is the Normalized Mutual Information)

3.7 Summary

After an extensive review of the various works on open-world AI, we summarize them in a table that provides for each of the works the data sets used, the goal of each one (tasks), the main methods used and some notes. This is all compressed in Table 3.4.

Papers Reviewed	Datasets	Tasks	Methods	Remarks
[47]	[48], [49]	Rejection of instances belonging to unseen classes	SVM	First model adapted to open-world AI; Parameters and open space risk not optimized (infinite)
[4]	[48], [51], [52], [53]	Rejection of instances beloning to unseen classes	SVM	Optimizes the Open Space risk from [47]
[9]	[57]	Detect instances belonging to unseen classes	DNN Layer	Replacing the last DNN layer (e.g. softmax) by one adapted to open-world recognition
[5]	[48], [52], [58], [59], [60], [61]	Detect instances belonging to unseen classes	Adapted Nearest Neighbor Classifier	Efficient performance of a Nearest Neighbor in open-world AI
[6]	[53], [62], [63]	Detect instances beloning to unseen classes	DNN	Solution focused on data representation (learning and adjusting)
[10]	[53], [64]	Unseen class distinction	CNN	Introduces distinction in rejected instances. Rejection method lacks on rejecting instances (low performance)

Table 3.4: Summary of the different papers reviewed on open-word AI

Chapter 4

Methods and materials

Before tackling the problem there are some points to be defined. In this chapter we lay out all the aspects and conditions to start implementing the solution. Therefore, we describe, in a first moment, the programming environment in which the work is performed, followed by the data set used and its treatment and ending with the model architecture.

4.1 Programming Environment

Firstly, we decide the conditions under which the work is carried out, from programming language to additional libraries.

4.1.1 Programming Language

One important step before developing the solution is the choice of the programming language where the work is developed, posing the following question: What is the best programming language for machine learning?

Despite being an easy question to make, it is not that simple to answer. There are many frameworks and libraries across all multiple languages, such as Python, C/C++, Java, among others, making it hard to single out any, and depending on what we want to build or what tools may already be employed, some languages are more favourable than others.

The choice could also be made based upon popularity [65], however it does not represent well since it does not takes into account professional background and, as stated previously, the aim of the project. Nevertheless, it can help choosing the programming language.

As this work is mostly being developed from scratch, there is not a restriction nor a promotion of any available language. However, the language should have a great and active support community of users [66] [67] [68]. It should also high-level as these are easier to use. For these reasons, the chosen language was Python

[69].

4.1.2 Machine Learning Framework

Once we have chosen the programming language, a more specific component has to be decided, which is the ML structure. This framework is a set of tools and libraries that allows us to quickly and easily build a machine learning model without having to fully understand its workings. It also facilitates the manipulation of already implemented models due to its low restrictions.

Following the same idea of the programming language, we opt for an high-level neural network framework. Tensorflow [70], and more specifically its high-level neural network API Keras [71], meets the criteria and is the second most used in academia [72]. Another possible framework would be PyTorch [73] since it is the most used in the academia.

4.1.3 Relevant Libraries

Some additional libraries were necessary to develop this work. One crucial library is NumPy [74], essential for manipulating arrays of multiple dimensions and for enabling the data to be used in neural networks. A second vital library is libMR [75], which provides MetaRecognition and Weibull fitting functionalities. Another library is Matplotlib [76], responsible for creating graphical representations of the data both used and obtained. SciPy [77] is another library used, which allows to compute different types of distances between two instances of data. Finally OpenCV [78], used for image processing, either for reading, cutting or re-shapping images.

4.2 Data set

Cityscapes [79] is a large-scale data set made for approaches for pixel and instance-level semantic labeling, providing a visual understanding of the urban scenes. The data set consists of images from the front viewpoint of the car (Figure 4.1), taken from a moving vehicle, mostly in areas with a lot of traffic, over several months in 50 cities in Germany, mainly, and neighbouring countries, during spring, summer, and fall.

For this data set, two different types of annotation methods were applied to the 2048x1024 dimension images: instance and pixel-level annotations. The first annotation type focuses on making a distinction between elements of the same class while the second is not concerned with this. As one of the objectives of this work is to detect people, cars, and other things, instead of determining who exactly is crossing the road or the model of the car in front, the pixel-level annotation was the one selected to develop this work. The pixel-level annotation data set comprises 5000 images and consists of key points around the marked objects, forming

Figure 4.1: Example of an image from the data set

polygons as it can be seen in Figure 4.2.

Figure 4.2: Example of a pixel-level annotated image from the data set

There are, however, background objects that are in the middle of others, such as buildings in between the tree leaves, or objects that are seen through transparent elements, as people behind car windows. In order to avoid holes in the closest ones, everything is considered as the single closest element. That is, in the first example, part of the building is considered a tree while in the second example part of the building is considered a car.

As it can be seen in Figure 4.2, various colors are available, where each color is associated to a class, obtained from another file. The auxiliary file contains the coordinates of the key pixels and the class of the respective object. Divided into eight categories, the data set contains 30 distinct classes illustrated by the Table 4.1 followed by the classes distribution Figure 4.3.

In this field of research, there are other data sets such as *KITTI Vision Benchmark Suite* [80], *CamVid* [81] and *Daimler Urban Segmentation* (DUS) [82]. When comparing the KITTI and Cityscapes data sets, the first exhibits more nature, less flat elements, and fewer humans due to Cityscapes exploring more inner-city traffic. *CamVid* has a large number of pixels of the sky, being this the most uninteresting class for self-driving cars, and DUS does not have any information about nature or objects categories. For these reasons, we chose the Cityscape data set.

Categories	Classes				
Flat	Road	Sidewalk	Parking	Rail track	
Construction	Building	Wall	Fence	Guard rail	Bridge
	Tunnel				
Nature	Vegetation	Terrain			
Vehicle	Car	Truck	Bus	On rails	Motorcycle
	Bicycle	Caravan	Trailer		
Sky	Sky				
Object	Pole	Pole group	Traffic sign	Traffic light	
Human	Person	Rider			
Void	Ground	Dynamic	Static		

Table 4.1: Categories and classes of the dataset

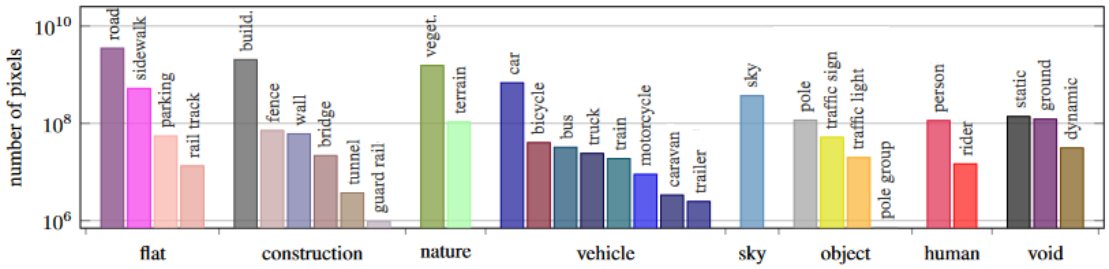


Figure 4.3: Number of finely annotated pixels per class and their associated categories

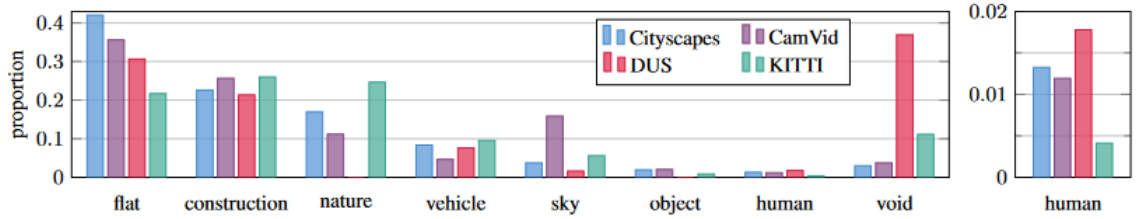


Figure 4.4: Proportion of finely annotated pixels per category for Cityscapes, KITTI, *CamVid* and DUS

4.3 Data Set Treatment

As one of the points of this work is to classify all the objects found in the street by a self-driving agent, being these objects known or unknown, each object has to be extracted from the images to either train or test the agent. The auxiliary file that contains the pixel coordinates of each object for a specific image mentioned in the previous section plays an important role in the extraction of the objects.

The coordinate system of the auxiliary file corresponds to the Cartesian plane of two dimensions on the image, whose origin is the upper left corner, the x-axis refers to the image width and the y-axis refers to the image height. Taking this into account, all the coordinates corresponding to a certain object were searched to find those that correspond to the smallest or largest value of either x or y. The

resulting four values define the corners of the rectangle that frames the object. Let us consider Figure 4.5 as representing the driver's point of view.

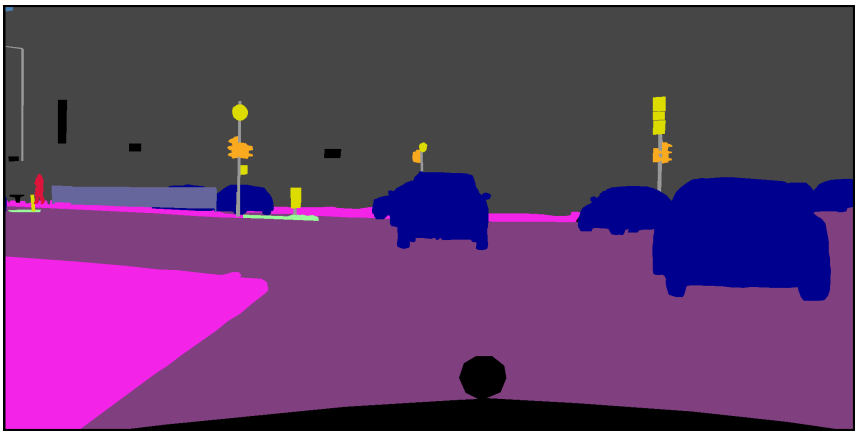


Figure 4.5: Point of view of the driver

In this image, only one person, marked red near the left margin, is in the driver's field of view. From the several coordinates that constrain the image object, the values of x are within the interval [74, 100] while the values of y are between [406, 479]. This produces the white rectangle shown in the following Figure 4.6.

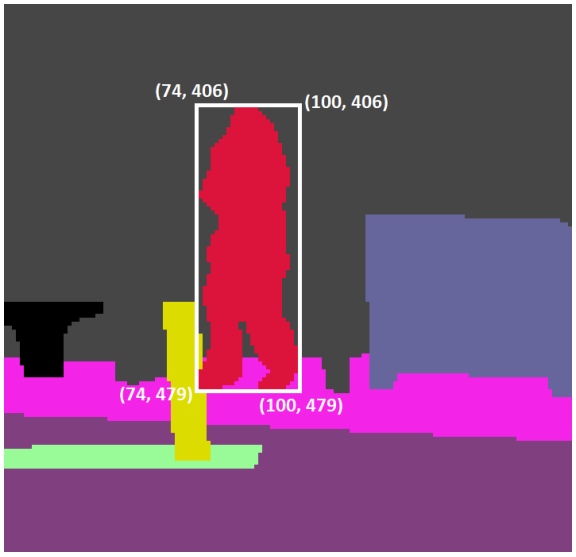


Figure 4.6: Person framed in a white rectangle

However, with just the information inside the rectangle in question, it is difficult to understand what the object in question is, should one not know the meaning of the colour. To this end, the rectangle was expanded on all sides by a value of 10 pixels, as illustrated in Figure 4.7. This increment makes a small object more noticeable due to the clear shape of the object. Large objects, on the other hand, are not affected due to their specific size and shape.

