A Comparative Study of Reinforcement-based and Semi-classical Learning in Sensor Fusion

Johan Bodén

Faculty of Health, Science and Technology

Master thesis in Computer Science

Second Cycle, 30 hp (ECTS)

Supervisor: Karl-Johan Grinnemo, Karlstad University

Examiner: Johan Garcia, Karlstad University

Karlstad, June 17th, 2021

Abstract

Reinforcement learning has proven itself very useful in certain areas, such as games. However, the approach has been seen as quite limited. Reinforcement-based learning has for instance not been commonly used for classification tasks as it is receiving feedback on how well it did for an action performed on a specific input. This slows the performance convergence rate as compared to other classification approaches which has the input and the corresponding output to train on. Nevertheless, this thesis aims to investigate whether reinforcement-based learning could successfully be employed on a classification task. Moreover, as sensor fusion is an expanding field which can for instance assist autonomous vehicles in understanding its surroundings, it is also interesting to see how sensor fusion, i.e., fusion between lidar and RGB images, could increase the performance in a classification task.

In this thesis, a reinforcement-based learning approach is compared to a semi-classical approach. As an example of a reinforcement learning model, a deep Q-learning network was chosen, and a support vector machine classifier built on top of a deep neural network, was chosen as an example of a semi-classical model. In this work, these frameworks are compared with and without sensor fusion to see whether fusion improves their performance.

Experiments show that the evaluated reinforcement-based learning approach underperforms in terms of metrics but mainly due to its slow learning process, in comparison to the semi-classical approach. However, on the other hand using reinforcement-based learning to carry out a classification task could still in some cases be advantageous, as it still performs fairly well in terms of the metrics presented in this work, e.g. F1-score, or for instance imbalanced datasets. As for the impact of sensor fusion, a notable improvement can be seen, e.g. when training the deep Q-learning model for 50 episodes, the F1-score increased with 0.1329; especially, when taking into account that the most of the lidar data used in the fusion is lost since this work projects the 3D lidar data onto the same 2D plane as the RGB images.

Keywords

machine learning, reinforcement learning, deep Q-learning network, classical learning, supervised learning, support vector machine, deep neural network, sensor fusion

Sammanfattning

Förstärkningsinlärning (reinforcement learning) har visat sig vara användbart i vissa områden som exempelvis spel, men tyvärr anses metoden ha vissa begränsningar. Förstärkningsbaserad inlärning har till exempel inte vanligtvis använts för klassificeringsuppgifter eftersom den får feedback beroende på hur bra en viss handling går på en viss typ av input. Detta saktar ner takten för modellen att hitta konvergensen för sin prestationsförmåga till skillnad från andra tillvägagångssätt för klassificering där dessa metoder både har input samt motsvarande utmatning för att träna på. Hur som helst, denna studie har som mål att undersöka om förstärkningsbaserad inlärning kan framgångsrikt appliceras på en klassificeringsuppgift. Utöver detta, eftersom sensorfusion är ett expanderande område vilket exempelvis kan hjälpa autonoma fordon att förstå sin omgivning, är det dessutom intressant att om sensorfusion, det vill säga fusion med lidar och RGB-bilder, kan öka prestandan för en klassificeringsuppgift.

I detta arbete jämförs ett förstärkningsbaserat tillvägagångssätt med en semiklassisk (semi-classical) metod. Som ett exempel för en förstärkningsinlärande metod, valdes ett djupt Q-inlärningsnätverk (deep Q-learning network) och en stödvektormaskin (support vector machine) byggd ovanpå ett djupt neuralt nätverk (deep neural network) som ett exempel på en semiklassisk metod. Dessa tillvägagångssätt jämförs med och utan sensorfusion för att i sin tur se om fusion förbättrar prestandan.

Experiment visar att den utvärderade förstärkningsbaserade metoden underpresterar med avseende på mätvärden men främst på grund av dess långsamma inlärningsprocess, i jämförelse med den semiklassiska metoden. Å andra sidan kan det dock i vissa fall vara fördelaktigt att applicera förstärkningsbaserad inlärning på en klassificeringsuppgift, eftersom modellen presterar dugligt med avseende på mätvärden, exempelvis F1-score; eller till exempel för applicering på obalanserade dataset. Av det som angår hur sensorfusion påverkar resultatet, en tydlig förbättring kan observeras, exempelvis när det djupa Q-inlärningsnätverket tränades för 50 episoder, upptäcktes en ökning på 0.1329 av F1-score. Att ha i åtanke är att det mesta av lidardatan som används i fusionen går förlorad eftersom detta arbete projicerar 3D lidar datan på samma 2D-plan som RGB-bilderna.

Nyckelord

maskinlärning, förstärkningsinlärning, djupt Q-inlärningsnätverk, klassisk inlärning, vägledd inlärning, stödvektormaskin, djupa neurala nätverk, sensorfusion

Acknowledgements

I would like to thank my supervisor Prasanth Sasikumar for his guidance and his time as well as TietoEVRY for making this thesis possible. I would also like to thank my supervisor at Karlstad University, Karl-Johan Grinnemo for advice and feedback throughout the thesis.

Acronyms

ML	Machine Learning
DNN	Deep Neural Network
CNN	Convolutional Neural Network
DQN	Deep Q-learning Network
SVM	Support Vector Machine
NLSPN	Non-Local Spatial Propagation Network
RPN	Region Proposal Network
RGB	Red Green Blue
ReLU	Rectified Linear Unit
mAP	mean Average Precision

Contents

Chapter 1

Introduction

TietoEVRY is a consulting company with around 24,000 employees in more than 90 countries. At TietoEVRY, telecommunications is one of the fields where they provide solutions for its customers, in which machine learning is an aspect of great interest due to its possibilities in telecommunications. In view of this, TietoEVRY wants to develop knowledge within this field as well as investigate its potential. Therefore, TietoEVRY offered during the Spring of 2021 a master thesis work, where they wanted a student to do a comparison study of reinforcement learning and semi-classical learning. As a part of the study, TietoEVRY wanted to investigate sensor fusion and its viability in areas pertinent to telecommunications.

1.1 Problem Description

Reinforcement-based learning is not commonly used for classification tasks, and this is mainly due to the fact that it is receiving feedback depending how well it did from an action performed on a specific input, which slows down the performance convergence. In other words, reinforcement learning does not need the desired outputs to specific inputs as the case with other typical typical classification algorithms [54]. Therefore, the performance of reinforcement learning for classification would be interesting to investigate. Moreover, in the automotive industry, there is currently a growing interest in sensor fusion as it could increase accuracy which in turn could lead to safer driving [32]; and this is only one application, sensor fusion can be used in numerous of other fields. As a part of this thesis, sensor fusion will be investigated to determine its possibilities on a specific classification task: Classification of fused data from RGB images and lidar. In particular, this paper tries to answer the following questions:

- In a classification task, how reinforcement learning performs in comparison to a semi-classical approach?
- Is reinforcement learning viable for classification tasks?
- Does fusion of RGB images and lidar assist in classification tasks?

To the author's best knowledge, there is no paper doing classification with reinforcement learning in sensor fusion.

1.2 Thesis Objective

The objective of this thesis is to compare the use of reinforcement-based learning as compared to a semi-classical approach for the classification of fused data from RGB images and lidar. Additionally, as a side objective, this work will result in deliverables as a proof-of-concept, as well as models which can be built upon. The aim in this paper is not necessarily trying to obtain the best performance from the models built but to do a high-level comparison.

1.3 Ethics and Sustainability

This thesis work only considers classification which only constitutes a tiny portion of machine learning's actual potential. In turn, machine learning is just a sub-field of Artificial Intelligence (AI). AI is currently receiving a significant interest in numerous fields; which has both pros and cons in regards of sustainability. In for instance a paper by van Wynsberghe [57], he argues and wants to raise awareness that AI actually has costs on the environment. For instance, the energy that is needed to train an AI model will leave a significant carbon footprint. On the other hand, AI can also be used to reduce carbon footprint. For instance, Google has developed an AI that was applied to their data centers and managed to reduce the amount of energy they use for cooling by up to 40 percent [7]. In terms of ethics, AI in autonomous vehicles is believed to lead to safer driving. However, this leads to various ethical issues. Consider a situation where an autonomous vehicle has to prioritize the safety of other users on the road; should the autonomous vehicle choose over the death of car occupants, bicyclists, pedestrians, et cetera?

1.4 Methodology

In this thesis, a comparison of reinforcement learning and a semi-classical approach is conducted. A Deep Q-learning network (DQN) [29] is chosen as an example of reinforcement learning framework. As for the semi-classical approach, a Support Vector Machine (SVM) [47] classifier is built on top of Convolutional Neural Network (CNN) [3]. Additionally, sensor fusion is investigated and how it combined with the different models can increase the performance. These networks are built with the Keras API [1]. This results in four different networks: a DQN with and without sensor fusion as well as the SVM with and without sensor fusion. As for the sensor fusion networks, RGB images and 2D-dense depth maps from lidar are used, in the networks without sensor fusion, only RGB images are used. From the RGB images and 2D-dense depth maps, proposed boxes with objects are returned and used to train the network with. This results in different models which can be tested and evaluated against each other.

1.5 Related Work

This work cover mainly two areas; deep reinforcement learning for classification and sensor fusion, hence work around these are introduced.

1.5.1 Deep Reinforcement Learning for Classification

Deep reinforcement learning has earlier shown great performance when applied to Atari games such as Breakout, Enduro and Pong where the model actually defeated expert human players [35], but its applications has been seen as limited. However, deep reinforcement learning has gotten attention in imbalanced data learning as it is possible to configure the reward function to pay more attention to the minority class by for instance penalizing the model when it has misclassified the minority class, or by giving a higher reward when it has correctly classified the minority class. The work by Lin et al. [29] propose a deep reinforcement learning model and study its performance through experiments and compare it with other methods for imbalanced classification. Lin et al. formulates the imbalanced classification problem as a sequential decision-making process where samples of data are fed into the model at each time step and rewarding the model for each sample depending on how well it did; this work formulates the classification problem in the same way. The dataset in this work is somewhat imbalanced and could optimize the reward function in the same way as the work by Lin et al., however due to time constraints this could not be covered and is instead encouraged as an avenue for future work.

1.5.2 Sensor Fusion

Sensor fusion has recently become a widely studied and growing field, where for instance lidar and RGB images has become common in autonomous driving. Sensor fusion can be done in several ways and can for instance be done in different stages, e.g., early fusion and late fusion. In turn, these concepts can be implemented with different methods. In the work by Wang et al. [51], they investigate the robustness of different sensor fusion methods when adversarial attacks are done to one or more sensors. In their work, the late fusion refers to the fusion taking place after all the convolutional layers and this work implements late fusion in the same way. But, unlike the work by Wang el al., where they feed lidar points along with RGB images into the models for feature extraction; this work constructs a dense depth map from the lidar and guidance of the RGB images which is then fed into the models along with the RGB images, as it is known that the lidar measurements are sparse and insufficient for real world applications [46].

1.6 Stakeholders

This work is done in collaboration with TietoEVRY. TietoEVRY has a growing interest in machine learning/artificial intelligence as it can help the company build solutions in for instance data communication. For instance, more than half of internet service providers expected to have implemented artificial intelligence within their network by the end of 2020. Additionally, internet service providers see the potential in maximizing the returns on networks investments, where 70% of internet service providers see the highest potential with implementation of artificial intelligence in network planning whilst 64% intend to maximize their returns by directing their efforts on network performance management [12]. Hence, TietoEVRY wants to build competence within these areas among their employees.

1.7 Delimitations

TietoEvry is primarily interested in machine learning as a way to reduce operational- and capital expenditures as well as to optimize network performance. The best approach would be to use data from one of TietoEVRY's business areas, but as this is in most cases sensitive data in regards for its customers this was not possible for this thesis. Visual recognition tasks on the other hand is a huge field; there are lots of open established datasets of images and suites for visual recognition tasks, as well as lots of related work within the area, hence images was used as it is more accessible.

1.8 Outline

This thesis is structured as follows, a review of the theory is given in Chapter 2, the DQN - and the semi-classical frameworks with and without sensor fusion are described in Chapter 3, the results including a quantitative evaluation with the models in comparison are shown in Chapter 4, and the conclusions as well as avenues for future work are presented in Chapter 5.