After cropping the object, since convolutional neural networks only accepts images of the same size, all images of objects have to be resized to a size that has to meet two requirements. The first requirement is that it has to be small due to the time availability to train the neural network. Therefore, we considered three

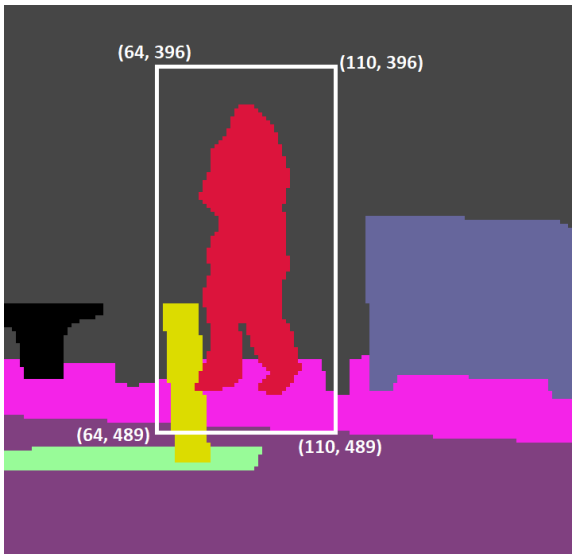
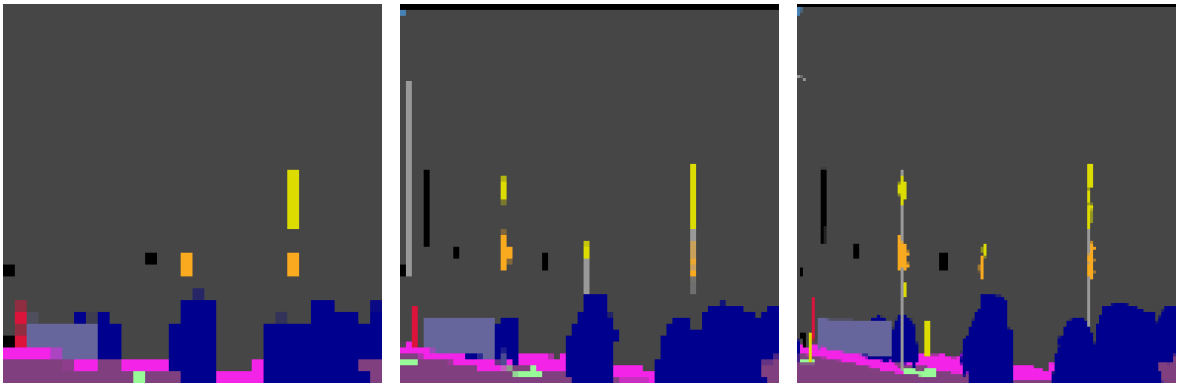


Figure 4.7: Person framed in a white rectangle with a margin of 10 pixels

small dimensions: 32x32, 64x64 and 128x128 pixels. The second requirement attempts to answer the question "How much information are we willing to lose?". Most of the time, changing the object dimension does not deform or cut the object. However, when it comes to objects whose size will be considerably reduced, the final dimensions play an important role on what is observable, beyond the intended object. In order to help us understand this, let us take as an example the building in Figure 4.5. This building extends the entire length of the image with various elements in front of it, from traffic signs to cars and people and its final size, as it is evidenced in Figure 4.8, can partially or totally destroy the remaining objects, as is the case of the traffic light and signs on the left side of the 32x32 pixel resolution image or in the middle of the 64x64 pixel resolution image.



(a) (b) (c)

Figure 4.8: Results of applying different resolutions: (a) 32x32 pixels; (b) 64x64 pixels; (c) 128x128 pixels;

To successfully train a model capable of detecting objects with noise (other overlapping objects), it is necessary to include it in several locations of an object, i.e. lose as little information as possible. For this reason, and according to the previous picture, the size set for all of images of objects is 128x128 pixels.

Once all objects have been cropped and resized, the distribution per class is given by Figure 4.9.

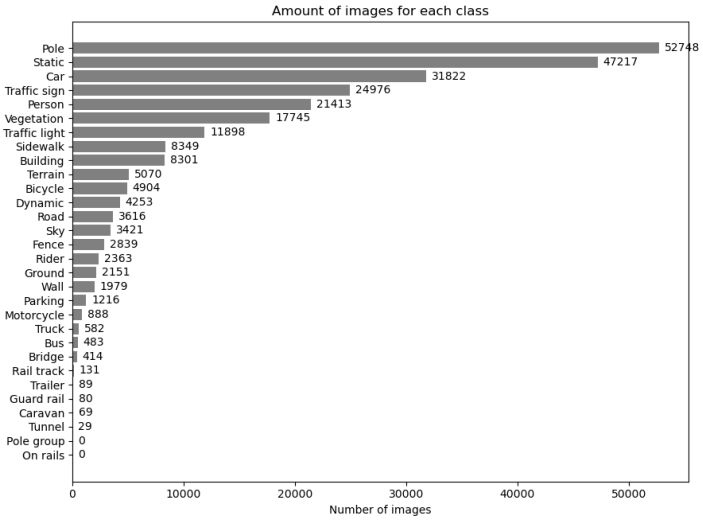


Figure 4.9: Number of images for each object/class

4.4 Model Architectures

In this section, we introduce the model architecture used as a solution for this work. Since the data set is composed of images, we need to firstly build a CNN to be able to classify them, hence addressing the CNNs architectures in the first place. Next, to identify instances belonging to unknown classes, we use the OpenMax method, presented in Section 3, due to its application in multiple approaches on open-world scenarios [10; 83] and the implementation support being in Python. Therefore, we also address this method. As the last stage, we use the algorithm DBSCAN to find as many hidden classes as possible within the instances identified as belonging to an unknown class by OpenMax. However, as it has already been introduced in Section 2.2.2, and only different values of ϵ and minimum number of neighbors needed to turn a instance into a core point will be tuned when applying this algorithm, this algorithm is not mentioned in this section.

4.4.1 Convolutional Neural Networks Architectures

Building a CNN is a challenging task since, despite general rules or ideas, there are no restrictions on the architecture and it is problem dependent. Usually the data set is referred to as the problem that CNN tries to solve. However, in this case, in addition to the data set, the problem is also the ability to find instances belonging to unknown classes while maintaining a good classification in known classes (performed by the OpenMax model described in the next subsection).

Therefore, we developed and tunned four CNNs to get a wider range of results and use the one with the highest accuracy.

As it is expected, since they are included in the same problem, the input layer size of all CNNs is 128x128x3, due to the data set being comprised of 128x128 pixel images with three channels for colors, and their output layer size is 4, whose reason for this value is given in the next section of this chapter.

It should be noted that all convolutional and max-pooling layers in each CNN model have one parameter in common, which is the stride value for each layer. The value set for it is 1 as we want to cover as many areas of the image as possible. In addition, all convolutional and fully-connected layers apply the same activation function, this being the "ReLu" function, except for the last fully-connected layers which applies the "Softmax" function.

The first CNN built, represented in the Figure 4.10, starts with a convolutional layer with 64 kernels, where each kernel has size 5x5, followed by a max-pooling layer with a kernel size of 4x4 and a dropout layer with probability of 20%. On top of this is applied yet another convolutional layer with the same number and size of kernels as the first convolutional layer, and a max-pooling layer with a 4x4 kernel size. The structure ends with two fully-connected layers intertwined with a dropout layer with a rate of 20%.

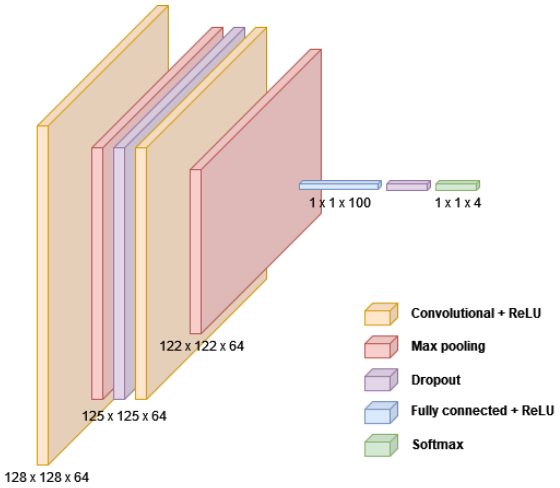


Figure 4.10: First CNN architecture

Figure 4.11 illustrates the architecture of the second CNN implemented. This has a convolutional layer with 32 3x3 kernels, followed by max-pooling layer with a kernel size of 2x2. A convolutional layer with 32 5x5 kernels follows it with another 2x2 kernel size max-pooling layer. One last 64 5x5 kernels convolution layer is applied before the final string of layers. This string is composed of two fully-connected layers interleaved with two dropout layers, where the first dropout layer has a 10% rate and the second has a 5% rate, ending with another two fully-connected layers.

As displayed in Figure 4.12, the third structure starts with a convolutional layer with 32 3x3 kernels followed by a 4x4 kernel size max-pooling layer and a dropout layer with a 20% rate. The structure continues with a 64 5x5 kernels convolutional

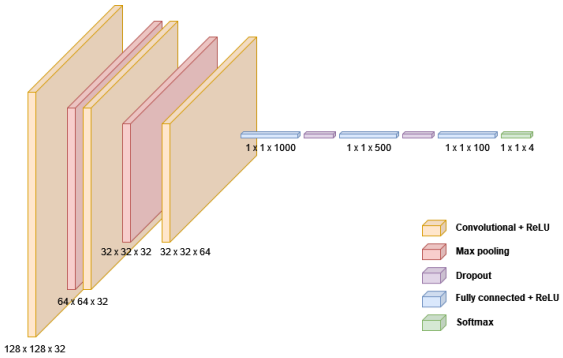


Figure 4.11: Second CNN architecture

layer and a max-pooling layer with a 4x4 kernel, ending with a fully connected layer followed by a dropout layer with 20% rate and another fully connected layer.

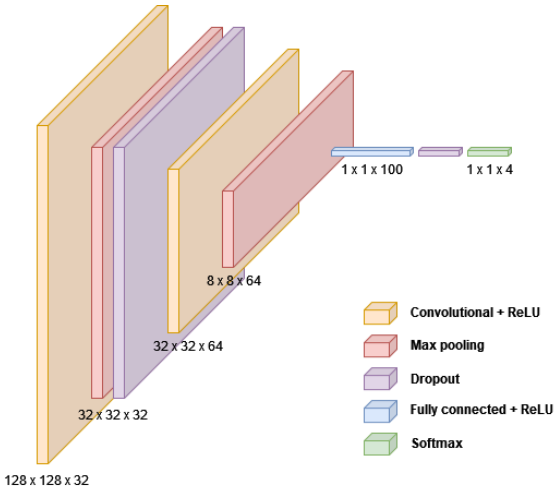


Figure 4.12: Third CNN architecture

The fourth and final structure, shown in Figure 4.13, begins with a 32 3x3 kernels convolutional layer ensued with a max-pooling layer with a kernel size of 4 and a dropout layer with 20% drop rate. The middle section has a convolution layer with 32 5x5 kernels, a max-pooling layer with a 4x4 kernel and two convolutional layer with 64 5x5 kernels each. The last three layers of this structure are a fully connected layer followed by a dropout layer with 20% rate and another fully connected layer.

4.4.2 OpenMax Model

As mentioned in Chapter 3, OpenMax was created with the purpose of adapting a DNN to an open-world situation by replacing the activation function of the last layer by an algorithm that integrates a Weibull continuous probabilistic distribution model. Since OpenMax is external to the DNN, being in this case a CNN, it is necessary to first train the neural network and then use the same training set to build the probabilistic model, so that they are in harmony. This process starts by

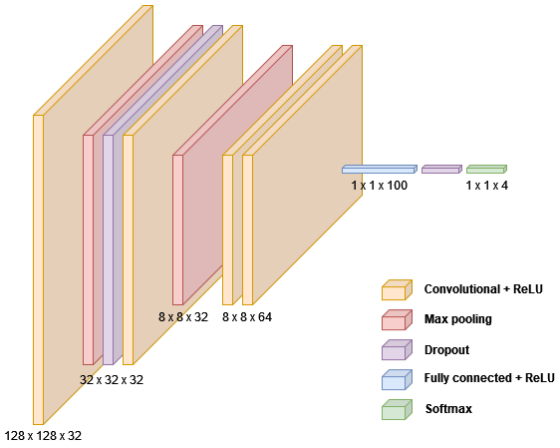


Figure 4.13: Fourth CNN architecture

providing the data to the CNN again, this time for classification, extracting from the correctly classified instances the values resulting from the last hidden layer, which the authors call activation vector. Note that, since it is about the last layer, recalling Subsection 2.2.1, the size of this vector is equal to the number of known classes.

Once all the data is provided, the vectors are arranged by classes, where for each one is calculated the mean and the distances between the vectors of that class to its mean. Then, from each group of distances, a predetermined number of the largest ones is selected, which number is given the name *tail*, and fed to the "fit_high" function from the library mentioned in Section 4.1.3. With this, the OpenMax model is considered trained, having as many Weibull continuous probabilistic distribution models as there are known classes.

At this point, it is possible to use the model for detection. Starting by feeding a new image to the CNN, both the activation vector and the probability vector, given, in this case, by the Softmax activation function, are extracted.

In a first moment, the probability vector is sorted and a predetermined number of the highest probability classes are selected as relevant for the given sample. To these classes is assigned a rank of importance that is related to its probabilistic value, i.e. the greater the probability of the sample belonging to the class, the greater is its rank. The predetermined number is referred to as *alpha*, which must be greater than 1 and can be as large as the number of known classes.