Chapter 2

Theory

2.1 Machine Learning

Machine Learning (ML) is considered a subfield of Artificial Intelligence which currently much effort is directed towards [47]. The intent with ML is to develop methods so that a system can learn from its experience. ML is appropriate whenever enough data is available and the task in question is too difficult to do by hand [10]. ML can be categorized into three different categories, which is essentially three different ways an algorithm interacts in order to learn the task at hand. These categories are supervised learning, unsupervised learning, and reinforcement learning. Supervised learning uses both input and the corresponding output to learn. Unsupervised learning has the advantage of learning without having available output to specific inputs, in other words being able to learn without a 'teacher'. The reinforcement learning approach learns through trial-and-error combined with a reward system [53]. Examples of ML-learning tasks are computer vision, speech recognition, and medical diagnosis.

A common task in computer vision is classification, which essentially entails being able to recognize objects and divide them into different classes. Classification together with localization is part of a concept called detection. Localization is the task of locating an object in an image and combined with classification it is possible to detect an object through localizing objects and in turn classifying them [36]. When evaluating the classification performance, a critical role in achieving an optimal model is the use of relevant metrics. Some of these are accuracy, recall, precision, and the so-called F1-score. All of these are good metrics but have different aims. Recall, precision and F1-score are class independent and focuses on the classes individually which makes a better reflection of performance on an imbalanced dataset. Accuracy on the other hand might lead to the wrong inferences being made on an imbalanced dataset because the results mostly reflect the performance of the majority class [52], i.e., a class that constitutes a large part of a dataset. Accuracy can be interpreted as the probability that the model is correct. The definition of the metrics for two classes are shown in

Equations 2.1, 2.2, 2.3 through 2.4:

$$Accuracy(A) : \frac{TP + TN}{TP + TN + FP + FN} = \frac{\textit{Number of correct predictions}}{\textit{Number of predictions}} \tag{2.1}$$

$$Recall(R) : \frac{TP}{TP + FN} \tag{2.2}$$

$$Precision(P) : \frac{TP}{TP + FP} \tag{2.3}$$

$$F1 : \frac{2 \times P \times R}{P + R} \tag{2.4}$$

where there is a positive and a negative class. The different outcomes are described below:

- TP denotes a true positive, that is when the model predicts a positive and the class is indeed positive.
- FP denotes a false positive, that is where the model predicts a positive but the class is actually negative.
- FN denotes a false negative, that is where the model predicts a negative but the class is actually positive.
- TN denotes a true negative, that is where the model predicts a negative and the class is indeed negative.

When more classes are added, and the problem becomes a multi-class problem, all classes have to be considered in the metrics. There are several ways to do this. For example, one way of doing this is to calculate an overall mean of the metrics for every individual class as well as to reformulate TP, FP, FN and TN. This is called a *macro average*. With *macro average*, the classes have the same weight in the calculation of the average, thus making no distinction between classes. Consider a problem where there is a minority and a majority class, i.e., there are two classes where one of the classes are much larger than the other one. As for *macro average*, poor performance on a minority class is reflected in the metric just as much as poor performance on a majority class. The rationale behind accuracy is still valid, and can still be seen as the number of correct predictions out of the total number of predictions. The concepts for the other metrics, when more than two classes are involved, are shown in Equations 2.5, 2.6 and 2.7 [18, 30]:

$$R_{Macro} : \frac{\sum_{k=1}^{K} R_k}{K} \tag{2.5}$$

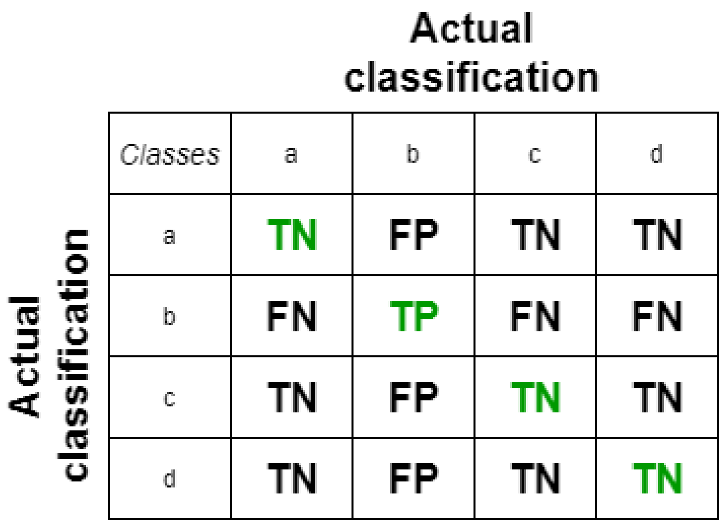
$$P_{Macro} : \frac{\sum_{k=1}^{K} P_k}{K} \tag{2.6}$$

$$F1_{Macro} : \frac{\sum_{k=1}^{K} F1_k}{K} \tag{2.7}$$

where *k* is a generic class. As mentioned, when calculating these metrics for every class individually; TP, FP, FN and TN are reformulated. Next, lets consider more than two classes. For the reformulations below, consider one class and all other classes except that one, i.e., let one class be relative at the time.

- TP is the event when the predicted class is the actual class in question.
- FP is the event when the other classes are predicted to be the class in question.
- FN is the event when the class in question is predicted to be another class.
- TN is every other event, i.e., when a correct prediction of another class is made.

This concept is visualized in Figure 2.1.1 where class *b* is the reference class.



Actual classification

Actual classification	Classes	a	b	c	d
	a	TN	FP	TN	TN
	b	FN	TP	FN	FN
	c	TN	FP	TN	TN
	d	TN	FP	TN	TN

Figure 2.1.1: Confusion matrix where class *b* is the reference class.

2.1.1 Support Vector Machine

Support Vector Machine (SVM) [34, 47] is a collection of supervised learning models used for classification and regression analysis. This collection belongs to the family of generalized linear classifiers. Supervised learning is a sub-category of classical learning. The SVM finds the highest predictive accuracy as well as automatically avoids overfitting data. Moreover, unlike other classical approaches, it is proven to be effective in cases that involve high-dimensional spaces. The goal is to separate the data with a hyperplane, but also maximizing the distance from the closest data point on each side. In other words, find the best possible line that separates classes in an arbitrary problem.

The three major concepts in an SVM are: hyperplane, support vectors, and margin. The concepts are described below:

- The hyperplane is basically a line that separates different classes.
- Support vectors are the data points which are closest to the hyperplane, the hyperplane is defined from these points. These support vectors is the reason of the name support vector machine.
- The margin is the distance to the closest data points in the different classes.

A demonstration of a simple classification problem with two classes as well as the concepts is shown in Figure 2.1.2.

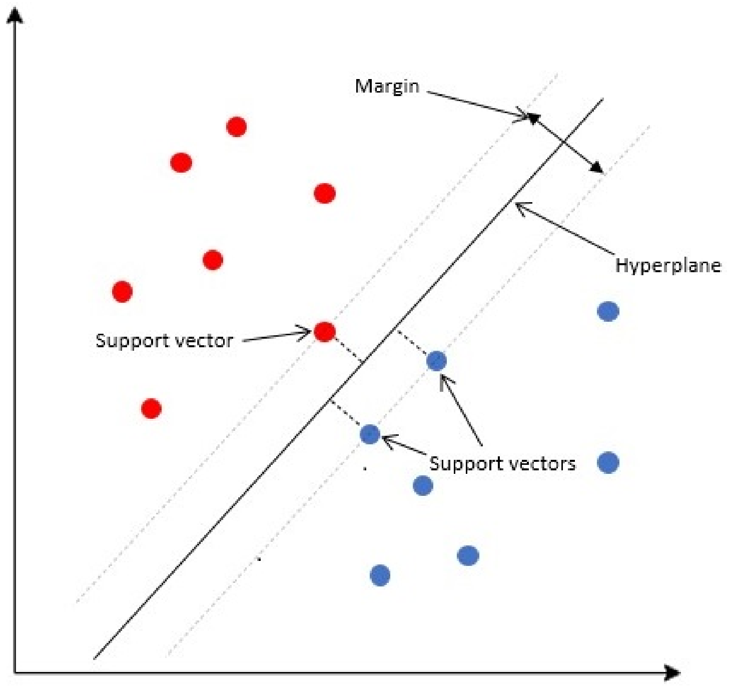


Figure 2.1.2: Demonstration of an SVM with samples from two classes.

To explain further, SVM is a kernel method [21]. Kernel transforms input data into the required space. Kernel methods use something called a *Kernel trick* which takes the original input space and computes a dot product operation between all datapoints of a higher dimensional space without actually transforming the data into that high dimensional space, this avoids the computational cost to compute every new coordinate in the new higher dimension [27]. Consider a problem which is not separable in its original space, the kernel in that case transforms the data in a higher dimension where the data is actually separable.

There are different kinds of kernels; what kind of kernel to use depends on the dataset and how its best transformed into a separable space. An example of a kernel is the polynomial kernel, which is shown in Equation 2.8. The details of how an SVM works is quite complex and is out of the scope of this work, the reader is encouraged to read the book by Koutroumbas and Theodoridis for more details [27].

$$k(x, x') = \langle x, x' \rangle^d \tag{2.8}$$

Where $\langle x, x' \rangle$ denotes the dot product, d is degree of polynomial, and x and x' are two vectors.

A simple classification problem in 1D which is transformed into 2D space using a polynomial kernel with d equal to 2 is shown in Figure 2.1.3. In Figure 2.1.3, it is seen that data is separable in a higher dimension.

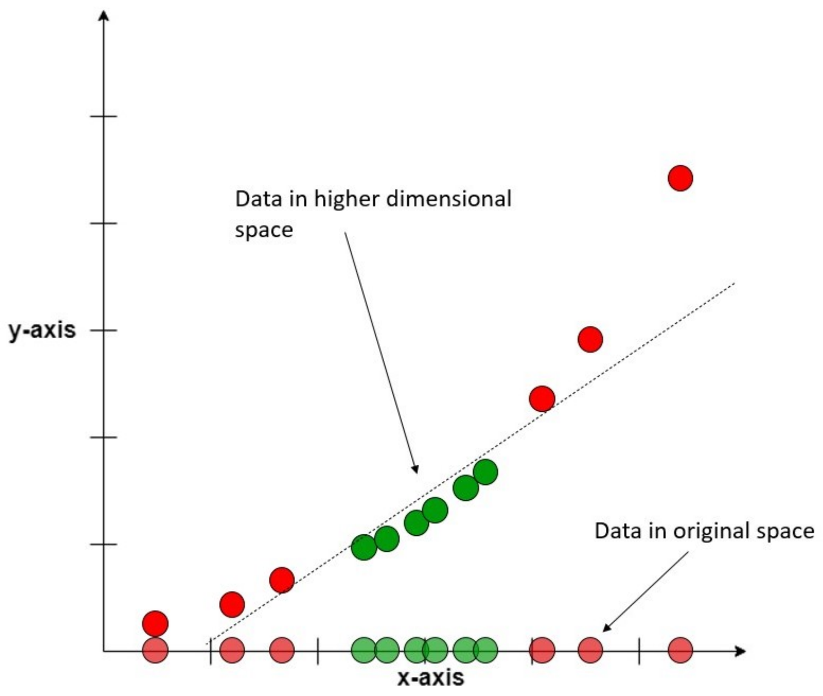


Figure 2.1.3: A 1D classification problem transformed into 2D in order to make the data separable.

2.1.2 Deep Q-learning Network

A Deep Q-learning network (DQN) [14, 29, 35] is one of the most representative reinforcement learning algorithms which uses a deep neural network to learn different tasks. A deep neural network is an efficient network to solve complex ML tasks as well as handle large amounts of data; more details on a deep neural network is found in Section 2.2. The deep neural network in a DQN can be seen as an agent in a DQN. In reinforcement learning, the agent observes a state s, at a discrete time t, and performs an action a; from that action, in that specific state, a reward r is returned. As a reinforcement learning interacts through an environment, new states occur which the agent performs an action on, every transition to a new state is called an episode. Similar to a regular reinforcement learning algorithm, the goal is to maximize future rewards by trying to find the optimal policy π in an unknown environment, which is considered to be a greedy optimal policy. Greedy in this context means that the agent will always take the action that result in the maximum current reward, which might not necessarily be the best action in the long run [19]. How the reward should be defined is up to the user, and depends on the task at hand. For instance in a classification task, a correct classification of a class could result in reward equal to +1, and a misclassification of that class could receive a reward of -1. Again, this is only an example. The policy π denotes the action that the agent performs in a specific state.

In reinforcement learning, there is a state-action value function called the Q-function, $Q^{\pi}(s, a)$, which is the maximum expected return from taking the action a in a state; this maximum expected return is denoted as E_{π}. For every new step in time, it is assumed that the future rewards are discounted by a factor of γ, γ can be interpreted as a balance of the future reward. The denotation γ is called the discount factor and is a value between zero and one. The Q-function follows an important equation called the Bellman equation [35]. With regards to the Q-function, the Bellman equation can be described as follows: If the optimal value $Q^{\pi}(s_{t+1}, a_{t+1})$ is known for all possible actions in the next state, then according to the Bellman equation, the optimal strategy for the next state is to select the action that maximizes the expected value of $r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1})$. If the Bellman equation [35] is incorporated into this equation, the Q-function can be expressed as shown in Equation 2.9.

$$Q^{\pi}(s, a) = E_{\pi}[r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1})|s_t = s, a_t = a] \tag{2.9}$$

The agent will find the maximized future rewards by solving Equation 2.9 and the greedy optimal policy. One way of setting this greedy optimal policy is to use an ϵ-greedy policy. The ϵ-optimal policy is shown in Equation 2.10.

$$\pi(a|s) = \begin{cases} a = \text{arg max}_a \, Q(s, a), & \text{with probability 1-}\epsilon \\ \text{random } a, & \text{with probability } \epsilon \end{cases} \tag{2.10}$$

To find the optimal Q-function, Equation 2.10 can be substituted into Equation 2.9, which results in Equation 2.11.

$$Q^{\pi}(s, a) = E_{\pi}[r_t + \gamma \, \text{max}_a \, Q(s_{t+1}, a_{t+1})|s_t = s, a_t = a] \tag{2.11}$$

ϵ in Equation 2.10 is a value between zero and one, which essentially is the probability of taking a random action or 1-ϵ, the action that results in the current maximum reward. For Equation 2.10, the first condition can be seen as *exploitation* and the second one *exploration*, a big challenge is the trade-off between these. *Exploration* is in this case a random action that might lead to the agent learning new things about the environment but not necessarily the maximized reward, which is the primary goal for the agent. *Exploitation*, entails performing an action based on past experience, i.e., taking the action that results in the current maximum reward. Nevertheless, too much exploitation might lead to the agent not exploring the environment enough, thus possibly leading to the agent missing out on the optimal solution [31].

Unlike Q-learning where the Q-functions are recorded by a table, DQN uses a deep neural network to record its Q-functions. Moreover, DQN uses an experience replay memory, which is basically a buffer in which the data gathered from Equation 2.11 is stored. The data stored is the current state (s), the action (a) taken from that state, the reward (r) from that action in that specific state as well as the next state (s'), i.e., (s, a, r, s').

Apart from the Q-function, DQN uses a so-called loss function which should be minimized and which determines the quality of the prediction. One way to define the loss function is shown in Equation 2.12 [6], which is equal to a general mean square error-loss function, i.e., $L(y, y') = \frac{1}{n}\sum_{i=0}^{n}(y - y_i')^2$ [33], where y and y' are two values and n the total number of terms for which the error is calculated. The agent selects a number of samples from the experience replay memory called a mini-batch. With the mini-batch, the agent minimizes the loss function by performing a so-called gradient descent step [41] on the deep neural network according to the loss function previously mentioned. Gradient descent is one of the most popular algorithms for optimization, and the most common way to optimize neural networks [41].

$$L(\theta) = \frac{1}{n}\sum_{(s,a,r,s')\in B}(y - Q(s, a; \theta))^2 \tag{2.12}$$

Where B denotes the mini-batch of samples that was selected from the experience replay memory, n denotes the number of samples in the mini-batch, θ denotes the weights of the deep neural network, and y, which is a target estimate of the Q-function in Equation 2.11. The target estimate can be expressed as shown in Equation 2.13.

$$y = \begin{cases} r, & terminal = True \\ r + \gamma \max_{a'} Q(s', a'; \theta^-), & terminal = False \end{cases} \tag{2.13}$$

The boolean variable *terminal* in Equation 2.13 is a flag that signals that the agent has finished the task. The definition of a finished task varies and depends on the task itself. A finished task could for instance be a classification task where the agent has gone through the entire dataset. θ^- denotes the weights of the target network, these weights are kept fixed. Additionally, a' is the action that is performed in state s'. Consequently, by minimizing the loss function in Equation 2.12, and performing the greedy optimal policy in Equation 2.10, the agent will be able to find the optimal Q-function in Equation 2.11 which maximizes future rewards.

2.2 Deep Neural Network

Deep Neural Networks (DNNs) [3, 42] are a growing field within ML which has shown to be very effective in solving complex ML tasks such as object detection and speech recognition, it is also able to handle large amounts of data . DNNs consist of a number of simulated neurons in layers which are meant to model the human brain. Every neuron is viewed as a node, and between every node there is a link. Every link has a weight that determines the influence on another node. A large weight corresponds to a strong influence while a small weight corresponds to weak influence [55]. The DNN is not transparent, and the reason why the DNN makes a certain decision is difficult to understand. One of the most popular DNNs is Convolutional Neural Networks (CNNs), CNN is highly effective for tasks involving images. A CNN consists of three types of layers: convolutional layers, pooling layers, and fully connected layers. A typical setup, is blocks of these convolutional and pooling layers which lastly feed into fully connected layer; a visualization of this structure is shown in Figure 2.2.1.

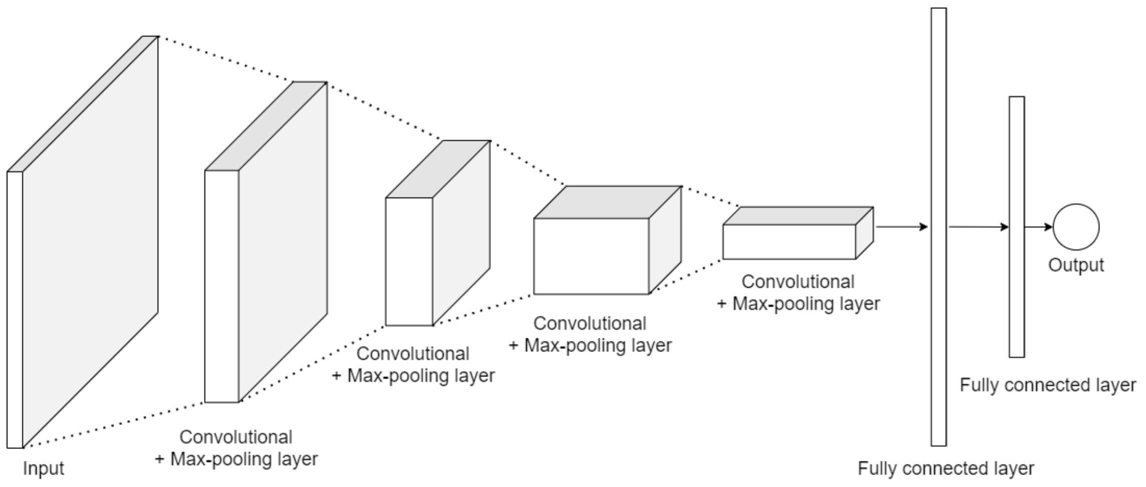

Input
Convolutional + Max-pooling layer
Convolutional + Max-pooling layer
Convolutional + Max-pooling layer
Convolutional + Max-pooling layer
Fully connected layer
Fully connected layer
Output

Figure 2.2.1: A mock-up CNN where blocks of convolutional and pooling layers are built in blocks which lastly feeds into fully connected layers.

In a convolutional layer, a kernel, i.e., a filter matrix, is applied over the input. An element-by-element product operation is performed between the input and the kernel, the result is then returned as the output from the convolutional layer. The goal is to find the kernel which works best for a specific task. In turn, the output is passed through a non-linear activation function, the most common one is the Rectified Linear Unit (ReLU) [58]. The ReLU activation function is simple and behaves as an identity function provided its input value is above zero; when the input value is less than zero, the function return zero, see Equation 2.14.

$$ReLU(x) = \begin{cases} x, & if x > 0 \\ 0, & otherwise \end{cases} \tag{2.14}$$

The pooling layer helps reducing the complexity of subsequent layers by down-sampling the input. The most commonly employed type of pooling layer is the so-called max-pooling layer. In max-pooling, the input is divided into smaller sections. From those smaller sections, the maximum value is returned in a reduced dimension as output. The most common settings for max pooling is a kernel size of 2×2 and stride of size 2. Stride is the step size the kernel takes when moving the kernel. This concept is visualized in Figure 2.2.2, in which a kernel size of 2×2 and a stride 2 is used.

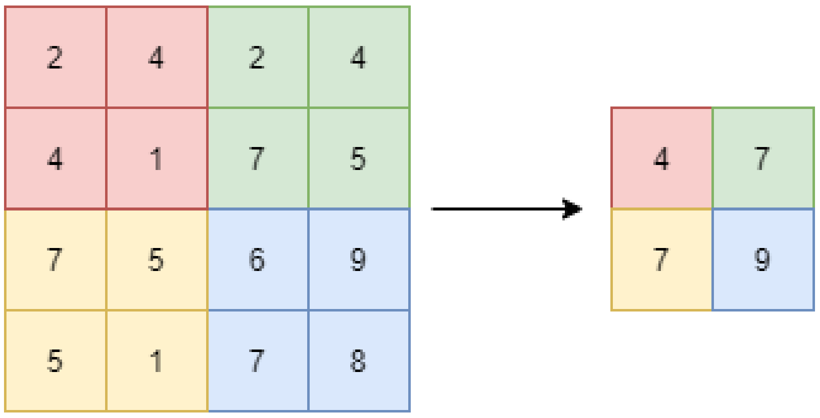


Figure 2.2.2: Visualization of a max-pooling with a kernel size 2×2 and stride a size of 2 is used.

Consequently, the output of a convolutional layer or a pooling layer, is commonly flattened which basically means transforming the input to a 1D array. The flattened output will in turn be fed to the fully connected layer, also called a dense layer, which connects every output from the flattened layer to the number of outputs. For example, in a classification task, the number of outputs are typically equivalent to the number of classes to classify. Every fully connected layer is typically followed by a non-linear function, for instance ReLU. As for the last fully connected layer, a so-called softmax function can be applied to the output. A softmax function is an activation function which normalizes the output values from the last fully connected layer to probabilities. In a classification problem this means the probability of belonging to a certain class.

A common problem with ML algorithms is overfitting. This basically means that a model performs good on training data but not so well on test data. Two ways to prevent this in a CNN is with *dropout* and *batch normalization* [15]. Dropout does this by randomly removing nodes from the network, and batch normalization by normalizing the output from a specific layer before it is fed into another one.

2.2.1 Keras

Keras is a deep learning API, written in Python and built on top of the open-source ML library, TensorFlow [1]. The CNN can be built with the Model class in the Keras API [24]. The model can be compiled with the *compile* method [22] and in turn be trained with the *fit* method [23] and evaluated with the *predict* method [25] in the Keras API. In the compile method, out of many other arguments, an optimizer, for instance the Adam algorithm [2], and a loss function can be configured. In the optimizer, a learning rate can be configured. The learning rate defines the amount with which the weights in the CNN are changed at each time the CNN is updated. A large learning rate will cause the agent to learn fast, but if it set too high it might lead to the agent taking too large steps causing the agent to jump past a optimum and thus lead to the agent never learning. An agent with a small learning rate will not act like this, but will on the other hand learn slower. Hence, a balance has to be found between a not too small but neither a not too high learning rate, in order to find a good learning rate for a specific task [38]; this problem is visualized in Figure 2.2.3.

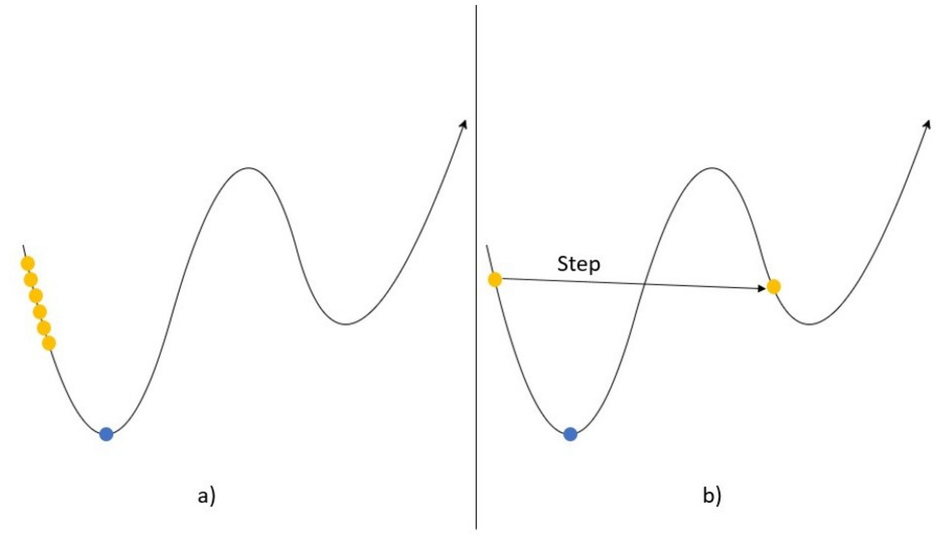


Figure 2.2.3: Balance of learning rate visualized. a) Shows a too low learning rate causing the agent to learn very slow. b) Shows a too high learning rate causing the agent to jump past the optimum.