In a second moment, the distance from the activation vector to the mean of each class is calculated, with each distance being used by the function "w_score", also belonging to the libMR library, returning the Weibull score.

Then, for a given class, the new probability is calculated according to equation 4.1, where $score_c$ is the Weibull score of class c and $rank_c$ is the importance rank of the same class.

$$new\_probability_c = old\_probability_c * (1 - score_c * rank_c) \tag{4.1}$$

Since the new probabilities cannot be higher than those provided by the Softmax activation function, the sum of these may not add up to 100%. Therefore, the probability that no known class is chosen is calculated through the Equation 4.2, where c corresponds to the number of known classes.

$$probability_{unknown} = \sum_{n=1}^{c} old\_probability_n - new\_probability_n \tag{4.2}$$

Combining the output of the two previous equations, a vector with size equal to the number of classes plus one is produced, where the first class is labeled as unknown. To eliminate possible residuals, the Softmax activation function is applied to this vector.

Chapter 5

Experimental Setup

Having establishing the methods and materials in general, it is necessary to define the training and testing process performed in this work, including the specifications of the machine and tools used and the class selection, as well as the specific methods related with the evaluation of the solution. All these topics are covered in this chapter.

5.1 Machine and Environment Specifications

All the work, more specifically training and testing, was developed on the same machine and environment so that the results obtained would be stable and unbiased. Their specifications are provided in this section so that others can to reproduce similar results.

The machine specifications are:

- Operating system: Microsoft Windows 10 Education 10.0.19042
- Processor: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz, 22021 MHz, 6 Cores, 12 Logical Processors
- Random-access memory: 8GB DDR4
- Graphics card: NVIDIA GeForce GTX 1050
- CUDA: Version 11.2

The environment specifications are:

- Python: Version 3.7.11
- Tensorflow: Version 2.8.0
- Keras: Version 2.6.0

- NumPy: Version 1.20.3
- Matplotlib: Version 3.5.1
- Scipy: Version 1.7.3
- OpenCV: Version 4.5.5

The CUDA version is made available as it was used to speed up the training and testing process.

5.2 Classes Selection

Despite the relatively large number of classes available in the data set (30 classes as mentioned in Section 4.2), not all are possible to use. In this subsection, with the support of Figure 5.1, we indicate the excluded classes (highlighted in red) along with the reason for exclusion, and the separation of the remaining ones into known and unknown classes (highlighted in green and blue) necessary for the development of the work.

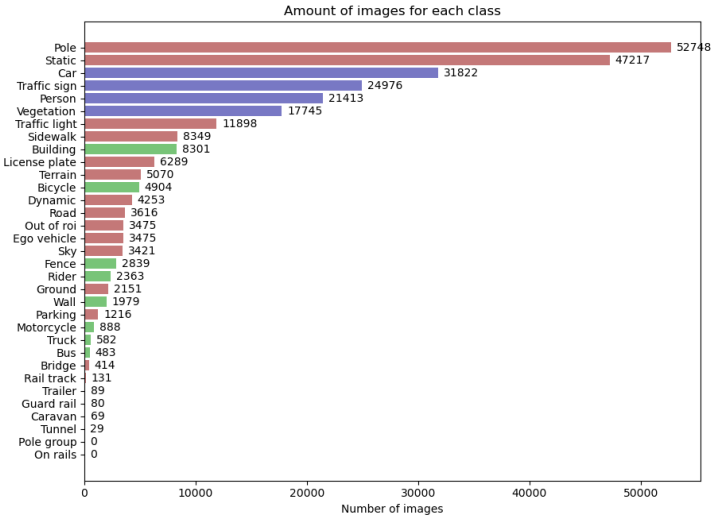


Figure 5.1: Number of images for each class

5.2.1 Excluded Classes

We start by listing the excluded classes, as choosing these is easier than selecting which ones to use.

We firstly dismiss the classes *Rail track*, *Trailer*, *Guard rail*, *Caravan*, *Tunnel* and *Pole group* due to their low volume of samples to either train a CNN or to use as

unknown classes in the DBSCAN algorithm, where they are most likely identified as outliers.

Secondly, even though it has a higher volume than the previous ones (around 3 times more samples), the class *Bridge* is an extra because it would not affect the drive if it were not for its pillars, which could be treated as another class (*Wall*). Thus, we also exclude it.

Then, we disregard the classes *Sidewalk*, *Terrain*, *Road*, *Ground* and *Parking* since they have similar colors, locations, and purpose (a space where cars and people travel). Otherwise, for this work, these classes could be combined into a single one.

We also find *Static* and *Dynamic* classes inadmissible as they are comprised of more than one object, i.e. in just these two classes, elements such as dogs, birds, street lights, garbage containers, traffic signs seen from behind, among others, may be found.

Having discarded all these classes, we are left with some futile ones such as *Sky*, given that it is not something that a driver has to pay attention or even consider while driving, and *Pole*, since they are complemented by either a lighting component, whether it is traffic or street light, or a signaling component (other classes). Therefore, these classes are also disregarded.

One last excluded class is *Traffic light* as they are similar to *Traffic signs* but with a lower volume of samples. However, this class could also be considered.

5.2.2 Accepted Classes

With the remaining 12 classes, we split them up into known, used in training the CNN, and unknown classes, used by the OpenMax and DBSCAN algorithms. As this report's focus is on the distinction between the multiple classes within the unknown samples detected by the OpenMax algorithm, only 4 of the 12 classes are used as known to achieve both the widest possible range of classes for distinction as well as having a healthy number of annotated classes.

Given that cars (*Car*) and people (*Person*) are the most important elements that drivers must be aware of while driving, these hold two of the known class slots. This assignment has benefits since, among the remaining 10 classes, there are variances of these two, such as *Truck*, *Bus*, *Rider*, among others, useful to understand the behavior of OpenMax when facing two similar classes, one of which is unknown, and DBSCAN when facing two similar unknown classes.

Two additional interesting classes to consider for known classes are *Traffic sign* and *Vegetation* because of their constant appearance in front of other classes like *Car* and *Building*.

Having filled the four available slots for known classes, we have the remaining eight as a single unknown class. These are arranged into groups of 2, with each group including the classes from the previous group. The goal of each is to study

the architecture's behavior towards similar known and unknown classes, for example *Car* versus *Bus* and *Truck*, the behavior towards similar unknown classes, as is the case of *Bus* versus *Truck* and *Fence* versus *Wall*, and the behaviour towards the increasing number of classes in the unknown component. The formed groups are displayed in the following list.

- Group 1: *Bus* and *Truck*
- Group 2: *Fence* and *Wall*
- Group 3: *Building* and *Bicycle*
- Group 4: *Rider* and *Motorcycle*

5.3 Training Convolutional Neural Networks

Having, in the previous section, selected which classes to be used in this work, including the separation between known and unknown classes, we can now start thinking about the process of training the neural networks, for which are used the images belonging to one of the 4 known classes.

However, as it is possible to notice in Figure 5.1, the difference between the class with highest and lowest volumes is almost the double so, to prevent the data set from being skewed, the number of instances of each class was reduced to the same amount as *Vegetation* (class with lowest volume). This means that we must select which instances to use from the classes with the higher volume, thus, in an attempt to prevent another problem, this time being a biased data set, these instances are chosen randomly.

To measure the training success of both CNNs and OpenMax it is important to reserve a portion of the data set for testing. Therefore, we randomly divide it, with 85% of it being used for training and the remaining 15% for testing. Still, the CNN is subject to biased training since its values are tuned based on the same images over and over again. Hence, we divide the training data set even more, so that it comprises of only 60% of the original data set, and the extra 25% is used for validation. This way, when the CNN is trained, at each tuning cycle, there is a validation of the new adjusted values, decreasing the model bias.

It is important to note that, as we use 4 different CNN architectures in our work, in order to compare their results, we used the same training, validation and test sets. These are also applied in the OpenMax model.

5.4 Training and Testing OpenMax

As stated in Subsection 4.4.2, OpenMax needs to be trained with the same images provided to CNNs, in order to get the means of each class as well as the sample

distances to those means. As such, the training and validation set were provided to the neural network, corresponding to 85% of the data set.

Having trained the OpenMax, we feed it 5 different test sets. The first test set is the one created in the previous subsection, corresponding to 15% of the total data set, which only contains the known classes (*Car*, *Person*, *Traffic sign* and *Vegetation*). The remaining 4 test sets, in addition to including the first test set, also include instances from unknown classes corresponding to their group, presented in Subsection 5.2.2. In other words, the second test set, besides the four known classes, contains classes *Truck* and *Bus* (*Group 1*) considered as unknown, the third test set has the four known classes and classes *Truck*, *Bus*, *Fence* and *Wall* (*Group 1*) considered as unknown, and so forth. The combination of known and unknown classes mixed in the test set has two purposes, the first of which is to test the quality of the neural network on the known classes, and the second is to simulate an environment where known and unknown elements co-exist.

It should be noted that we have to balance the test sets just as was necessary for the known classes. We attempt to ensure that every set of 2 unknown classes takes up 12.5% of the original test set, with each of this pair occupying half. There are two reasons for this, the first reason being that it is a reasonable fraction to not overshadow the number of instances of known classes while still being able to use a significant amount of unknown instances, and the second reason being that *Group 1* does not have enough instances to satisfy higher percentages. As it has been done so far, the instances for each unknown class are randomly selected.

As it is a dependent solution of the problem, the parameters addressed in Subsection 4.4.2, *alpha* and *tail*, play an important role on detecting instances form unknown classes and so, they must assume multiple possible values to have the possibility of achieving good results. For that, we define the following: *alpha* can assume the possible values, which are all integer numbers from 1 to 4, and, in order to understand the impact of low and high values, the parameter *tail* can assume [50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 1000].

Once trained and tested, the combination (CNN + OpenMax + *alpha* + *tail*) with the best results for a given test set moves on to the next stage. The quality is measured not only the number of instances belonging to the unknown class but also the F-score value.

5.5 Experimenting with DBSCAN

This is the last yet most important stage of our solution because it is here that we receive the instances classified by the OpenMax method as belonging to one major unknown class and identify multiple smaller classes within it.

It is worth mentioning that OpenMax's model not only detects instances belonging to unknown classes, but also processes them, as they are fed one more time to the CNN in order to extract the activation vectors. This way, instead of supplying the DBSCAN algorithm with data of size 128x128, only vectors of size 4 are

provided, given that there are 4 known classes.

Since DBSCAN is also a problem-dependent solution, there are two parameters, ϵ and minimum number of neighbors, mentioned in section 2.2.2, that have to assume multiple values until we find a combination that approximates the number of clusters to the number of real unknown classes, with each cluster having mostly instances corresponding to the same class. The choice for the values of the minimum number of neighbors is based on the following reasons. Since a given point is taken into account as its neighbour, we excluded the possibility that the minimum number of neighbors is 1. As we do not want two close outliers to form a class, we also excluded the possibility that the minimum number of neighbors is 2. Finally, to understand the impact that a high values have on the solution, we set as upper limit 18. Thus, the values that this parameter can assume are all integer numbers between 3 and 18 inclusive. As for ϵ, it is more difficult to narrow it because it requires a knowledge of the distances between the different instances. Therefore, we choose all values from 0.1 to 1000.0 in intervals of 0.1.

Having defines the DBSCAN experimentation process, it is necessary to establish the desired outcomes. Given that the purpose of this algorithm is to find as many clusters as there are unknown classes in the test set, we should have this as a quality measure. However, using only this variable alone, we cannot claim that this algorithm solves the problem at hands. For this, we also take into account the percentage of samples of each class per cluster.

Chapter 6

Results and Discussion

In this chapter we display the experimental results and analyze the performance of the models.

6.1 Experimental Results

In this section we present the experimental results obtained from OpenMax, both F-score and the number of correctly classified unknowns, for all combinations of *alpha* and *tail* in each CNN, grouped by test sets. Following this, we also present the results obtained from the DBSCAN algorithm for all combinations of the parameters ϵ and minimum number of neighbors. Recalling Section 5.4, this algorithm is applied on the best CNN/OpenMax architecture for each test set only.

At first, we provide the results for a test set which only contains instances belonging only to the known classes. Therefore, for this case, only display the F-score for all combinations of *alpha* and *tail* for the four CNNs, Figure 6.1.

Then, since all the remaining test sets include instances belonging to unknown classes, we present, for each test set, the F-score obtained for all combinations of *alpha* and *tail* for the four CNNs, displayed in Figures 6.2, 6.4, 6.6 and 6.8, and the number of instances belonging to unknown classes correctly classified, shown in Figures 6.3, 6.5, 6.7 and 6.9.