Examples of loss functions which can be configured in the compile method are mean square error and hinge loss. The mean square error equals the loss function used in DQN [14]. The hinge loss can together with a random Fourier features layer be approximated to a SVM [39]. The idea with random Fourier features is to transform the original space to a fixed dimensional space which is of a lower dimension, but with the resulting dot product which approximates to the kernel $k(x, x')$, where x and x' are two vectors. For more details on random Fourier features, see paper by Li et al. [28].

In the *fit* method there is an argument called *epochs*, which denotes the number of iterations which the model goes over the entire dataset to train the model [23]. Keras also provides a method which instantiates a VGG16 model [50]. VGG16 is a CNN structure proposed by Simonyan and Zisserman [44], which is proven to be very effective in visual recognition tasks. In the ”ImageNet large scale visual recognition Challenge” in 2014 it won the 1st place in object localization and 2nd place in classification [49]. The network consists of 13 convolutional layers and 3 fully-connected layers, and is the reason for the suffix 16 in the name of the model. The VGG16 is the most common used network from the networks that was introduced in the paper by Simonyan and Zisserman [20].

2.3 Region Proposal Network

A part of image detection is as mentioned in Section 2.1 to localize where objects actually are before classifying them. This is where a Region Proposal Network (RPN) comes in. An RPN is a network that extract boxes from images where it is likely to be an object in, these are called RPN boxes. A simple way to this is to have a window slide over the images and have an another network check if the box in question actually has an object in that box. Through research it has been shown that it is better to do this through a neural network [20].

2.4 Detectron2

Detectron2 [56] is the next-generation library of Facebook AI Research which provides algorithms for recognition tasks such as object detection. Detectron2 also provides a collection of baseline results as well as openly accessible, trained models. The results from this collection are ranked on the basis of the mean average precision (mAP) to determine the performance of the models. Detectron2 uses COCO's metrics [9], which is another suite for recognition tasks [5]. COCO makes no distinction between AP and mAP, hence the AP value in the baseline results in Detectron2 are actually mAP. Details of the definition of mAP can be found in the paper by Everingham et al. [13] as well as the specific settings COCO uses [8]. Note, this metric is not calculated by taking the average of the precision values.

2.5 Sensor Fusion

Sensor fusion is a concept where data from several information sources are used in combination in order to make a decision about a specific task or situation [11]. It can be used in autonomous driving to, e.g., combine lidar, RGB camera images, and 3D radar data. Lidar and 3D radar are techniques to obtain information about an environment and can for instance be used to map landscapes and buildings. Lidar uses light to obtain such information while 3D radar uses radio waves [4]. The result of doing sensor fusion estimates a better understanding of a state compared to using different sensors individually. In turn, sensor fusion may result in improved reliability, improved accuracy, improved availability and higher safety for a specific framework [32]. There are numerous ways to do sensor fusion and much effort is put into developing tools to carry out sensor fusion. A common way to implement sensor fusion is with neural networks. One way to this is through late fusion, which also has several methods. Late fusion can for instance be done after all the convolutional layers, another way in object detection is fusion after boxes has been proposed from the RPN [51].

2.6 KITTI

KITTI [16] is a benchmark suite for tasks such as 3D-object detection in autonomous driving. KITTI provides data from high resolution video cameras, Velodyne laser scanners (lidar) and a state-of-the-art localization systems. KITTI also provides benchmarks as well as evaluation metrics for them[1], e.g., it provides a benchmark suite to increase density from sparse point clouds from a lidar, i.e., producing dense depth maps from sparse lidar data with guidance from RGB images. This is desired as the sparse data is insufficient for real world applications [48].

[1] The benchmarks are available at *www.cvlibs.net/datasets/kitti*

Chapter 3

Methodology

Two main frameworks are set up: one reinforcement learning and one semi-classical framework. DQN is chosen as an example of a reinforcement-learning framework. As for the semi-classical approach, an SVM classifier is built on top of a CNN. In fact, the CNN is essential for both frameworks. This is where the sensor fusion and the feature extraction are done. Both frameworks are fed with images, with the intent of being able to classify them.

3.1 Environment

The environment is a series of boxes of different objects gathered from images of different traffic scenarios. One traffic scenario might have several objects such as persons and cars leading to one traffic scenario having several boxes. The different traffic scenarios provide RGB images and raw lidar scans. However, these lidar scans can only obtain sparse measurements of the environment regardless of the quality of the lidar equipment. These sparse scans are insufficient for real world applications [46]. Therefore, estimating a dense perception of the environment is needed to produce good results for classification. This dense perception is called a dense depth map. The data in question is gathered from the KITTI dataset [16]. KITTI claims that there are up to 30 pedestrians and 15 cars visible per image, meaning that 45 different boxes could possibly be gathered from one scenario. KITTI also provides a benchmark suite with different machine learning vision challenges, one of them is depth completion. Depth completion is used to get the dense depth map needed for the classification. For simplicity, results from the best scoring algorithm are used. The results from these algorithms are 2D-dense depth maps. In this case, as for the KITTI completion evaluation, the Non-Local Spatial Propagation Network (NLSPN) is chosen. The results for NLPSN are the best scoring as for the metrics that KITTI benchmark suite evaluates but who also has results that are openly accessible. See KITTI completion evaluation page [26] and the page for the NLSPN network for more details [37]. In total, with the results from the NLPSN algorithm leads to 1000 RGB images and 2D-dense depth maps which can be used for classification. Once again, one image can contain several objects to classify.

To get the boxes, an algorithm from the Detectron2 [56] library is used. Detectron2 provides a large set of baseline results and pre-trained models, open to download and to try out. Basically, the algorithm with the best score for boxes is used (see Section 2.4 for details). This happened to be X101-FPN under the COCO instance segmentation baselines results with the Mask R-CNN method [56]. The algorithm is in turn run on the RGB images from the different traffic scenarios. With the results from the X101-FPN, boxes and corresponding labels could be gathered. To get as clear images of objects as possible, the threshold of the predictions from the X101-FPN is set to be 0.95. In other words, only objects where the algorithm provided an accuracy of more than 95% in its predictions will be returned to the SVM or the DQN model. Finally, coordinates for the boxes that are returned from X101-FPN is used to cut out the corresponding box from the depth map. When sensor fusion is implemented, this lead to the model being fed with boxes from RGB- and 2D-dense depth maps images. The 2D-dense depth maps are actually grayscale but are read as RGB. The different classes that are classified are: cars, persons, trucks, and bicycles. Hence, the returned boxes are also from these classes. The 1000 RGB images and 2D-dense depth maps are divided into 80% for the model train on and 20% for testing. The distribution of the different classes can be seen in Table 3.1.1. From Table 3.1.1, it can be derived that the dataset is fairly imbalanced.

Table 3.1.1: Table representing the distribution of objects in the dataset.

	Cars	Persons	Trucks	Bicycles
Train	2230	406	59	89
Testing	507	127	11	21
Total	2737	533	70	110

3.2 Convolutional Neural Network

The CNNs look different depending on their ultimate objective. The networks are built as similar as possible but are different depending on whether sensor fusion should be employed, and depending on the unique top layers for the SVM and the DQN models, which results in four different structures for the CNNs. The different CNN structures are essentially built on a reduced VGG16 network with a number of layers built on top of it: a flattening layer, a fully connected layer, a batch normalization layer, a ReLU activation layer and a dropout layer. Henceforth, let us call these layers the ”top structure”. As for all networks, a dense layer is used on the top to combine the outputs of the neurons with the number of available classes. For instance, if the task is to classify cars and persons, the very last dense layer has two outputs. The VGG16 networks are regular VGG16 networks, except that the four top layers are excluded, i.e., a reduced VGG16 network. Also the weights are set to random, i.e., the VGG16 networks are not pre-trained. The input shape of the images are (50,50,3), in other words 50x50 RGB images. The reason for a specific structure and settings in a network is to a certain extent empirical, for instance the number of neurons in a layer. Therefore, the final versions of these networks might not be the optimal ones. In this thesis, the Model class from the Keras API is used to build the CNN networks. The *fit* and *predict* method in the Keras API are used to train and test the model (see Section 2.2.1 for details). At a fundamental level, for a network with sensor fusion, two reduced VGG16 networks are used as inputs, one for RGB images and one for 2D-dense depth maps. As for a network without sensor fusion, only one reduced VGG16 network is used for the input of RGB images.

3.2.1 DQN-specific Settings

Without sensor fusion, the structure of the DQN is essentially a reduced VGG16 with a ”top structure” on top of it as well as the dense layer with a softmax activation. The CNN for the DQN, when sensor fusion is employed, is roughly the same except that two reduced VGG16 networks are used, these are in turn concatenated. The two reduced VGG16 networks create two input channels where the RGB images and the 2D-dense depth maps can be fed respectively. On top of the concatenation, the ”top structure” is built as well as the dense layer with the softmax activation layer on top of it. While building the CNN for the DQN, without this softmax activation layer, the agent could not continuously learn. The hypothesis is that the learning could be too fast causing the agent to jump out of an optimum, similar to a learning rate. Since the softmax activation layer normalizes the output between zero and one, causing the network to learn slower but more stable. However, this is not proven, it is in fact only a hypothesis, and the proof goes beyond the scope of this thesis. An Adam optimizer is used to optimize the parameters in the CNN as well as a mean square error loss function as shown in Equation 2.12. The learning rate for the Adam optimizer is set to 1E-6. The batch size is 64. Figure A.0.1 illustrates a CNN with sensor fusion and Figure A.0.2 a CNN without sensor fusion.

3.2.2 SVM-specific Settings

As for the SVM, the structure is essentially the same as the DQN network except the top of the network as well as the loss function. On top of the ”top structure”, a random Fourier features layer and a regular dense layer are placed. The loss function is a hinge loss with the Adam optimizer set to a learning rate of 1E-6. The batch size is 64 samples. Similar to the DQN, the difference between a network with sensor fusion and one without sensor fusion is the number of VGG16 networks (channels for the number of sensors) as well as the concatenation of them. Figure A.0.3 illustrates a CNN with sensor fusion and Figure A.0.4 a CNN without sensor fusion.

3.3 Deep Q-learning Network

How the DQN is set up and the reason for its design is influenced by Lin et al. and their paper [29] . As for the code and structuring of it, is done with help of a tutorial [17]. The states of the environment are determined by the training samples. A box with an object corresponds to the state s_t which the model is in at a specific time, t. Every new box is considered to be a step in time, i.e., $s_t \rightarrow \mathrm{s}_{t+1}$ is a step in time where the model goes from one box to another. An episode ends when the model misclassifies a minority class, or when the model has gone through the entire training data. A misclassification can be done by randomly choosing the wrong class or using the *predict* method from the Keras API, i.e, when the model predicts an incorrect class. A minority class is either a ”person, a ”truck”, or a ”bicycle”. Therefore when more classes are added to the model for classification, the episodes are in general shorter since there are more minority classes. When an episode ends, the order of the training samples is shuffled. The hypothesis is that this will avoid that the model is learning a pattern in the ordering instead of actually learning features about different classes.

As for specific settings: The reward function is simply assigned a value of +1 for a correct classification and a -1 for an incorrect classification, regardless of the class. Initially, a reward function similar to the one employed by the paper by Lin et al. [29] was implemented, but the metrics studied in this work did not improve for the dataset that was used, and thus, ending up with the simple reward function just mentioned. However, in hindsight, the reward function should have been optimized further. In this thesis, the ϵ is set to one with an ϵ-decay of 0.9999. That is, after every state, ϵ is decreased by 0.0001%. This causes the agent to make a lot of exploration of the environment initially and more exploration as the ϵ decreases. As mentioned, an episode ends when the model misclassifies a minority class. Hence, as ϵ decreases and more exploitation is done, the episodes will become longer over time. See Section 2.1.2 and Equation 2.10 for details regarding this concept. Before a training run is started, the model initializes the experience replay memory to a minimum number of samples. This minimum is set to 1000 samples where the max length of the experience replay memory is 2000 samples. In Figure 3.3.1, a simplified visualization of the DQN network when using sensor fusion is shown. The network is similar when not using sensor fusion except that the network does not take the 2D-dense depth maps into account. Hence, only RGB images are fed into the agent and consists only of one channel throughout the entire CNN. When the the framework was being built, a graph similar to Figure B.0.1 was used to debug and determine if the algorithm was actually working.

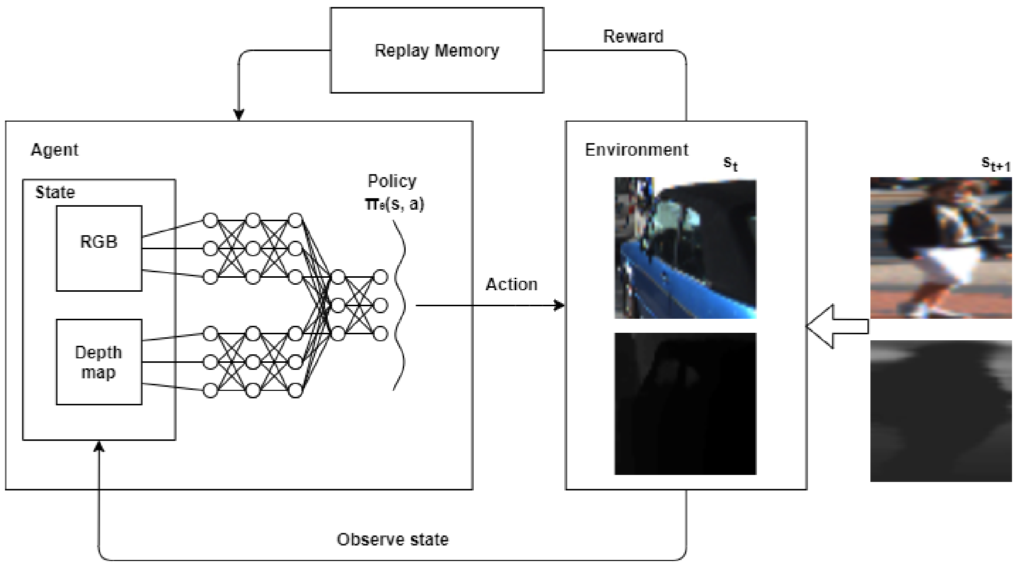


Figure 3.3.1: A conceptual visualisation of a DQN network when sensor fusion is employed.

3.4 Semi-classical Approach

The semi-classical approach is essentially a CNN network with an SVM approximation on top of it. The CNN works as a feature extractor and the SVM approximation as the classifier. The composition of a random Fourier feature layer combined with a hinge-loss function are what essentially builds the SVM. The combination of these approximates to an SVM [39]. By building the SVM on top of the CNN the network is able to train end-to-end. In general, the SVM model works as a regular supervised model. All the training data is fed into Keras *fit* method with corresponding labels to train against. The argument epochs is used to train the semi-classical approach over a number of epochs. For evaluation, the model is fed with testing data using the *predict* method, which in turn is checked against the actual labels for results. Throughout this paper, this model will be referred to as the SVM model.

Chapter 4

Evaluation and Result

The main objective is to compare reinforcement-based learning, against a semi-classical approach which uses a CNN feature extractor and an SVM as a classifier. Moreover, to explore whether reinforcement learning is a viable option for classification of objects. Additionally, as a part of this work, investigate whether sensor fusion could assist the classification in the studied models. This thesis is not necessarily trying to obtain the highest result, instead it aims to conduct a comparison and investigate the viability for the different concepts. In this thesis, a DQN network is chosen as a reinforcement-learning model.

The metrics considered includes accuracy, precision, recall, and F1-score. The reason for the latter three metrics is due to the imbalance that is present in the considered dataset (see Table 3.1.1). Moreover, these metrics are also an average over all classes, thus penalize the result in those cases the model scores worse on a minority class. The rationale behind these metrics is to be able to tell if the models have flaws classifying these minority classes. Once again, since the dataset is imbalanced; precision, recall, and F1-score will be the most interesting metrics. Accuracy is included as it still might be interesting, to for instance disregard that the dataset is imbalanced and see the overall probability that the model predicts the correct class. Keep in mind that F1-score gives equal weight to precision and recall. In this paper, the attention will be put on F1-score to give a quick high-level comparison. When actually studying the performance of a run from a specific model, the reader is encouraged to look at precision and recall separately. Throughout this section, the metrics presented in the tables are all averages over the metric in question. The averages are gathered from the scikit-learn API using the *classification_report* method [45]. Scikit-learn is an API which provides tools for predictive data analysis [43].

In summary, the SVM is generally faster, with less training, in achieving higher results in terms of the studied metrics, in comparison to the DQN model. On the other hand, the best performing DQN is very close to the performance of the best performing SVM and the difference is likely within the margin of error. To make a fair comparison against the results that are presented in this thesis, the versions and architecture of the system are shown in Table 4.0.1.

Table 4.0.1: Evaluation platform.

GPU	CUDA Toolkit	cuDNN	PyTorch	TensorFlow	Detectron2
GeForce GTX 1060 3GB	v11.1.1	v8.1.1	v1.8.1	v2.4.1	v0.4

4.1 DQN vs SVM

Four different models are evaluated: DQN with and without sensor fusion as well as SVM with and without sensor fusion. Every model is evaluated in three different use cases, with the number of studied classes ranging between 2- 4 classes. The tables in the remainder of this chapter are shown with the corresponding amount of epochs and episodes. To get a relatively high spread of epochs/episodes; the models are evaluated for 50-3200 epochs/episodes, which increases with a multiplication of two, i.e., 50,100,200,400... and so on up to 3200 epochs/episodes. However, with an exception of the DQN frameworks, those models are not run for more than 1600 episodes. This seems to also be an adequate number of epochs for the SVM as the performance has more or less converged at that time. Still, it should be emphasized that this does not prove that an actual convergence has actually been found.

4.1.1 Using Sensor Fusion

When sensor fusion is used, it can be concluded that the SVM model outperforms the DQN model in most cases. The SVM model scores generally a higher F1-score and a higher score for accuracy, recall and precision. Additionally, the DQN model is significantly slower to train than the SVM model.

Table 4.1.1, shows the results from the experiments where cars and persons are classified. It follows that the highest F1-score in the two-class experiments are obtained after 400 episodes or $\sim$1.62 hours for the DQN and 100 epochs or $\sim$0.22 hours for the SVM. Although DQN scored higher in its best run, the DQN takes more time to train. Next, lets consider the three-class experiments, i.e., the experiments where cars, persons, and trucks are classified. It follows from Table 4.1.2 that the run with highest F1-score is obtained after 1600 episodes or $\sim$56.20 hours for DQN and 3200 epochs or $\sim$7.47 hours for the SVM. In regards of F1-score, the SVM scores higher with 0.0065 compared to the DQN even though it is trained for significantly shorter time.

Table 4.1.1: The two different methods using sensor fusion, for n number of epochs/episodes; 2 classes.

n Model	50	100	200	400	800	1600	3200
DQN							
Accuracy	0.9842	0.9905	0.9905	0.9984	0.9905	x	x
Precision	0.9657	0.9825	0.9852	0.9990	0.9852	x	x
Recall	0.9872	0.9882	0.9852	0.9961	0.9852	x	x
F1-score	0.9759	0.9853	0.9852	0.9975	0.9852	x	x
SVM							
Accuracy	0.9890	0.9968	0.9953	0.9937	0.9953	0.9953	0.9937
Precision	0.9788	0.9951	0.9912	0.9902	0.9912	0.9971	0.9847
Recall	0.9872	0.9951	0.9941	0.9902	0.9941	0.9882	0.9961
F1-score	0.9829	0.9951	0.9926	0.9902	0.9926	0.9925	0.9903

Table 4.1.2: The two different methods using sensor fusion, for n number of epochs/episodes; 3 classes.

n Model	50	100	200	400	800	1600	3200
DQN							
Accuracy	0.8899	0.9457	0.9752	0.9767	0.9907	0.9891	x
Precision	0.6412	0.7742	0.8825	0.9823	0.9614	0.9877	x
Recall	0.7656	0.7102	0.8077	0.8024	0.9309	0.9618	x
F1-score	0.6806	0.7151	0.8353	0.8583	0.9455	0.9740	x
SVM							
Accuracy	0.9752	0.9845	0.9891	0.9907	0.9922	0.9876	0.9953
Precision	0.9782	0.9858	0.9915	0.9903	0.9909	0.9890	0.9942
Recall	0.7484	0.8432	0.8748	0.9348	0.9651	0.8742	0.9684
F1-score	0.7939	0.8922	0.9197	0.9595	0.9773	0.9181	0.9805

Finally, lets consider the four-class experiments, i.e., the experiments that includes all class of objects, cars, persons, trucks, and bicycles. The results are seen in Table 4.1.3. The best results, i.e., the experiments with the highest F1-score, are obtained after 1600 episodes or $\sim$35.33 hours for the DQN and 3200 epochs or $\sim$6.74 hours for the SVM. Once again, as for the highest F1-score, SVM scores higher; additionally it takes less time to train.

Table 4.1.3: The two different methods using sensor fusion, for n number of epochs/episodes; 4 classes.

Model \ n	50	100	200	400	800	1600	3200
DQN							
Accuracy	0.7958	0.9219	0.9459	0.9625	0.9820	0.9865	x
Precision	0.4704	0.7274	0.8788	0.8661	0.9639	0.9884	x
Recall	0.5374	0.6830	0.7204	0.8580	0.9145	0.9476	x
F1-score	0.4823	0.6843	0.7728	0.8615	0.9336	0.9668	x
SVM							
Accuracy	0.9565	0.9805	0.9850	0.9895	0.9880	0.9895	0.9925
Precision	0.8500	0.9472	0.9908	0.9808	0.9904	0.9908	0.9933
Recall	0.6413	0.8467	0.8596	0.9377	0.9065	0.9392	0.9525
F1-score	0.6957	0.8801	0.9079	0.9570	0.9424	0.9626	0.9717

4.1.2 Without Sensor Fusion

Also without sensor fusion, the SVM model does better than DQN model in general. And this holds both in terms of the F1-score as well as for the time the time taken for the models to train. However, for the best performing models for 3 and 4 classes, the DQN outperforms the SVM in terms of the F1-score; although a conclusion cannot be drawn, this could show tendency that the DQN performs better with an imbalanced dataset which for instance the paper by Lin et al. mentions is an advantage with reinforcement learning in imbalanced data learning [29], this work does not disprove this. Additionally, comparing the best performing models when sensor fusion is employed and when its not, the DQN show a tendency of having issues with the fusion of the data, however this conclusion cannot be drawn as there are such small differences.

As seen in Table 4.1.4, the best results in the two-class experiments are achieved after 400 episodes or $\sim$0.87 hours for the DQN and 100 epochs or $\sim$0.11 hours for the SVM model. The DQN resulted in a F1-score of 0.9902 and the SVM in a F1-score of 0.9926, i.e., a better performance of the SVM for these runs. In the same way for three classes, Table 4.1.5 can be studied and derive that the best result are obtained after 1600 episodes or $\sim$35.67 hours for the DQN and 1600 epochs or $\sim$1.66 hours for the SVM model. From studying the best runs for the three-class experiment in Table 4.1.5, it is seen that the DQN actually outperforms the SVM in terms of F1-score, however, once again, the DQN take significantly more time to train.

Table 4.1.4: The two different methods without sensor fusion, for *n* number of epochs/episodes; 2 classes.

Model \ n	50	100	200	400	800	1600	3200
DQN							
Accuracy	0.9637	0.9874	0.9874	0.9937	0.9937	x	x
Precision	0.9290	0.9860	0.9777	0.9874	0.9901	x	x
Recall	0.9655	0.9744	0.9833	0.9931	0.9901	x	x
F1-score	0.9457	0.9801	0.9804	0.9902	0.9901	x	x
SVM							
Accuracy	0.9858	0.9953	0.9953	0.9937	0.9905	0.9937	0.9921
Precision	0.9739	0.9912	0.9912	0.9961	0.9881	0.9874	0.9811
Recall	0.9823	0.9941	0.9941	0.9845	0.9823	0.9931	0.9951
F1-score	0.9780	0.9926	0.9926	0.9900	0.9851	0.9902	0.9879

Table 4.1.5: The two different methods without sensor fusion, for *n* number of epochs/episodes; 3 classes.