It should be noted that throughout the images, the id of the CNN is indicated in the title ("CNN 1", "CNN 2", "CNN 3" and "CNN 4"), and this id represents which of the CNNs presented in Subsection 5.3 is used. In other words, "CNN 1" corresponds to the first presented architecture, "CNN 2" corresponds to the second architecture, and so on.

Finally, we present the results of the DBSCAN algorithm for each of the test sets containing instances of unknown classes (Tables 6.1, 6.2, 6.3 and 6.4). These results are merely the most interesting ones since there are numerous combinations, with the majority of them producing repeated results.

It should be noted that the percentage represents the portion of instances of a

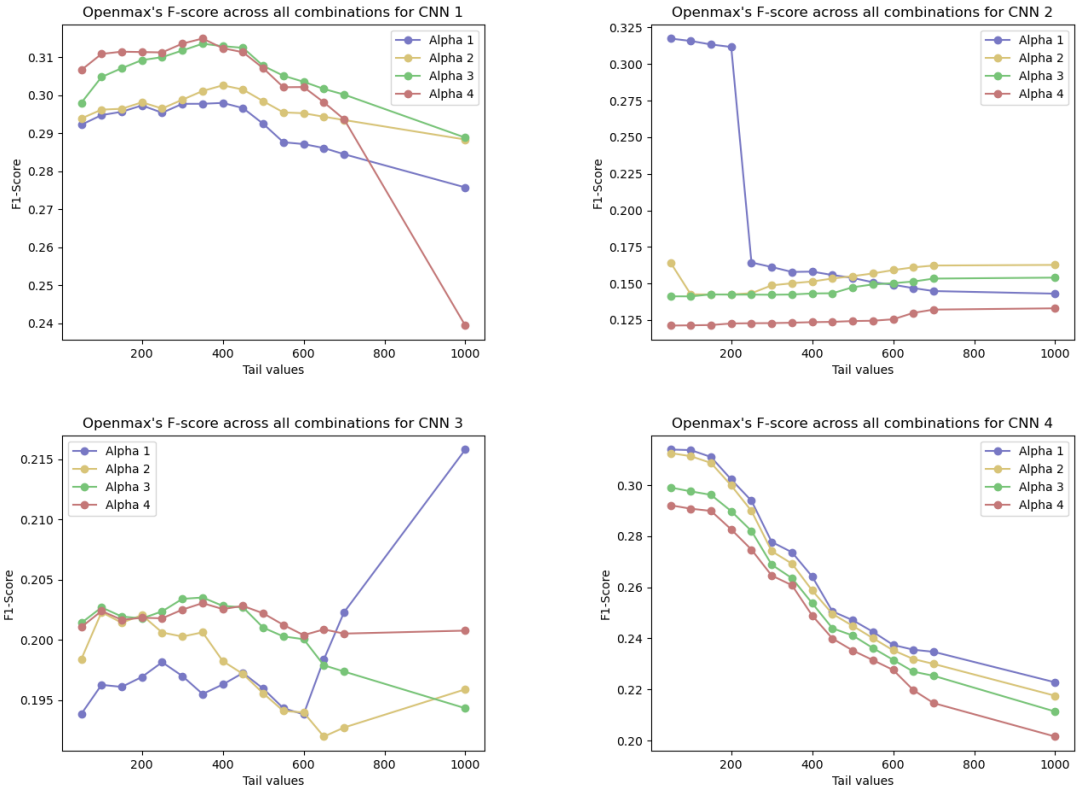


Figure 6.1: OpenMax's F-score across all combinations of *tail*, *alpha* and CNNs given the test set with no instances belonging to unknown classes

given class within each cluster, i.e. the sum of the rows in a column for a given combination can exceed 100%, however, the sum of the columns in a row can only add up that value and, when it is not able to achieve it, that is because the algorithm deemed the remaining as outliers.

6.2 Discussion

In this section we analyze and compare the data presented in the previous section, dividing it by the algorithms OpenMax and DBSCAN. During the discussion of the results provided by each of these algorithms, we divide them further by test set.

6.2.1 OpenMax

First Test Set

Our goal in using a test set with only instances belonging to known classes is to evaluate the quality of the implemented CNN + OpenMax model and understand the impact of the parameters *alpha* and *tail*.

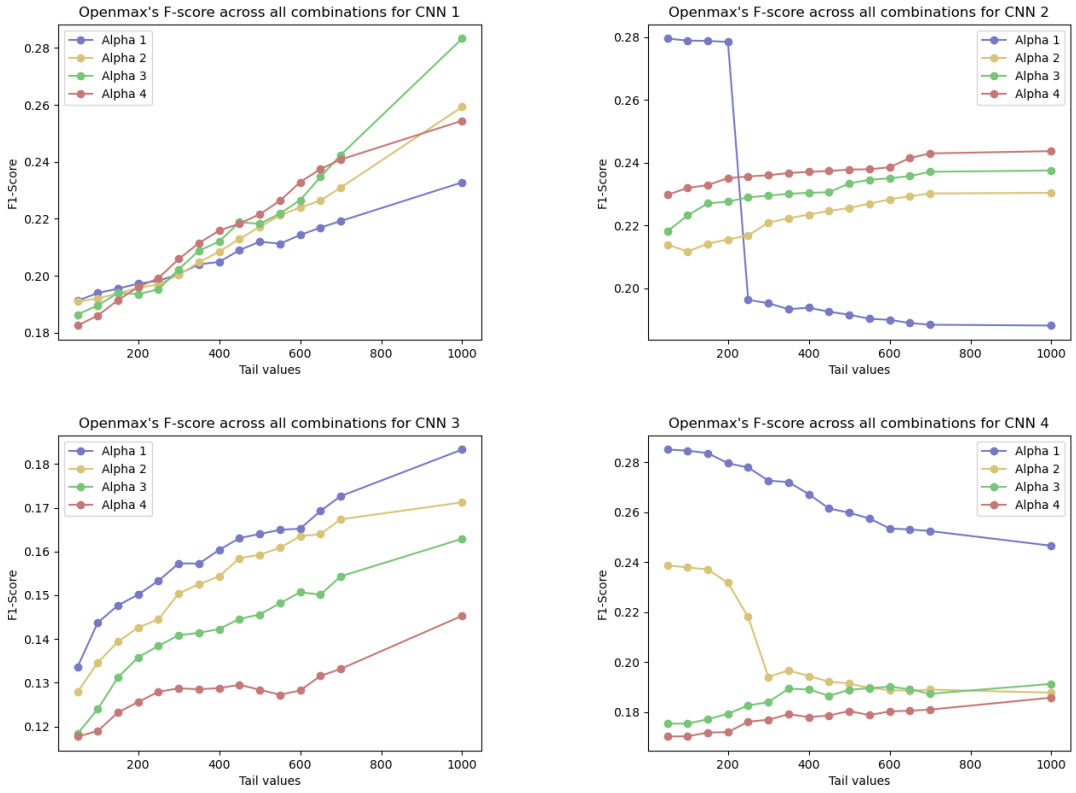


Figure 6.2: OpenMax's F-score across all combinations of *tail*, *alpha* and CNNs given the test set with instances belonging to classes in *Group 1*

Classes	ϵ	Min Nei	Clusters	
			1	2
Truck	265,1	3	90,0%	1,3%
Bus			82,2%	0,0%
Truck	306,2	3	95,5%	1,3%
Bus			79,7%	0,0%
Truck	175,3	5	5,2%	58,9%
Bus			0,0%	65,2%
Truck	96	6	10,8%	1,3%
Bus			0,0%	24,6%
Truck	102,3	9	12,1%	2,2%
Bus			0,0%	24,7%
Truck	108,1	15	2,6%	12,1%
Bus			23,7%	0,0%
Truck	108,7	16	13,9%	1,7%
Bus			0,8%	22,0%

Table 6.1: Percentage of samples of each unknown class in *Group 1* found per cluster given an ϵ and minimum number of neighbors

According to the obtained results, for the CNNs with architectures 1, 2 and 3, varying *alpha* and *tail* has no significant impact on the F-score. That is, changing these, with some exceptions, causes a variation in the F-score in the range of 0.03

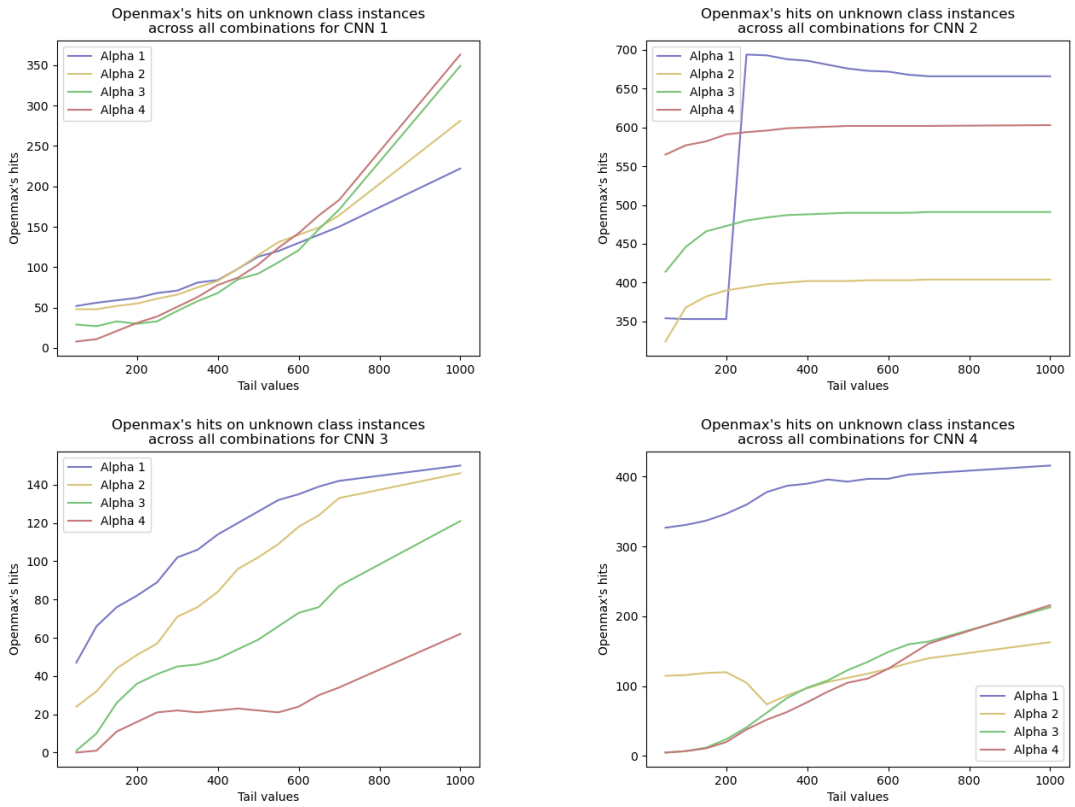


Figure 6.3: Amount of instances that belong to unknown classes correctly classified by OpenMax across all combinations of *tail*, *alpha* and CNNs given the test set with instances belonging to classes in *Group 1*

in architecture 1, 0.05 in architecture 2, and 0.015 in architecture 3. This occurs due to the fact that the most distant instances are close to the means. Thus, the new probabilities of the known classes are not sufficiently smaller to allow the probability of the new class (*unknown*) to be greater than them.

It is important to note that, although peaks occur for the values of *tail* = 1000 and *alpha* = 4 in architecture 1 and *tail* = 1000 and *alpha* = 1 in architecture 3, due to its small magnitude (only about 0.05 for the former and 0.10 for the latter), it is not necessary to consider them as abnormal behavior. It is, however, worth considering the difference in F-score between the combination of values of *alpha* = 1 and *tail* = [50, 100, 150, 200] with the remaining ones since this peak has a magnitude of about 0.150. The reason for this occurrence is that, for these combinations, the model classifies half of the instances as belonging to the *Traffic light* class instead of only 25%, correctly classifying most of them.

Unlike the other CNNs, the F-score value decreases with increasing *tail* for the fourth architecture. However, changing the *alpha* does not significantly impact it, with the variation being about 0.02.

Overall, although all the F-score values are low, the architecture capable of producing the best results for this test set is the first one because while achieving an F-score as good as the second and fourth architectures, the F-values are constant and is not confined to classify all the instances as a single class.