Model \ n	50	100	200	400	800	1600	3200
DQN							
Accuracy	0.8326	0.9550	0.9659	0.9705	0.9907	0.9876	x
Precision	0.5307	0.8833	0.9640	0.8436	0.9597	0.9656	x
Recall	0.5911	0.7695	0.7721	0.8333	0.9012	0.9810	x
F1-score	0.5477	0.7979	0.8182	0.8380	0.9277	0.9725	x
SVM							
Accuracy	0.9659	0.9767	0.9876	0.9891	0.9829	0.9907	0.9876
Precision	0.9675	0.9769	0.9889	0.9570	0.9831	0.9903	0.9525
Recall	0.6851	0.8360	0.9019	0.9006	0.8979	0.9348	0.9019
F1-score	0.6997	0.8841	0.9382	0.9261	0.9333	0.9595	0.9245

Finally, Table 4.1.6 presents the results from the four-class experiments. As follows, the highest F1-score comes from running the DQN for 1600 episodes or ~25.75 hours and the SVM for 3200 epochs or ~3.41 hours. Once again, the best performing DQN model outperforms the best performing SVM model in terms of F1-score. For these runs, this is due to the low recall score of the SVM in comparison to the DQN, i.e., 0.8802 for the SVM compared to 0.9253 of the DQN.

Table 4.1.6: The two different methods without sensor fusion, for n number of epochs/episodes; 4 classes.

Model \ n	50	100	200	400	800	1600	3200
DQN							
Accuracy	0.7402	0.8168	0.9114	0.9640	0.9730	0.9865	x
Precision	0.5793	0.4533	0.6835	0.8527	0.8961	0.9667	x
Recall	0.7065	0.5357	0.5958	0.8921	0.8515	0.9253	x
F1-score	0.5462	0.4632	0.6008	0.8696	0.8722	0.9449	x
SVM							
Accuracy	0.9339	0.9429	0.9745	0.9865	0.9835	0.9805	0.9835
Precision	0.7077	0.7586	0.9146	0.9696	0.9495	0.9472	0.9903
Recall	0.5209	0.6830	0.8655	0.8911	0.9109	0.9005	0.8802
F1-score	0.5335	0.7054	0.8876	0.9258	0.9257	0.9220	0.9270

As mentioned earlier, the DQN model performed generally worse than the SVM model. Note that the F1-score equally weighs in precision and recall and thus not always reflect the best run. For example, if precision is deemed more important than recall, then the reader is encouraged to look at that metric, and vice-versa if precision is deemed more important. A general higher score for the accuracy of the SVM is also found but too much weight should not be put on this since an imbalanced dataset is used which accuracy is as mentioned not suitable for.

On the other hand, the DQN outperforms the SVM in some cases as for the metrics, but keep in mind, as regards to time, an epoch does not equal an episode. For those runs where the DQN outperforms the SVM, it can be argued that the SVM could have been trained longer but nevertheless, the SVM could have also found a convergence already but this as mentioned is not proven with these experiments. It can also be argued that the models could have run longer in general to achieve better metrics. However, as mentioned in the beginning of this chapter, this thesis is not trying to find out the model that provides the highest F1-score, but rather check the viability for the different concepts. In addition, this could not also been done due to time constraints.

Despite the fact that the SVM is in general performing better in terms of the metrics, the best performing DQN is fairly close to the performance of the best performing SVM, and, as mentioned, the DQN even outperforms the SVM in some cases; but as mentioned, the training for the DQN is significantly longer. To conclude, the DQN model is not viewed as a viable model for classification mainly due to the slow learning process but should not be negligible as it performs fairly well in terms of the metrics presented. Additionally, the DQN model could as mentioned in Section 3.3 be optimized even further in the reward function to improve the performance. The slow learning process for a DQN is likely the reason for why it is not commonly applied in classification. However, the fact that reinforcement based learning approach is slow is known. According to the paper by Botvinick et al. [40], this is due to the incremental parameter adjustment and weak inductive bias.

Moreover, by comparing the results with and without sensor fusion, a noticeable improvement can be seen in the metrics studied in this work. Both using the SVM and the DQN model. However, the improvement varies a lot, in some cases it differs only a few thousandths and sometimes up to 0.1329. For instance classifying 3 classes with the DQN model trained for 50 episodes, the F1-score with sensor fusion is 0.6806 whereas without is 0.5477. Keep in mind that the NLSPN network that outputs the dense depth map uses data that is transposed onto a 2D plane, hence a lot of the lidar data is actually lost, arguably most of it. In other words, the 3D structure of an object and surroundings. By keeping the lidar data in 3D as well as doing a sensor fusion with the RGB images, the results are expected to be even better. Furthermore, the NLSPN network might also not have been the optimal solution. As for the results with both inputs coming in as a 2D shape, a well-considered decision has to be made since there is in fact an improvement but maybe not enough to outweigh the cost of lidar equipment.

Chapter 5

Conclusions and Future Work

This thesis seeks to answer the viability of using a reinforcement-learning approach to solve a classification task, more specifically a Deep Q-learning network, and compare this approach to a semi-classical one which uses a CNN feature extractor and SVM as a classifier. Furthermore, to investigate whether sensor fusion would assist the DQN and/or the semi-classical approach in the classification of objects. To make an evaluation, models of these approaches were built which were then compared. The results show that the DQN approach is not a good option, not least since it takes a longer time to train, and, in general, obtains a worse score (in terms of accuracy, precision, recall, and F1-score) as compared to the evaluated semi-classical approach. However, the DQN should not be negligible as it still performs fairly well, and even outperforms the semi-classical approach in some cases. As for classification tasks, reinforcement learning might still be a viable option on imbalanced text data sets and extremely imbalanced data and this is something that concurs with the findings of Lin et al.

On the other hand, the employment of sensor fusion shows a notable improvement; especially, when taking into account that most of the lidar data is lost when the data is transformed from 3D into 2D. Hence, a considerable improvement is expected to be seen when the lidar data is kept in 3D. Additionally, the dataset used in this paper does not include cases where the scene is dark, for example, in use cases taking place at night. In such cases, the employment of a lidar combined with an RGB camera is also expected to make a big improvement since unlike RGB camera, lidar works equally well in light as in dark environments. Avenues for future work includes an evaluation of sensor fusion with lidar in dark-scene scenarios; investigate the adaptability of the models, for instance between an indoor and outdoor environment; to keep the lidar data in 3D space, and investigate how much the entire lidar data actually improves the performance; to optimize the reward function in the DQN further in order to improve the performance; to train the SVM models even longer to prove the performance convergence as this work could not to this due to time constraints; and, to expand the reinforcement learning model to also be able to detect objects with the use of sensor fusion.

Bibliography

[1] *About Keras*. https://keras.io/about/. Accessed: 2021-06-06.

[2] *Adam*. https://keras.io/api/optimizers/adam/. Accessed: 2021-06-06.

[3] Albawi, Saad, Mohammed, Tareq Abed, and Al-Zawi, Saad. "Understanding of a convolutional neural network". In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

[4] Carter, Jamie, Schmid, Keil, Waters, Kirk, Betzhold, Lindy, Hadley, B, Mataosky, R, and Halleran, J. "An introduction to LiDAR technology, data, and applications". In: *NOAA Coastal Services Center* 2 (2012).

[5] *COCO Common Objects in Context*. https://cocodataset.org/#home. Accessed: 2021-06-06.

[6] Cruz Jr au2, Gabriel V. de la, Du, Yunshu, and Taylor, Matthew E. *Pre-training Neural Networks with Human Demonstrations for Deep Reinforcement Learning*. 2019. arXiv: 1709.04083 [cs.LG].

[7] *DeepMind AI Reduces Google Data Centre Cooling Bill by 40%*. https://deepmind.com/blog/article/deepmind-ai-reduces-google-data-centre-cooling-bill-40. Accessed: 2021-06-06.

[8] *Detection Evaluation*. https://cocodataset.org/#detection-eval. Accessed: 2021-06-06.

[9] *detectron2.evaluation*. https://detectron2.readthedocs.io/en/latest/modules/evaluation.html. Accessed: 2021-06-06.

[10] Dietterich, T. "Machine Learning". In: *ACM Comput. Surv.* 28.4es (Dec. 1996), 3–es. ISSN: 0360-0300. DOI: 10.1145/242224.242229.

[11] Elmenreich, Wilfried. "An introduction to sensor fusion". In: *Vienna University of Technology, Austria* 502 (2002), pp. 1–28.

[12] *Employing AI techniques to enhance returns on 5G network investments*. https://www.ericsson.com/49b63f/assets/local/ai-and-automation/docs/machine-learning-and-ai-aw-screen.pdf. Accessed: 2021-06-06.

[13] Everingham, Mark, Van Gool, Luc, Williams, Christopher KI, Winn, John, and Zisserman, Andrew. "The pascal visual object classes (voc) challenge". In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[14] Farebrother, Jesse, Machado, Marlos C., and Bowling, Michael. “Generalization and Regularization in DQN”. In: *CoRR* abs/1810.00123 (2018). arXiv: 1810.00123. URL: http://arxiv.org/abs/1810.00123.

[15] Garbin, Christian, Zhu, Xingquan, and Marques, Oge. “Dropout vs. batch normalization: an empirical study of their impact to deep learning”. In: *Multimedia Tools and Applications* 79.19-20 (). DOI: 10.1007/s11042-019-08453-9. URL: https://par.nsf.gov/biblio/10166570.

[16] Geiger, Andreas, Lenz, Philip, and Urtasun, Raquel. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[17] *Github pythonlessons Cartpole reinforcement learning*. https://github.com/pythonlessons/Reinforcement_Learning/blob/master/02_CartPole-reinforcement-learning_DDQN/Cartpole_DDQN.py. Accessed: 2021-06-06.

[18] Grandini, Margherita, Bagli, Enrico, and Visani, Giorgio. *Metrics for Multi-Class Classification: an Overview*. 2020. arXiv: 2008.05756 [stat.ML].

[19] *Greedy Algorithms*. https://brilliant.org/wiki/greedy-algorithm/. Accessed: 2021-06-06.

[20] Grossman, Mikael. “Proposal networks in object detection”. MA thesis. KTH, Mathematical Statistics, 2019.

[21] Hofmann, Thomas, Schölkopf, Bernhard, and Smola, Alexander J. “Kernel methods in machine learning”. In: *The Annals of Statistics* 36.3 (June 2008). ISSN: 0090-5364. DOI: 10.1214/009053607000000677. URL: http://dx.doi.org/10.1214/009053607000000677.

[22] *Keras compile method*. https://www.tensorflow.org/api_docs/python/tf/keras/Model#compile. Accessed: 2021-06-06.

[23] *Keras fit method*. https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit. Accessed: 2021-06-06.

[24] *Keras Model Class*. https://www.tensorflow.org/api_docs/python/tf/keras/Model. Accessed: 2021-06-06.

[25] *Keras predict method*. https://www.tensorflow.org/api_docs/python/tf/keras/Model#predict. Accessed: 2021-06-06.

[26] *Kitti depth completion evaluation*. http://www.cvlibs.net/datasets/kitti/eval_depth.php?benchmark=depth_completion. Accessed: 2021-06-06.

[27] Koutroumbas, Konstantinos and Theodoridis, Sergios. *Pattern Recognition 4th Ed.* Elsevier Science, 2009.

[28] Li, Zhu, Ton, Jean-Francois, Oglic, Dino, and Sejdinovic, Dino. “Towards a unified analysis of random Fourier features”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 3905–3914.

[29] Lin, Enlu, Chen, Qiong, and Qi, Xiaoming. "Deep Reinforcement Learning for Imbalanced Classification". In: *CoRR* abs/1901.01379 (2019). arXiv: 1901.01379. URL: http://arxiv.org/abs/1901.01379.

[30] Lipton, Zachary Chase, Elkan, Charles, and Narayanaswamy, Balakrishnan. *Thresholding Classifiers to Maximize F1 Score*. 2014. arXiv: 1402.1892 [stat.ML].

[31] Louis, Ruwaid and Yu, David. *A study of the exploration/exploitation trade-off in reinforcement learning : Applied to autonomous driving*. 2019.

[32] Lundquist, Christian. "Sensor fusion for automotive applications". PhD thesis. Linköping University Electronic Press, 2011.

[33] *Mean squared error*. https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-error. Accessed: 2021-06-06.

[34] *ML - Support Vector Machine(SVM)*. https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_classification_algorithms_support_vector_machine.htm. Accessed: 2021-06-06.

[35] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin A. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[36] *Object Detection vs Object Recognition vs Image Segmentation*. https://www.geeksforgeeks.org/object-detection-vs-object-recognition-vs-image-segmentation/. Accessed: 2021-06-06.

[37] Park, Jinsun, Joo, Kyungdon, Hu, Zhe, Liu, Chi-Kuei, and Kweon, In So. "Non-Local Spatial Propagation Network for Depth Completion". In: *European Conference on Computer Vision (ECCV)*. 2020.

[38] Patterson, Josh and Gibson, Adam. *Deep Learning: A Practitioner's Approach*. O'Reilly, 2017.

[39] *QuasiSVM*. https://keras.io/examples/keras_recipes/quasi_svm/. Accessed: 2021-06-06.

[40] *Reinforcement Learning, Fast and slow*. https://www.sciencedirect.com/science/article/pii/S1364661319300610. Accessed: 2021-06-06.

[41] Ruder, Sebastian. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[42] Samek, Wojciech, Binder, Alexander, Montavon, Grégoire, Lapuschkin, Sebastian, and Müller, Klaus-Robert. “Evaluating the Visualization of What a Deep Neural Network Has Learned”. In: *IEEE Transactions on Neural Networks and Learning Systems* 28.11 (2017), pp. 2660–2673. DOI: 10.1109/TNNLS.2016.2599820.

[43] *scikit-learn Machine Learning in Python*. https://scikit-learn.org/stable/index.html. Accessed: 2021-06-06.

[44] Simonyan, Karen and Zisserman, Andrew. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[45] *sklearn.metrics.classification*$_r$*eport*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html. Accessed: 2021-06-06.

[46] Tang, Jie, Tian, Fei-Peng, Feng, Wei, Li, Jian, and Tan, Ping. *Learning Guided Convolutional Network for Depth Completion*. 2019. arXiv: 1908.01238 [cs.CV].

[47] *Tutorial on Support Vector Machine (SVM)*. https://course.ccs.neu.edu/cs5100f11/resources/jakkula.pdf. Accessed: 2021-06-06.

[48] Uhrig, Jonas, Schneider, Nick, Schneider, Lukas, Franke, Uwe, Brox, Thomas, and Geiger, Andreas. *Sparsity Invariant CNNs*. 2017. arXiv: 1708.06500 [cs.CV].

[49] *VGG-16 | CNN model*. https://www.geeksforgeeks.org/vgg-16-cnn-model/. Accessed: 2021-06-06.

[50] *VGG16 and VGG19*. https://keras.io/api/applications/vgg/. Accessed: 2021-06-06.

[51] Wang, Shaojie, Wu, Tong, and Vorobeychik, Yevgeniy. *Towards Robust Sensor Fusion in Visual Perception*. 2020. arXiv: 2006.13192 [cs.CV].

[52] Weng, Cheng G and Poon, Josiah. “A new evaluation measure for imbalanced datasets”. In: *Proceedings of the 7th Australasian Data Mining Conference-Volume 87*. 2008, pp. 27–32.

[53] *What Is Machine Learning: Definition, Types, Applications And Examples*. https://www.potentiaco.com/what-is-machine-learning-definition-types-applications-and-examples/. Accessed: 2021-06-06.

[54] Wiering, Marco A., Hasselt, Hado van, Pietersma, Auke-Dirk, and Schomaker, Lambert. “Reinforcement learning algorithms for solving classification problems”. In: *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. 2011, pp. 91–96. DOI: 10.1109/ADPRL.2011.5967372.

[55] Winston, Patrick Henry. *Artificial intelligence (3rd ed.)* Addison-Wesley Pub. Co, 1992.

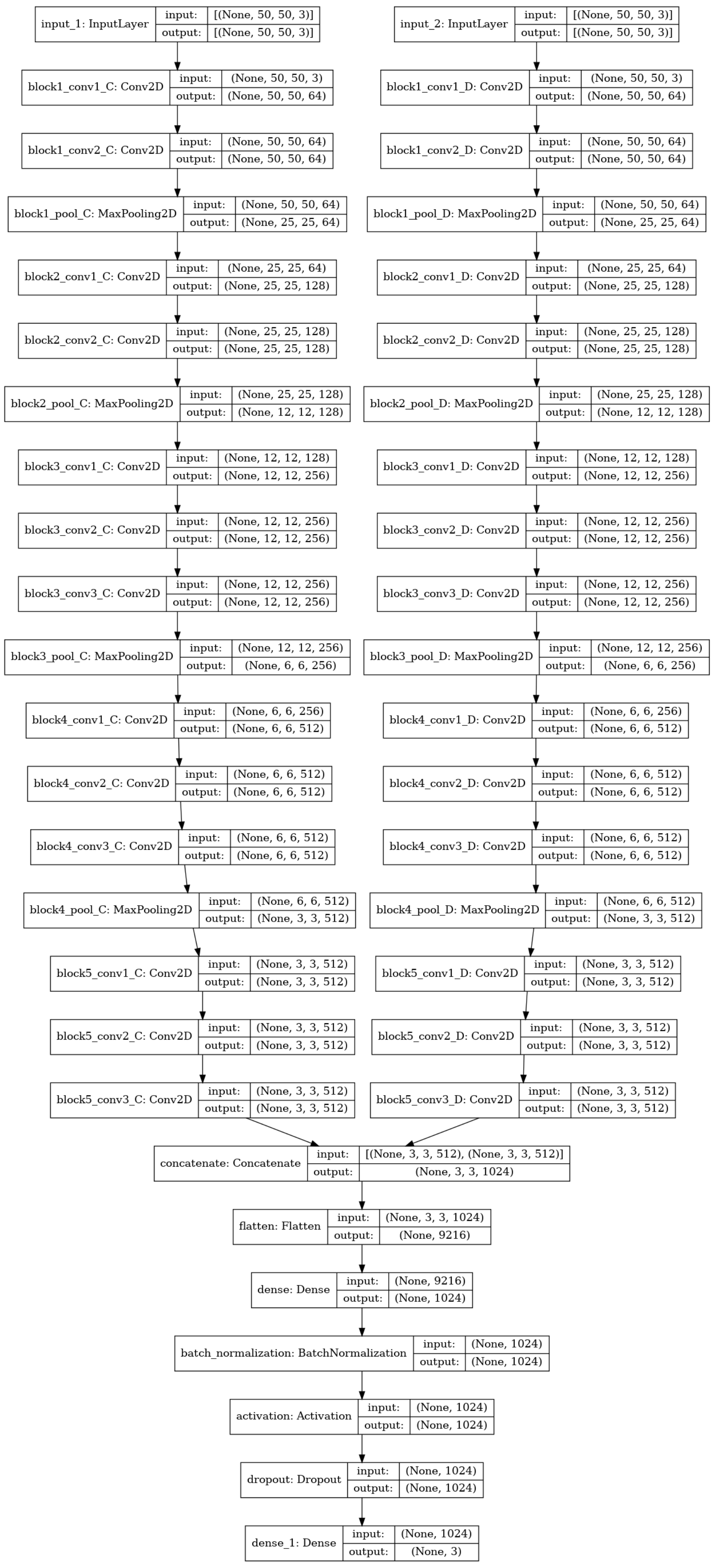
[56] Wu, Yuxin, Kirillov, Alexander, Massa, Francisco, Lo, Wan-Yen, and Girshick, Ross. *Detectron2*. `https://github.com/facebookresearch/detectron2`. 2019.

[57] Wynsberghe, Aimee van. “Sustainable AI: AI for sustainability and the sustainability of AI”. In: *AI and Ethics* (2021), pp. 1–6.

[58] Yamashita, Rikiya, Nishio, Mizuho, Do, Richard, and Togashi, Kaori. “Convolutional neural networks: an overview and application in radiology”. In: *Insights into Imaging* 9 (June 2018). DOI: `10.1007/s13244-018-0639-9`.

Appendix A

CNN networks



input_1: InputLayer | input: [(None, 50, 50, 3)] | output: [(None, 50, 50, 3)]
input_2: InputLayer | input: [(None, 50, 50, 3)] | output: [(None, 50, 50, 3)]
block1_conv1_C: Conv2D | input: (None, 50, 50, 3) | output: (None, 50, 50, 64)
block1_conv1_D: Conv2D | input: (None, 50, 50, 3) | output: (None, 50, 50, 64)
block1_conv2_C: Conv2D | input: (None, 50, 50, 64) | output: (None, 50, 50, 64)
block1_conv2_D: Conv2D | input: (None, 50, 50, 64) | output: (None, 50, 50, 64)
block1_pool_C: MaxPooling2D | input: (None, 50, 50, 64) | output: (None, 25, 25, 64)
block1_pool_D: MaxPooling2D | input: (None, 50, 50, 64) | output: (None, 25, 25, 64)
block2_conv1_C: Conv2D | input: (None, 25, 25, 64) | output: (None, 25, 25, 128)
block2_conv1_D: Conv2D | input: (None, 25, 25, 64) | output: (None, 25, 25, 128)
block2_conv2_C: Conv2D | input: (None, 25, 25, 128) | output: (None, 25, 25, 128)
block2_conv2_D: Conv2D | input: (None, 25, 25, 128) | output: (None, 25, 25, 128)
block2_pool_C: MaxPooling2D | input: (None, 25, 25, 128) | output: (None, 12, 12, 128)
block2_pool_D: MaxPooling2D | input: (None, 25, 25, 128) | output: (None, 12, 12, 128)
block3_conv1_C: Conv2D | input: (None, 12, 12, 128) | output: (None, 12, 12, 256)
block3_conv1_D: Conv2D | input: (None, 12, 12, 128) | output: (None, 12, 12, 256)
block3_conv2_C: Conv2D | input: (None, 12, 12, 256) | output: (None, 12, 12, 256)
block3_conv2_D: Conv2D | input: (None, 12, 12, 256) | output: (None, 12, 12, 256)
block3_conv3_C: Conv2D | input: (None, 12, 12, 256) | output: (None, 12, 12, 256)
block3_conv3_D: Conv2D | input: (None, 12, 12, 256) | output: (None, 12, 12, 256)
block3_pool_C: MaxPooling2D | input: (None, 12, 12, 256) | output: (None, 6, 6, 256)
block3_pool_D: MaxPooling2D | input: (None, 12, 12, 256) | output: (None, 6, 6, 256)
block4_conv1_C: Conv2D | input: (None, 6, 6, 256) | output: (None, 6, 6, 512)
block4_conv1_D: Conv2D | input: (None, 6, 6, 256) | output: (None, 6, 6, 512)
block4_conv2_C: Conv2D | input: (None, 6, 6, 512) | output: (None, 6, 6, 512)
block4_conv2_D: Conv2D | input: (None, 6, 6, 512) | output: (None, 6, 6, 512)
block4_conv3_C: Conv2D | input: (None, 6, 6, 512) | output: (None, 6, 6, 512)
block4_conv3_D: Conv2D | input: (None, 6, 6, 512) | output: (None, 6, 6, 512)
block4_pool_C: MaxPooling2D | input: (None, 6, 6, 512) | output: (None, 3, 3, 512)
block4_pool_D: MaxPooling2D | input: (None, 6, 6, 512) | output: (None, 3, 3, 512)
block5_conv1_C: Conv2D | input: (None, 3, 3, 512) | output: (None, 3, 3, 512)
block5_conv1_D: Conv2D | input: (None, 3, 3, 512) | output: (None, 3, 3, 512)
block5_conv2_C: Conv2D | input: (None, 3, 3, 512) | output: (None, 3, 3, 512)
block5_conv2_D: Conv2D | input: (None, 3, 3, 512) | output: (None, 3, 3, 512)
block5_conv3_C: Conv2D | input: (None, 3, 3, 512) | output: (None, 3, 3, 512)
block5_conv3_D: Conv2D | input: (None, 3, 3, 512) | output: (None, 3, 3, 512)
concatenate: Concatenate | input: [(None, 3, 3, 512), (None, 3, 3, 512)] | output: (None, 3, 3, 1024)
flatten: Flatten | input: (None, 3, 3, 1024) | output: (None, 9216)
dense: Dense | input: (None, 9216) | output: (None, 1024)
batch_normalization: BatchNormalization | input: (None, 1024) | output: (None, 1024)
activation: Activation | input: (None, 1024) | output: (None, 1024)
dropout: Dropout | input: (None, 1024) | output: (None, 1024)
dense_1: Dense | input: (None, 1024) | output: (None, 3)

Figure A.0.1: A visualisation of the CNN for the DQN when sensor fusion is employed.