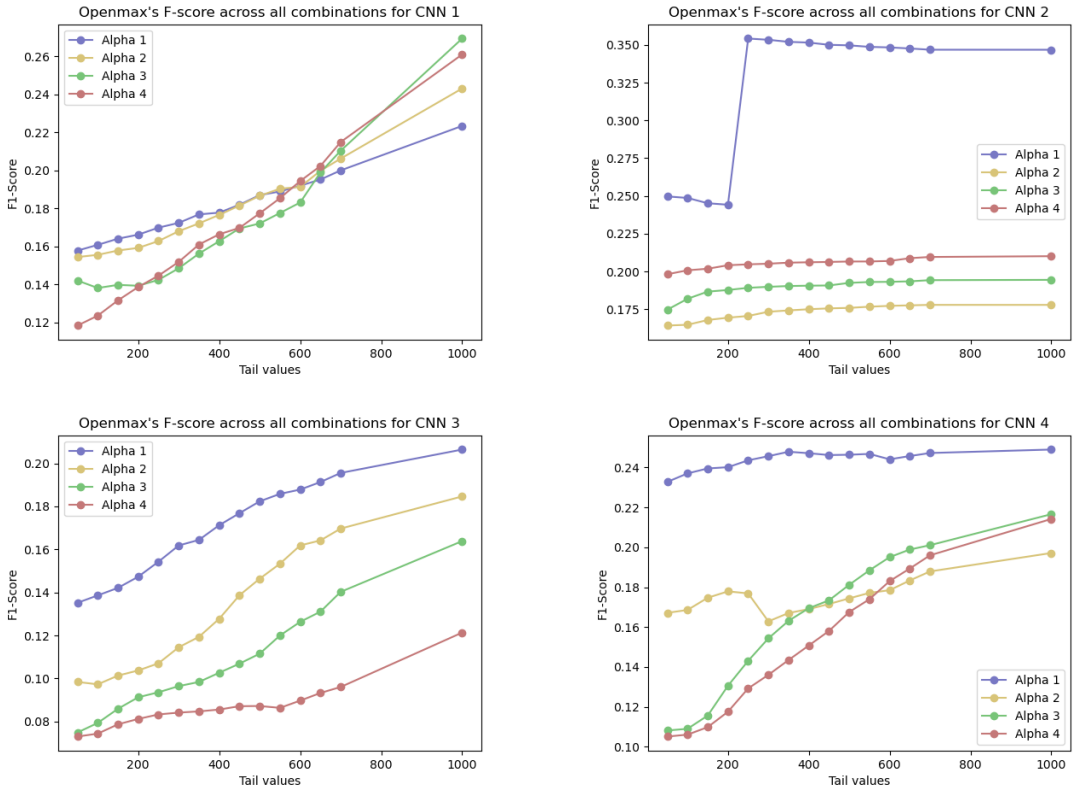


Figure 6.4: OpenMax's F-score across all combinations of *tail*, *alpha* and CNNs given the test set with instances belonging to classes in *Group 2*

Second Test Set

The second test set is similar to the previous test set with the addition of classes belonging to *Group 1* (*Truck* and *Bus*). With this set, we already intend to analyze the model's detection ability on instances not belonging to any of the four known classes.

For this test set it is possible to notice a different behavior when comparing it with the previous one. While the alpha does not have a significant impact, with some exceptions, the increase of the *tail* has, in general, a positive influence on the F-score. The reason for this is that now we have instances belonging to an unknown class, these being the furthest from any class mean, and the more we take them into account on the Weibull model, the better the model fits the test data.

Continuing to take into account the results of the previous test set, we find a repeated problem. The second CNN architecture, for the values of *alpha* = 1 and *tail* = [50, 100, 150, 200], produces significantly high F-score values when compared to the other combinations. Since the problem and reason persist (biased results), this architecture is disregarded for this test set.

Although it is new, another problem is found in this test set, this being with the third CNN architecture. Comparing the results of this one with the remaining two, both the F-score values and the amount of instances belonging to unknown

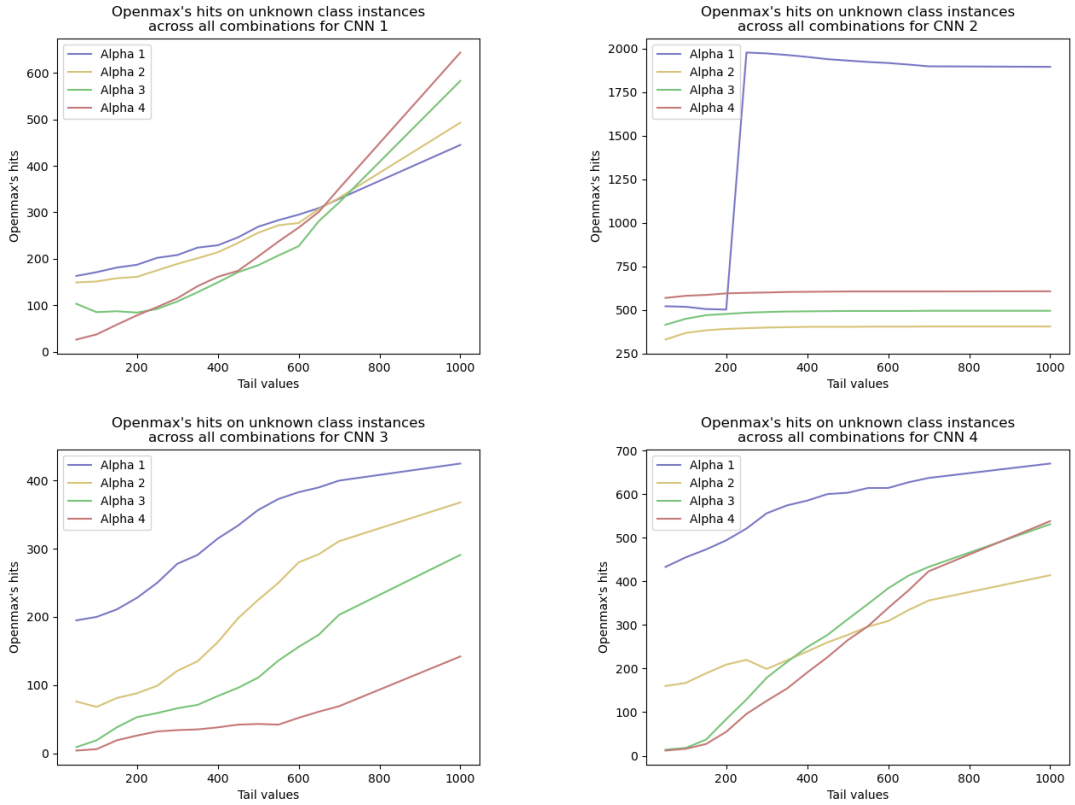


Figure 6.5: Amount of instances that belong to unknown classes correctly classified by OpenMax across all combinations of *tail*, *alpha* and CNNs given the test set with instances belonging to classes in *Group 2*

classes found are significantly lower, and so this architecture is also discarded for this test set.

As a consequence of these exclusions, we are left with two architectures, 1 and 4. Starting with the latter, we may observe a strange behavior for *alpha* value = 1. That is, when raising the *tail* value, the F-score decreases while the number of instances belonging to well classified unknown classes increases. This is due to the fact that the Weibull model classifies more instances as belonging to unknown classes as the value of the parameter *tail* becomes bigger. This way, the amount of true positives is reduced (decreasing F-score) while the amount of the unknown class instances correctly classified grows. Therefore, from this architecture, we can only consider the other results of the remaining *alpha* values but, as these are lower than the ones produced by the first architecture, we deem this architecture as not suitable for this test.

Thus, we remain only with the first architecture. As it does not pose any major issues, besides the low F-score values, it is the one to proceed on to the next stage, more specifically with the parameter *tail* = 1000 since this produces the highest F-score values for any *alpha* parameter. There is, however, an indecision in the choice of *alpha* value, where for *alpha* = 3 the F-score value is greater while for *alpha* = 4 the number of instances belonging to the unknown class is higher. Due to the fact that the F-score better describes the classification quality of the CNN, thus having more importance in this tie, and the difference in the number

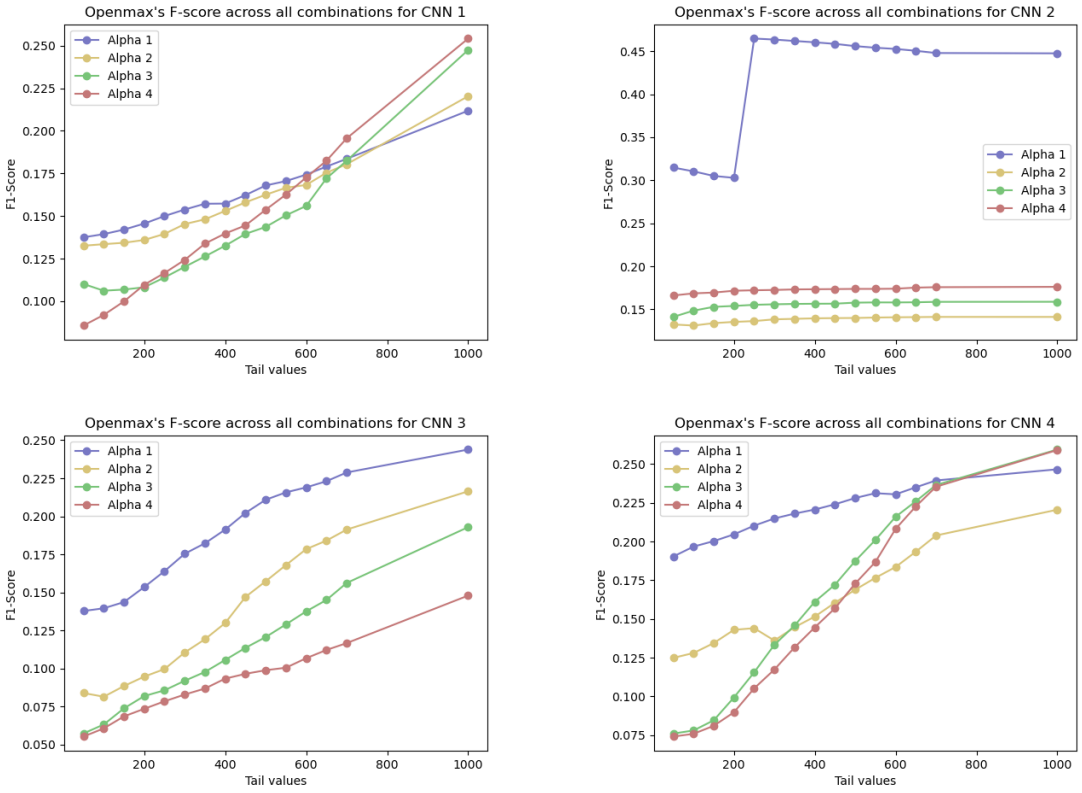


Figure 6.6: OpenMax's F-score across all combinations of *tail*, *alpha* and CNNs given the test set with instances belonging to classes in *Group 3*

of instances belonging to the unknown class between the two values of *alpha* is minimal, we proceed to select *alpha* = 3. The rate of correctly classified unknown class instances for this combination is 32.8%.

Third Test Set

As mentioned in Subsection 5.2.2, the third set includes instances from the set of known classes as well as instances from the unknown classes in *Group 1* (set used in the previous test set), and *Group 2* (*Fence* and *Wall*) (set to be used in this test set).

Most of the behaviors detected in the second test set are still present in this one. The first similarity is found in the impact of the *alpha* and *tail* parameters, where *alpha*, with some exceptions, does not significantly impact the two dependent variables, i.e. the F-score and the number of samples belonging to the unknown class correctly classified, while the *tail* parameter has a positive influence on them. The second similarity is the sharp increase in the both dependent variables for the second architecture given *alpha* = 1 and *tail* = 250. Therefore, as was decided for the previous test set, we disregard this architecture for the third test set. The third and last similarity lies in the low values of the dependent variables obtained with the third architecture. In terms of F-score, seemingly not, the remaining, still considered, architectures have an improvement of about 20% to 30%. In terms of

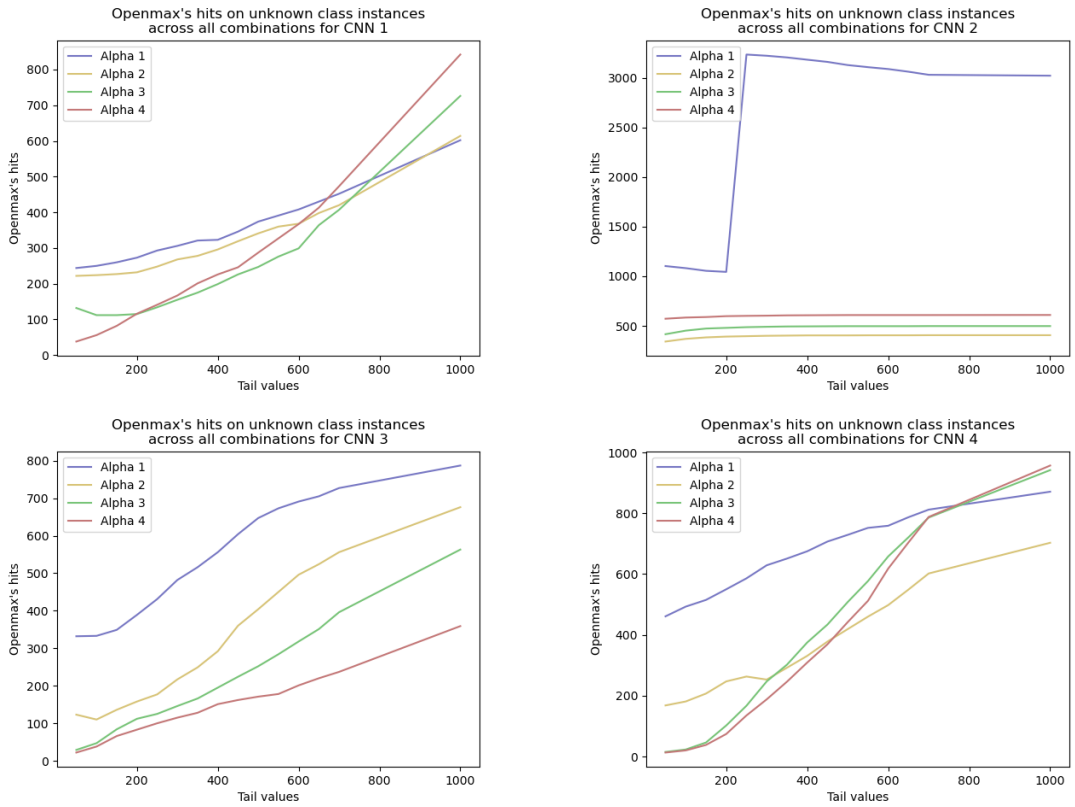


Figure 6.7: Amount of instances that belong to unknown classes correctly classified by OpenMax across all combinations of *tail*, *alpha* and CNNs given the test set with instances belonging to classes in *Group 3*

the number of samples correctly classified as belonging to the unknown class, the other architectures have an improvement from 50% to 75%. For these reasons, we also exclude this architecture.

This narrows it down again to the first and fourth architectures. Comparing these two architectures, we can observe that the first one achieves higher F-score values (about 8% increase) while the other one achieves a higher number of samples correctly classified as belonging to an unknown class (around 14% gain). It should also be noticed that for the parameter alpha = 1, the fourth architecture has an unusual behavior, where the F-score does not increase with values higher of *tail* (*tail* > 350), and since we focus more on this dependent variable, as the activation vectors are used in the next phase, we prefer the first architecture.

This then simply requires us to choose the best combination of parameters. Since the two dependent variables achieve their highest values when tail = 1000, all that is left is to define the value of alpha. For the same reason when choosing the value of the parameter alpha for the previous test set, we opt for alpha = 3. The rate of correctly classified unknown class instances for this combination is 24.3%.

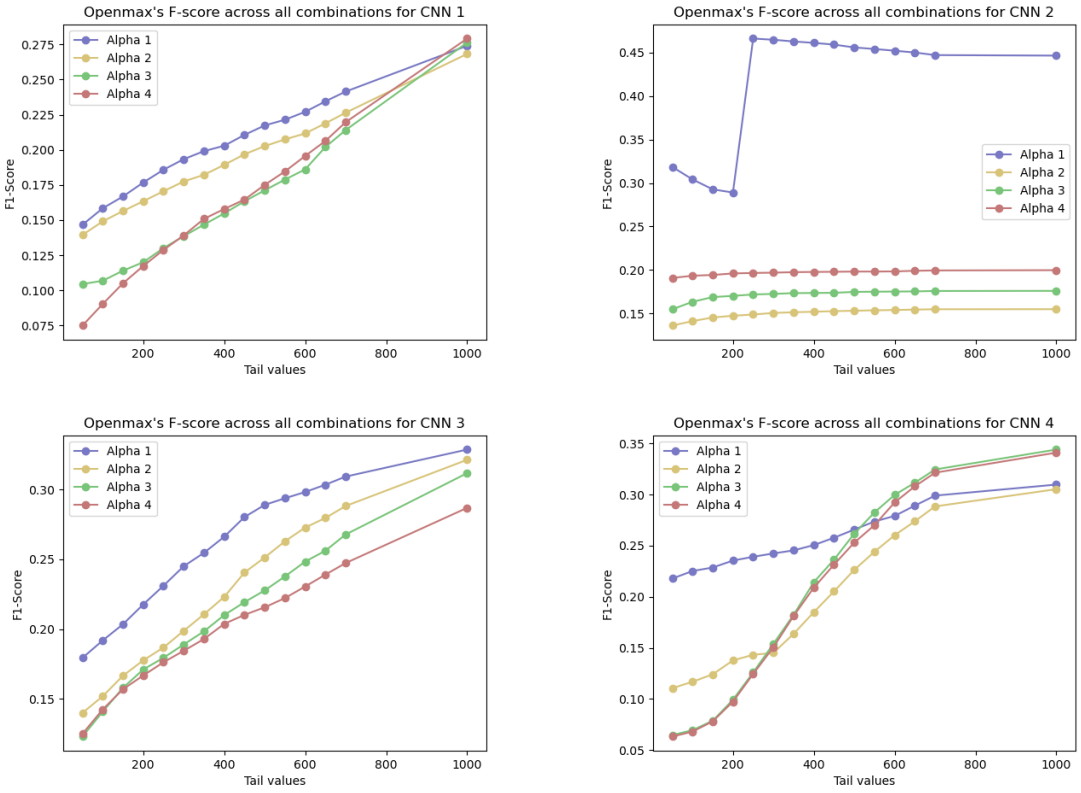


Figure 6.8: OpenMax's F-score across all combination of *tail*, *alpha* and CNNs given the test set with instances belonging to classes in *Group 4*

Fourth Test Set

Similar to the previous test set, this fourth one consists of instances belonging the set of four known classes, instances from the previous groups 1 and 2, and instances from the *Group 3* (*Building* and *Bicycle*), destined to be used in this test set.

For this test set, we continue to observe behaviors seen so far. One of these is the impact of the alpha and tail parameters on the dependent variables. A second repeated behaviour is the sharp increase of the dependent variables for the second architecture given the values of the parameters *alpha* = 1 and *tail* = 250, leading us, as it was done for the previous test sets, to disregard this architecture. Another behaviour is the low values, even if just by a small amount, for both dependent variables on the third architecture, when comparing with it the other two, forcing us to also exclude it for this test set.

Yet again, we are left with only the first and fourth architectures but, this time, apart from the difference of the dependent variables between *alpha* = 1 and the others for initial values of *tail*, the latter architecture does not exhibit significant issues to be discarded. In fact, on top of yielding approximately the same F-score as the first architecture, it has a higher amount of samples belonging to the unknown class correctly classified. For this reason, the fourth architecture is chosen for this test set.

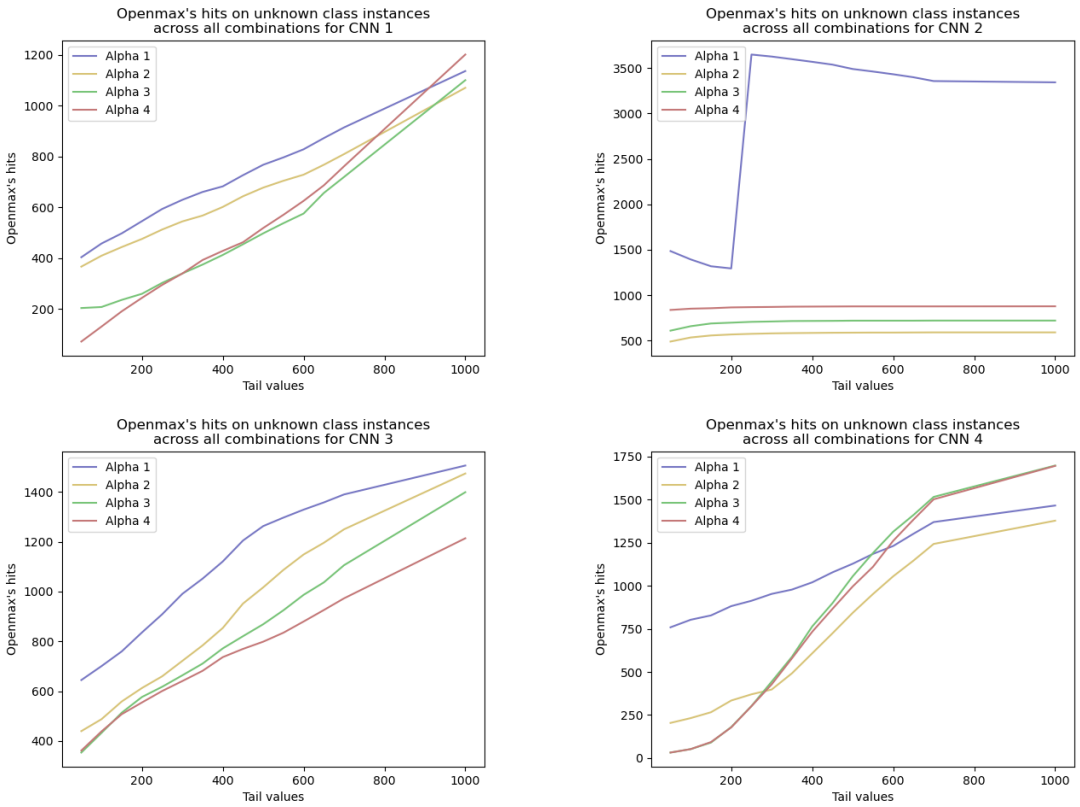


Figure 6.9: Amount of instances that belong to unknown classes correctly classified by OpenMax across all combinations of *tail*, *alpha* and CNNs given the test set with instances belonging to classes in *Group 4*

Thus, there remains the choice of the best set of parameters, having tail = 1000 since, regardless of the alpha, it provides the highest values of the dependent variables. Noting that there is no difference in the F-score between alpha = 3 and alpha = 4, the one with the highest volume of samples belonging to the unknown class correctly classified is chosen, this being alpha = 4. The rate of correctly classified unknown class instances for this combination is 25.7%.

Fifth Test Set

This is the last test set to be used on the OpenMax model and, similarly to the previous ones, this set includes instances of all classes that make up the fourth test set as well as instances belonging to the classes within *Group 4* (*Rider* and *Motorcycle*).

By comparing the results of this test set with the results of the previous ones, it is possible to observe some repeated events. These are the impact that the *alpha* and *tail* parameters have on the dependent variables and the abrupt change in the values of these variables for *alpha* = 1 and *tail* = 250 on the second architecture, discarding it for the fifth time.

However, it is possible to find a new situation. As we have been noticing, the increase in the number of instances of unknown classes causes a decrease in the

Classes	ϵ	Min Nei	Clusters			
			1	2	3	4
Truck	75,1	3	0,0%	0,0%	12,7%	0,0%
Bus			1,2%	5,2%	1,2%	0,0%
Fence			0,0%	0,0%	31,7%	2,4%
Wall			0,0%	0,0%	36,0%	0,9%
Truck	85,2	3	0,0%	0,0%	20,3%	0,0%
Bus			2,8%	1,6%	9,6%	0,0%
Fence			0,0%	0,0%	34,1%	2,4%
Wall			0,0%	0,0%	36,9%	1,8%
Truck	230,6	3	74,6%	0,0%	0,0%	0,0%
Bus			77,1%	1,6%	0,8%	0,0%
Fence			56,9%	0,0%	0,0%	1,6%
Wall			54,0%	0,0%	0,9%	0,9%
Truck	175,3	3	0,0%	66,1%	0,0%	0,0%
Bus			4,8%	54,6%	1,6%	1,6%
Fence			0,0%	53,7%	0,0%	0,0%
Wall			0,0%	50,6%	0,0%	0,0%
Truck	84,8	4	0,0%	16,1%	0,0%	0,0%
Bus			8,0%	1,6%	1,6%	0,0%
Fence			0,0%	0,0%	34,1%	2,4%
Wall			0,0%	0,0%	36,9%	1,8%

Table 6.2: Percentage of samples of each unknown class in *Group 2* found per cluster given an ϵ and minimum number of neighbors

Classes	ϵ	Min Nei	Clusters					
			1	2	3	4	5	6
Truck	23,8	3	28,4%	1,1%	0,0%	0,0%	0,0%	0,0%
Bus			49,2%	0,0%	0,0%	0,0%	0,0%	0,0%
Fence			2,5%	36,3%	1,3%	0,0%	0,0%	0,0%
Wall			0,0%	29,1%	0,0%	1,2%	0,0%	0,0%
Building			64,5%	0,0%	0,0%	3,2%	0,0%	0,0%
Bicycle			0,0%	0,0%	0,3%	8,8%	0,8%	0,8%

Table 6.3: Percentage of samples of each unknown class in *Group 3* found per cluster given an ϵ and minimum number of neighbors

results produced by the first architecture, being in this test set that architecture 1 presents the lowest results in relation to the other two. For these reason, we disregarded it. The opposite is true for the last architecture, where raising the number of unknown class instances causes an increase in the results obtained, and for this test set, despite the similar F-score values, the third architecture cannot compete with about 300 fewer samples correctly classified as belonging to the unknown class.

Once again, the parameter tail = 1000 produces the highest results, and the best match is given by alpha = 3 since, although the volume of instances belonging

Classes	ϵ	Min Nei	Clusters							
			1	2	3	4	5	6	7	8
Truck	100,6	3	0,0%	76,9%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
Bus			0,0%	81,1%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
Fence			82,6%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
Wall			77,0%	0,6%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
Building			0,0%	89,7%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
Bicycle			0,0%	0,5%	0,8%	0,0%	0,0%	57,6%	0,0%	0,0%
Rider			6,6%	34,4%	0,0%	0,6%	1,3%	0,6%	0,6%	0,6%
Motorcycle			0,5%	33,0%	0,5%	0,0%	0,0%	0,0%	0,0%	0,0%

Table 6.4: Percentage of samples of each unknown class in *Group 4* found per cluster given an ϵ and minimum number of neighbors

to the unknown class correctly classified is similar to alpha = 4, it has a higher F-score. The rate of correctly classified unknown class instances for this combination is 33.6%.

Summary

In this sub-subsection, we proceed to summarize our findings in the results obtained from the OpenMax algorithm. In a first moment, we clarify the influence that the parameters *alpha* and *tail* have on the algorithm for a given test set. Secondly, we address the interesting behaviours of the second CNN architecture, followed by possible justifications for the low values of F-score for all architectures. Finally, we mention the best combination for each test set.

Let us start with the impact of the parameters on the OpenMax algorithm. When the test set has no instances belonging to the unknown class, the parameters, in general, have little to no influence on the F-score, except for the fourth architecture, where increasing the *tail* decreases the F-score. This low impact is due to the fact that the farthest instances are still very close to the classes'means. Thus, the difference between the new probabilities calculated by OpenMax for the known classes and the probabilities calculated by CNN is not enough for the probability of the new (*unknown*) class obtained from the OpenMax algorithm to be higher, regardless of the amount of farthest instances considered. However, introducing instances belonging to unknown classes causes an increase in the F-score. The reason for this is that this time, the instances farthest from the means are actually far away, and the more we consider them (the bigger the *tail*), the better the Weibull model is fit. This way, it is possible to decrease the new probabilities of the known classes in such a way that the new (*unknown*) class starts being used to classify instances, and more elements start to be correctly classified.

Next, we have the interesting behavior of the second architecture. The problem starts with a bad training, making the CNN overfitted and classifying every instance as belonging to the class *Vegetation*. As a result, the distance of every instance to the mean of this class is small. Having said this, by providing the probabilistic Weibull model with instances very close to the mean of a class, causes

the model to assert that everything close it is unknown. That is why, when only one class is considered as important (alpha = 1), for tail > 200 we find a sharp variation in the F-score.

Considering, more specifically, each test set, as the first one only contains instances belonging to known classes, the OpenMax model classifies most of them as belonging to the unknown class, resulting in a very low F-score (around 0.16). As more instances of unknown classes are added, the F-score increases, reaching a value of 0.47.

Having addressed the behavior of the second architecture, let us glance at the F-score values obtained. Since, for the best combinations, F-score values range from 0.25 to 0.4, we may conclude that the results obtained were not the best and there are multiple possible reasons for this. A possible reason can be found in the construction of the data set, both in treatment and in separation and balancing. Another possible problem can be found in the structure of the CNNs, which might not be the optimal ones for the problem at hand. A third and last one is with the OpenMax algorithm by not exploring enough this solution or by not being good enough for the data set used.

At last, the best combinations for each test set. For the one with instances from the unknown classes within *Group 1*, the best combination is the first CNN architecture with *alpha* = 3 and *tail* = 1000. Regarding the test set with unknown classes from *Group 2*, we find the best combination for the first CNN architecture with *alpha* = 3 and *tail* = 1000. As for the test set with class samples from *Group 3*, we define the best combination as the fourth CNN architecture with *alpha* = 4 and *tail* = 1000. The last test set, which includes samples of unknown classes from *Group 4*, we selected again the fourth CNN architecture but with *alpha* = 3 and *tail* = 1000.

6.2.2 DBSCAN

Second Test Set

As mentioned in the Subsection 6.2.1, the only use of the first test set is to test evaluate the quality of the OpenMax model and understand the influence of the parameters *alpha* and *tail*, and as it does not contain any unknown class instance, this test set is not used in this stage. Therefore, we begin by analyzing the results for second test set.

Recalling the process described in Subsection 5.5, we supply the OpenMax method with a test set, where the the instances classified as belonging to an unknown class are selected and provided to the DBSCAN algorithm, obtaining the intended results.

Before analyzing the results, it is necessary to mention that the percentages are obtained for 188 samples of the *Truck* class and 231 samples of the *Bus* class.

According to the results obtained, it is possible to detect two different outcomes.

First, DBSCAN classifies most of the instances as belonging to a single class, as it happens for the combinations ϵ = [265.1, 306.2] and minimum neighbors = 3, and ϵ = 175.3 and minimum neighbors = 5. Although it is not what is intended, this solution is understandable since the test set consists only of trucks and buses, with these two elements being similar, as it can be seen in Figure 6.10. In the other outcome, the algorithm is able to find two clusters with a significant amount of instances of just one class per cluster. That is, for example, for the combination ϵ = 102.3 and minimum number of neighbors = 9, the algorithm is able to isolate 12.1% of the Truck class samples in one cluster, containing no samples from the Bus class, and is almost able to isolate 24.6% of samples of the Bus class in the other cluster, having only 2.2% instances of Truck. Similar behavior occurs for the remaining combinations.

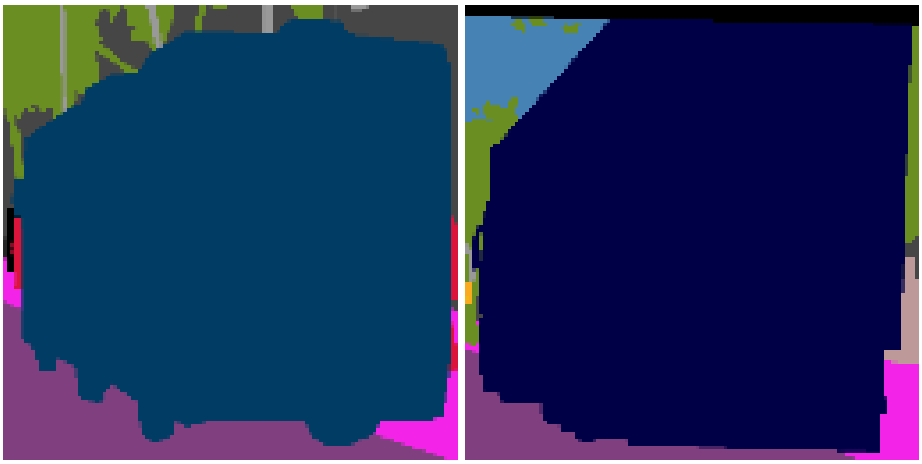


Figure 6.10: Example of a bus (right image) and a truck (left image) in the data set

Third Test Set

Similarly to the previous test set, the instances used by the DBSCAN are the ones deemed by the OpenMax method as belonging to unknown classes, being these unknown classes defined by *Group 2*.

As for the previous test set, it is necessary to mention the amount of samples per class to understand the dimension of the solution. The *Truck* class holds 118 samples, the *Bus* class contains 231 samples, the *Fence* class has 123 samples, and finally the Wall class consists of 111 samples.

Now, comparing the results obtained for this test set with those obtained with the previous one, it is possible to find a similar outcome. This outcome consists in the aggregation of a large percentage of instances of each class into a single cluster, as is the case with the combinations ϵ = 85.2 and minimum number of neighbors = 3 or ϵ = 230.6 and minimum number of neighbors = 3 or even ϵ = 175.3 and minimum number of neighbors = 3. Due to the low amount of samples from the class *Bus* clustered with the others, the combination ϵ = 75.1 and minimum number of neighbors = 3 do not quite represent the same outcome, forming a cluster with 3 classes.

Another outcome provided by this algorithm is the one with a combination of ϵ = 84.8 and minimum number of neighbors = 4, where the algorithm is able to find a cluster with 8% of samples belonging to the class *Bus*, a second cluster with mostly *Truck* class samples, containing 16.1% of them, a third cluster combined of mostly *Fence* and *Wall* samples (34.1% and 36.9% of the samples, respectively), and a fourth cluster with residues of the last two classes.

Thus, for this test set, the algorithm can distinguish between the *Truck* and *Bus* classes, and still create a class containing samples from the *Wall* and *Fence* classes.

Fourth Test Set

This test set consists of instances belonging to the unknown classes defined in *Group 3*, containing 88 samples from the *Truck* class, 128 samples from the *Bus* class, 157 samples from the *Fence* class, 165 samples from the *Wall* class, 31 samples from the *Building* class, and 388 samples from the *Bicycle* class.

For this test set only one interesting solution was found, as the remaining ones are just clusters consisting of a number of samples between 1 and 10. This solution is given by the combination ϵ = 23.8 and minimum number of neighbors = 3, and consists of 6 clusters. The first is mostly made up of samples from the *Truck* class, with 28.4% of its samples, the *Bus* class, with 49.2% of its samples, and the *Building* class, with 64.5% of its samples. The second cluster is mostly composed of samples belonging to the *Fence* class, with 36.3% of its samples, and *Wall* class, with 29.1% of its classes. The fourth cluster consists of 8.8% of the samples from the *Bicycle* class and 3.2% of the samples from the *Building* class. The other clusters have few samples from the various classes, not enough to determine anything in particular.

Hence, for this test set, the algorithm is able to divide samples of the *Bicycle* class from the others, it can also isolate the new classes from *Group 2*, but it is, however, unable to distinguish between trucks, buses and buildings, which is understandable since trucks and buses are similar and there may be buildings behind them, putting them all together in one cluster.

Fifth Test Set

Finally, the last test set, which is composed of instances belonging to the classes defined in *Group 4*. This set has 91 samples from the *Truck* class, 122 samples from the *Bus* class, 161 samples from the *Fence* class, 157 samples from the *Wall* class, 29 samples from the *Building* class, 382 samples from the *Bicycle* class, 468 samples from the *Rider* class and 188 samples from the *Motorcycle* class.

Similar to the fourth test set, only a combination of parameters is presented due the the repetition of values. Although the algorithm found the correct number of clusters, most of them contain a small portion of instances as it happens with the third cluster, only having three samples of the *Bicycle* class and one sample of the *Motorcycle* class. However, for the clusters that characterize one or more classes,

we have the first, second and sixth. Starting with the first, it is mostly made up of samples from the *Fence* class, with 82.6% of its samples, the *Wall* class, with 77.0% of its samples, and the *Rider* class, with 6.6% of its samples. The second cluster is mostly composed of samples from the *Truck* class, with 76.9% of its samples, the *Bus* class, with 81.1% of its samples, the *Building* class, with 89.7% of its samples, the *Rider* class, with 34.4% of its samples, and the *Motorcycle* class, with 33% of its samples. The sixth and last cluster is mostly comprised of samples from the class *Bicycle*, having 57.6% of them.

Thus, for this test set, the algorithm was able to isolate the class *Bicycle* again and was also aggregate the classes *Fence* and *Wall*, with some samples from the class *Rider*. It achieved a similar result on the second cluster by joining samples from the classes *Truck*, *Bus* and *Building*. In this cluster it also included samples from the classes *Rider* and *Motorcycle*.

Summary

From the obtained results, it is possible to say that, although the algorithm found the correct number of clusters for each test set, its separation fell short. Although there is a plausible reason why it cannot distinguish samples between the classes truck, bus, building and motorcycle, which is samples from different classes look similar or some elements appear behind others (such as buildings behind these vehicles), it does not justify the association of samples from the *Rider* class with these nor does it explain the inability to distinguish samples between the *Fence* and *Wall* classes.

These problems are strongly related to the problems encountered in the results obtained by OpenMax, since this step is dependent on the previous one and its F-score values are low.

Despite all this, this algorithm proves to be useful for solving problems in open-world scenarios when combined with better data processing methods, given its ability to isolate some classes and find groups of similar classes.

Chapter 7

Conclusion and Future Work

Self-driving cars are intelligent agents within a complex environment, and they must learn about all the elements they may encounter in order to take action and cause as little damage as possible. However, these environments can be constantly changing having too many different elements to learn. One option for training them is to provide a small initial amount of elements or classes, and, over time, increase this number of classes, with the agent having to learn about these new, unknown classes. The issue with many of the solutions implemented for this problem is that they assume that there is only one unknown class. Nevertheless, there is always the possibility that an agent will encounter instances belonging to several unknown classes.

The main purpose of this work was to explore an open-world solutions capable of identifying multiple unknown classes within a data set built for self-driving cars, which data set is Cityscapes [79]. To do this, since this is a data set of images, we created 4 different CNN architectures in order to find one able to correctly identify as many samples of known classes as possible. This architectures were later incorporated with the OpenMax algorithm to enable the detection of samples belonging to unknown classes. Finally, these samples were provided to the DBSCAN algorithm to identify both the number of unknown classes as well as the percentage of each class in each cluster. This percentage helps understand if the clusters created are a good approximation of each class.

The results obtained by the OpenMax algorithm show difficulty in both classifying known classes and identifying samples from unknown classes. This may have to do with a poorly treated and/or balanced data set, non-optimal CNN architectures or with OpenMax not being a good solution to handle this data set. Regardless of the reason, since the DBSCAN algorithm is dependent on the OpenMax outcome, these low results negatively impact the next stage. This can be observed when its separation fell short, despite finding the correct number of clusters for each test set. However, it still manages to isolate some instances and is able to group similar classes together.

In conclusion, despite the poor results, this methodology proves to be useful to solve this problem, but has much room for improvement.

For future work, it would be interesting to use other feature extraction method to process the data set and use its outcome as input data on DBSCAN. New CNN architectures or deeper search on the parameters to find a more optimal combination could also be explored in order to improve the the OpenMax model. A diferent data set or approach on the data set used could be implemented to improve the results from our methodology.

References

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 ed., 2010.

[2] A. I. Khan and S. Al-Habsi, "Machine learning in computer vision," *Procedia Computer Science*, vol. 167, pp. 1444–1451, 2020.

[3] L. A. Berrueta, R. M. Alonso-Salces, and K. Héberger, "Supervised pattern recognition in food analysis," *Journal of Chromatography A*, vol. 1158, pp. 196–214, July 2007.

[4] W. J. Scheirer, L. P. Jain, and T. E. Boult, "Probability models for open set recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, pp. 2317–2324, 2014.

[5] P. R. M. Júnior, R. M. de Souza, R. de Oliveira Werneck, B. V. Stein, D. V. Pazinato, W. R. de Almeida, O. A. B. Penatti, R. da Silva Torres, and A. Rocha, "Nearest neighbors distance ratio open-set classifier," *Machine Learning*, vol. 106, pp. 359–386, 2016.

[6] M. Hassen and P. K. Chan, "Learning a neural-network-based representation for open set recognition," in *Proceedings of the 2020 SIAM International Conference on Data Mining (SDM)*, pp. 154–162.

[7] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixão, F. Mutz, L. de Paula Veronese, T. Oliveira-Santos, and A. F. D. Souza, "Self-driving cars: A survey," *Expert Systems with Applications*, vol. 165, p. 113816, Mar. 2021.

[8] J. Ni, Y. Chen, Y. Chen, J. Zhu, D. Ali, and W. Cao, "A survey on theories and applications for self-driving cars based on deep learning methods," *Applied Sciences*, vol. 10, p. 2749, Apr. 2020.

[9] A. Bendale and T. Boult, "Towards open set deep networks," in *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, IEEE, 2016.

[10] L. Shu, H. Xu, and B. Liu, "Unseen class discovery in open-world classification," *ArXiv*, vol. abs/1801.05609, 2018.

[11] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, "A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955," *AI Magazine*, vol. 27, p. 12, Dec. 2006.

[12] R. Kurzweil, *The Age of Intelligent Machines*. Kurzweil Foundation, 1990.

[13] R. Bellman, *An Introduction to Artificial Intelligence: Can Computers Think?* Boyd & Fraser Publishing Company, 1978.

[14] P. Winston, *Artificial Intelligence*. Addison-Wesley series in computer science, Addison-Wesley, 1984.

[15] D. Poole, A. Mackworth, and R. Goebel, *Computational Intelligence: A Logical Approach*. USA: Oxford University Press, Inc., 1997.

[16] N. Nilsson, *Artificial Intelligence: A New Synthesis*. The Morgan Kaufmann Series in Artificial Intelligence, Elsevier Science, 1998.

[17] S. J. Russell and P. Norvig, *Artificial Intelligence: a modern approach*. Pearson, 3 ed., 2009.

[18] M. Hoy, "Alexa, siri, cortana, and more: An introduction to voice assistants," *Medical Reference Services Quarterly*, vol. 37, pp. 81–88, 01 2018.

[19] F. Isinkaye, Y. Folajimi, and B. Ojokoh, "Recommendation systems: Principles, methods and evaluation," *Egyptian Informatics Journal*, vol. 16, 08 2015.

[20] A. M. Turing, *Computing Machinery and Intelligence*, pp. 23–65. Dordrecht: Springer Netherlands, 2009.

[21] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, pp. 210–229, July 1959.

[22] A. L. Samuel, "Some studies in machine learning using the game of checkers. ii—recent progress," *IBM Journal of Research and Development*, vol. 11, no. 6, pp. 601–617, 1967.

[23] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[24] C. Bishop, *Pattern Recognition and Machine Learning*. Information Science and Statistics, Springer, 2006.

[25] B. E. Boser, I. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *COLT '92*, 1992.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[27] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Annals of Data Science*, vol. 2, no. 2, pp. 165–193, 2015.

[28] G. Pang, A. van den Hengel, C. Shen, and L. Cao, "Deep reinforcement learning for unknown anomaly detection," *arXiv preprint arXiv:2009.06847*, 2020.

[29] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep actor-critic reinforcement learning for anomaly detection," in *2019 IEEE global communications conference (GLOBECOM)*, pp. 1–6, IEEE, 2019.

[30] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[31] "Aws deepracer." Available at `https://docs.aws.amazon.com/deepracer/latest/developerguide/what-is-deepracer.html`, Accessed: 2022-08-18.

[32] J. Feng, C. Zhang, and P. Hao, "Online learning with self-organizing maps for anomaly detection in crowd scenes," in *2010 20th International Conference on Pattern Recognition*, pp. 3599–3602, IEEE, 2010.

[33] R. Laxhammar and G. Falkman, "Online learning and sequential anomaly detection in trajectories," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 6, pp. 1158–1173, 2013.

[34] S. C. Tan, K. M. Ting, and T. F. Liu, "Fast anomaly detection for streaming data," in *Twenty-second international joint conference on artificial intelligence*, 2011.

[35] K. P. Murphy, *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.]: MIT Press, 2013.

[36] B. Settles, "Active learning literature survey," Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[37] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[38] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *arXiv preprint arXiv:1901.03407*, 2019.

[39] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal processing*, vol. 99, pp. 215–249, 2014.

[40] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial intelligence review*, vol. 22, no. 2, pp. 85–126, 2004.

[41] Y. Zhang, B. Kang, B. Hooi, S. Yan, and J. Feng, "Deep long-tailed learning: A survey," *arXiv preprint arXiv:2110.04596*, 2021.

[42] C. Feng, Y. Zhong, and W. Huang, "Exploring classification equilibrium in long-tailed object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3417–3426, 2021.

[43] B. Kang, S. Xie, M. Rohrbach, Z. Yan, A. Gordo, J. Feng, and Y. Kalantidis, "Decoupling representation and classifier for long-tailed recognition," *arXiv preprint arXiv:1910.09217*, 2019.

[44] J. Tan, C. Wang, B. Li, Q. Li, W. Ouyang, C. Yin, and J. Yan, "Equalization loss for long-tailed object recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11662–11671, 2020.

[45] Z. Liu, Z. Miao, X. Zhan, J. Wang, B. Gong, and S. X. Yu, "Large-scale long-tailed recognition in an open world," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2537–2546, 2019.

[46] J. Parmar, S. S. Chouhan, and S. S. Rathore, "Open-world machine learning: Applications, challenges, and opportunities," *ArXiv*, vol. abs/2105.13448, 2021.

[47] W. Scheirer, A. Rocha, A. Sapkota, and T. Boult, "Toward open set recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, pp. 1757–72, 07 2013.

[48] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," *CalTech Report*, 03 2007.

[49] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," Tech. Rep. 07-49, University of Massachusetts, Amherst, October 2007.

[50] N. Pinto, J. J. DiCarlo, and D. D. Cox, "How far can you get with a modern face recognition test set using only simple features?," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2591–2598, 2009.

[51] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, June 2009.

[52] P. W. Frey and D. J. Slate, "Letter recognition using holland-style adaptive classifiers," *Machine Learning*, vol. 6, pp. 161–182, Mar. 1991.

[53] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[54] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 12 2014.

[55] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv 1412.6572*, 12 2014.

[56] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia*, 06 2014.

[57] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, pp. 211–252, Apr. 2015.

[58] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)*, IEEE.

[59] T. Johnston, "Language standardization and signed language dictionaries," *Sign Language Studies*, vol. 3, no. 4, pp. 431–468, 2003.

[60] J.-M. Geusebroek, G. J. Burghouts, and A. W. Smeulders, "The amsterdam library of object images," *International Journal of Computer Vision*, vol. 61, pp. 103–112, Jan. 2005.

[61] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)*, IEEE.

[62] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," *CoRR*, vol. abs/1802.10135, 2018.

[63] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *2012 IEEE Symposium on Security and Privacy*, IEEE, May 2012.

[64] G. Cohen, S. Afshar, J. C. Tapson, and A. van Schaik, "Emnist: Extending mnist to handwritten letters," *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2921–2926, 2017.

[65] M. Wilcox, S. Schuermans, C. Voskoglou, and A. Sobolevski, "State of the developer nation 12th edition - q1 2017," *Developer Nation*, 2017.

[66] "Stack overflow trends." Available at `https://insights.stackoverflow.com/trends?tags=java%2Cc%2Cc%2B%2B%2Cpython%2Cc%23%2Cvb.net%2Cjavascript%2Cassembly%2Cphp%2Cperl%2Cruby%2Cswift%2Cr%2Cobjective-c`, Accessed: 2022-08-18.

[67] "Tiobe index for august 2022." Available at `https://www.tiobe.com/tiobe-index/`, Accessed: 2022-08-18.

[68] "Pypl popularity of programming language." Available at `https://pypl.github.io/PYPL.html`, Accessed: 2022-08-18.

[69] "Python." Available at `https://www.python.org/`, Accessed: 2022-08-18.

[70] "Tensorflow." Available at `https://www.tensorflow.org/`, Accessed: 2022-08-22.

[71] "Keras: the python deep learning api." Available at `https://keras.io/`, Accessed: 2022-08-22.

[72] "Papers with code: Trends." Available at `https://paperswithcode.com/trends`, Accessed: 2022-08-22.

[73] "Pytorch." Available at `https://pytorch.org/`, Accessed: 2022-08-22.

[74] "Numpy." Available at `https://numpy.org/`, Accessed: 2022-08-18.

[75] W. J. Scheirer, A. Rocha, R. Michaels, and T. E. Boult, "Meta-recognition: The theory and practice of recognition score analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 33, pp. 1689–1695, 2011.

[76] "Matplotlib: Visualization with python." Available at `https://matplotlib.org/`, Accessed: 2022-08-18.

[77] "Scipy." Available at `https://scipy.org/`, Accessed: 2022-08-18.

[78] "Opencv." Available at `https://opencv.org/`, Accessed: 2022-08-18.

[79] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3213–3223, 2016.

[80] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, pp. 1231 – 1237, 2013.

[81] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognit. Lett.*, vol. 30, pp. 88–97, 2009.

[82] T. Scharwächter, M. Enzweiler, U. Franke, and S. Roth, "Efficient multi-cue scene segmentation," in *GCPR*, 2013.

[83] S. D. Zongyuan Ge and R. Garnavi, "Generative openmax for multi-class open set classification," in *Proceedings of the British Machine Vision Conference (BMVC)* (G. B. Tae-Kyun Kim, Stefanos Zafeiriou and K. Mikolajczyk, eds.), pp. 42.1–42.12, BMVA Press, September 2017.