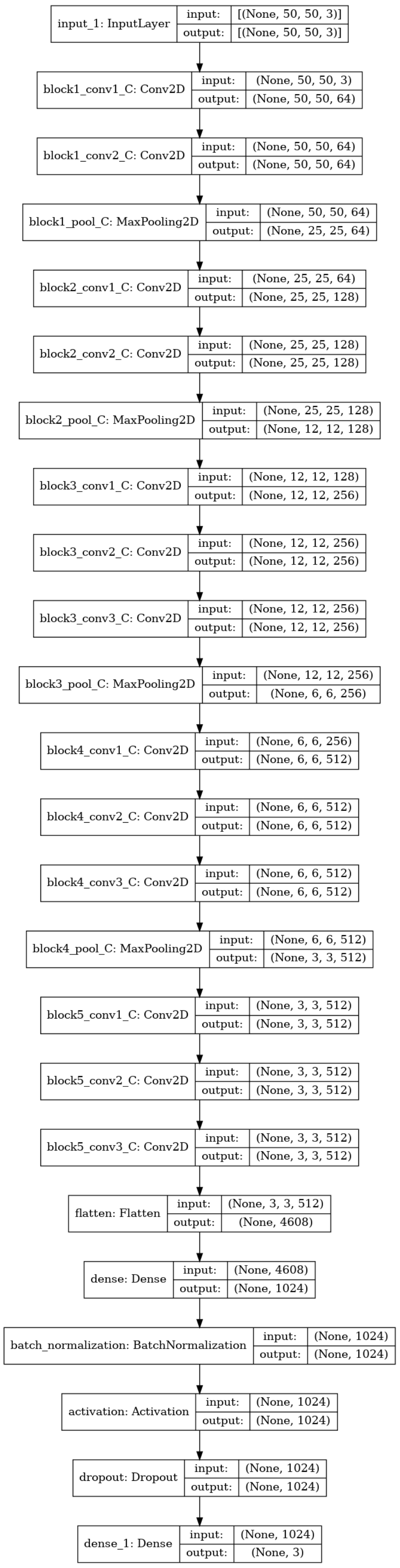

input_1: InputLayer
input: [(None, 50, 50, 3)]
output: [(None, 50, 50, 3)]
block1_conv1_C: Conv2D
input: (None, 50, 50, 3)
output: (None, 50, 50, 64)
block1_conv2_C: Conv2D
input: (None, 50, 50, 64)
output: (None, 50, 50, 64)
block1_pool_C: MaxPooling2D
input: (None, 50, 50, 64)
output: (None, 25, 25, 64)
block2_conv1_C: Conv2D
input: (None, 25, 25, 64)
output: (None, 25, 25, 128)
block2_conv2_C: Conv2D
input: (None, 25, 25, 128)
output: (None, 25, 25, 128)
block2_pool_C: MaxPooling2D
input: (None, 25, 25, 128)
output: (None, 12, 12, 128)
block3_conv1_C: Conv2D
input: (None, 12, 12, 128)
output: (None, 12, 12, 256)
block3_conv2_C: Conv2D
input: (None, 12, 12, 256)
output: (None, 12, 12, 256)
block3_conv3_C: Conv2D
input: (None, 12, 12, 256)
output: (None, 12, 12, 256)
block3_pool_C: MaxPooling2D
input: (None, 12, 12, 256)
output: (None, 6, 6, 256)
block4_conv1_C: Conv2D
input: (None, 6, 6, 256)
output: (None, 6, 6, 512)
block4_conv2_C: Conv2D
input: (None, 6, 6, 512)
output: (None, 6, 6, 512)
block4_conv3_C: Conv2D
input: (None, 6, 6, 512)
output: (None, 6, 6, 512)
block4_pool_C: MaxPooling2D
input: (None, 6, 6, 512)
output: (None, 3, 3, 512)
block5_conv1_C: Conv2D
input: (None, 3, 3, 512)
output: (None, 3, 3, 512)
block5_conv2_C: Conv2D
input: (None, 3, 3, 512)
output: (None, 3, 3, 512)
block5_conv3_C: Conv2D
input: (None, 3, 3, 512)
output: (None, 3, 3, 512)
flatten: Flatten
input: (None, 3, 3, 512)
output: (None, 4608)
dense: Dense
input: (None, 4608)
output: (None, 1024)
batch_normalization: BatchNormalization
input: (None, 1024)
output: (None, 1024)
activation: Activation
input: (None, 1024)
output: (None, 1024)
dropout: Dropout
input: (None, 1024)
output: (None, 1024)
dense_1: Dense
input: (None, 1024)
output: (None, 3)

Figure A.0.2: A visualisation of the CNN for the DQN without sensor fusion.

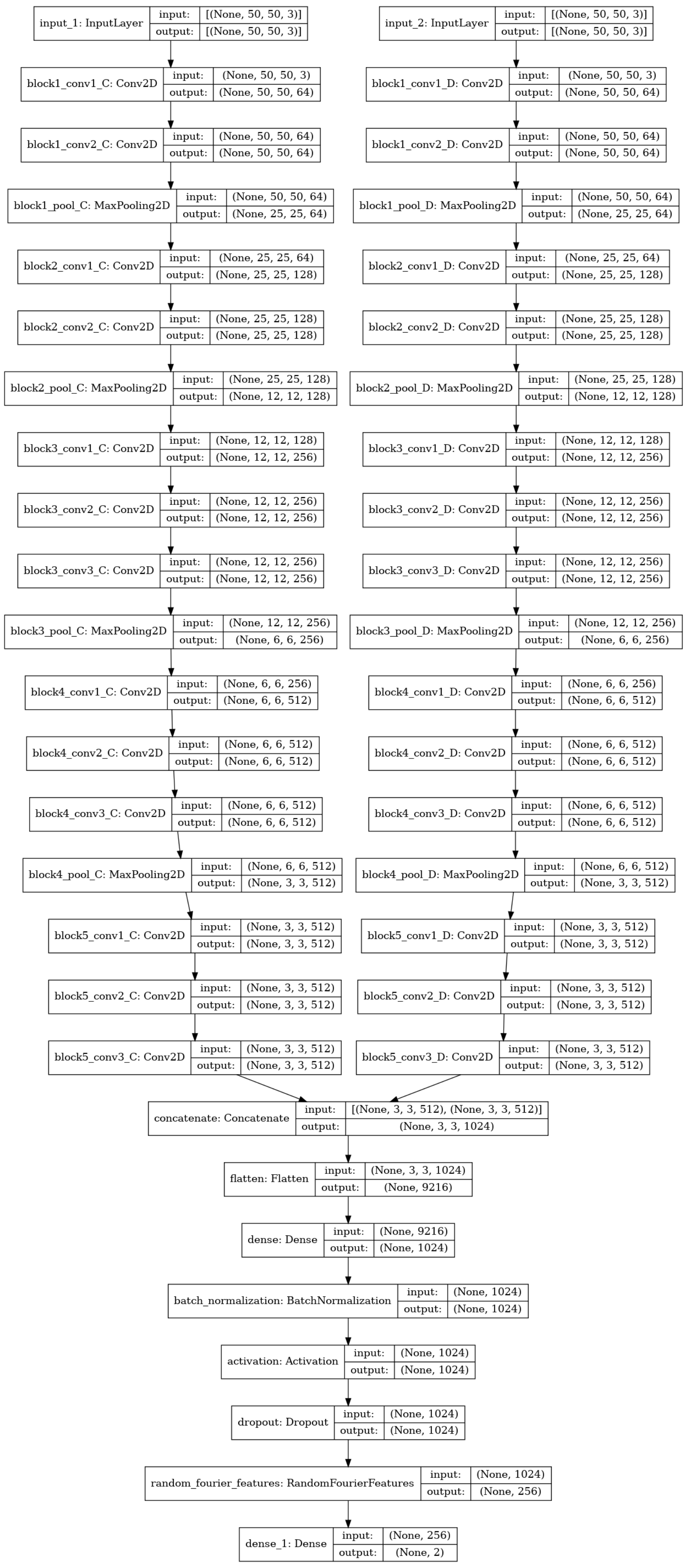

input_1: InputLayer | input: [(None, 50, 50, 3)] | output: [(None, 50, 50, 3)]
input_2: InputLayer | input: [(None, 50, 50, 3)] | output: [(None, 50, 50, 3)]
block1_conv1_C: Conv2D | input: (None, 50, 50, 3) | output: (None, 50, 50, 64)
block1_conv1_D: Conv2D | input: (None, 50, 50, 3) | output: (None, 50, 50, 64)
block1_conv2_C: Conv2D | input: (None, 50, 50, 64) | output: (None, 50, 50, 64)
block1_conv2_D: Conv2D | input: (None, 50, 50, 64) | output: (None, 50, 50, 64)
block1_pool_C: MaxPooling2D | input: (None, 50, 50, 64) | output: (None, 25, 25, 64)
block1_pool_D: MaxPooling2D | input: (None, 50, 50, 64) | output: (None, 25, 25, 64)
block2_conv1_C: Conv2D | input: (None, 25, 25, 64) | output: (None, 25, 25, 128)
block2_conv1_D: Conv2D | input: (None, 25, 25, 64) | output: (None, 25, 25, 128)
block2_conv2_C: Conv2D | input: (None, 25, 25, 128) | output: (None, 25, 25, 128)
block2_conv2_D: Conv2D | input: (None, 25, 25, 128) | output: (None, 25, 25, 128)
block2_pool_C: MaxPooling2D | input: (None, 25, 25, 128) | output: (None, 12, 12, 128)
block2_pool_D: MaxPooling2D | input: (None, 25, 25, 128) | output: (None, 12, 12, 128)
block3_conv1_C: Conv2D | input: (None, 12, 12, 128) | output: (None, 12, 12, 256)
block3_conv1_D: Conv2D | input: (None, 12, 12, 128) | output: (None, 12, 12, 256)
block3_conv2_C: Conv2D | input: (None, 12, 12, 256) | output: (None, 12, 12, 256)
block3_conv2_D: Conv2D | input: (None, 12, 12, 256) | output: (None, 12, 12, 256)
block3_conv3_C: Conv2D | input: (None, 12, 12, 256) | output: (None, 12, 12, 256)
block3_conv3_D: Conv2D | input: (None, 12, 12, 256) | output: (None, 12, 12, 256)
block3_pool_C: MaxPooling2D | input: (None, 12, 12, 256) | output: (None, 6, 6, 256)
block3_pool_D: MaxPooling2D | input: (None, 12, 12, 256) | output: (None, 6, 6, 256)
block4_conv1_C: Conv2D | input: (None, 6, 6, 256) | output: (None, 6, 6, 512)
block4_conv1_D: Conv2D | input: (None, 6, 6, 256) | output: (None, 6, 6, 512)
block4_conv2_C: Conv2D | input: (None, 6, 6, 512) | output: (None, 6, 6, 512)
block4_conv2_D: Conv2D | input: (None, 6, 6, 512) | output: (None, 6, 6, 512)
block4_conv3_C: Conv2D | input: (None, 6, 6, 512) | output: (None, 6, 6, 512)
block4_conv3_D: Conv2D | input: (None, 6, 6, 512) | output: (None, 6, 6, 512)
block4_pool_C: MaxPooling2D | input: (None, 6, 6, 512) | output: (None, 3, 3, 512)
block4_pool_D: MaxPooling2D | input: (None, 6, 6, 512) | output: (None, 3, 3, 512)
block5_conv1_C: Conv2D | input: (None, 3, 3, 512) | output: (None, 3, 3, 512)
block5_conv1_D: Conv2D | input: (None, 3, 3, 512) | output: (None, 3, 3, 512)
block5_conv2_C: Conv2D | input: (None, 3, 3, 512) | output: (None, 3, 3, 512)
block5_conv2_D: Conv2D | input: (None, 3, 3, 512) | output: (None, 3, 3, 512)
block5_conv3_C: Conv2D | input: (None, 3, 3, 512) | output: (None, 3, 3, 512)
block5_conv3_D: Conv2D | input: (None, 3, 3, 512) | output: (None, 3, 3, 512)
concatenate: Concatenate | input: [(None, 3, 3, 512), (None, 3, 3, 512)] | output: (None, 3, 3, 1024)
flatten: Flatten | input: (None, 3, 3, 1024) | output: (None, 9216)
dense: Dense | input: (None, 9216) | output: (None, 1024)
batch_normalization: BatchNormalization | input: (None, 1024) | output: (None, 1024)
activation: Activation | input: (None, 1024) | output: (None, 1024)
dropout: Dropout | input: (None, 1024) | output: (None, 1024)
random_fourier_features: RandomFourierFeatures | input: (None, 1024) | output: (None, 256)
dense_1: Dense | input: (None, 256) | output: (None, 2)

Figure A.0.3: A visualisation of the CNN for the SVM when sensor fusion is employed.

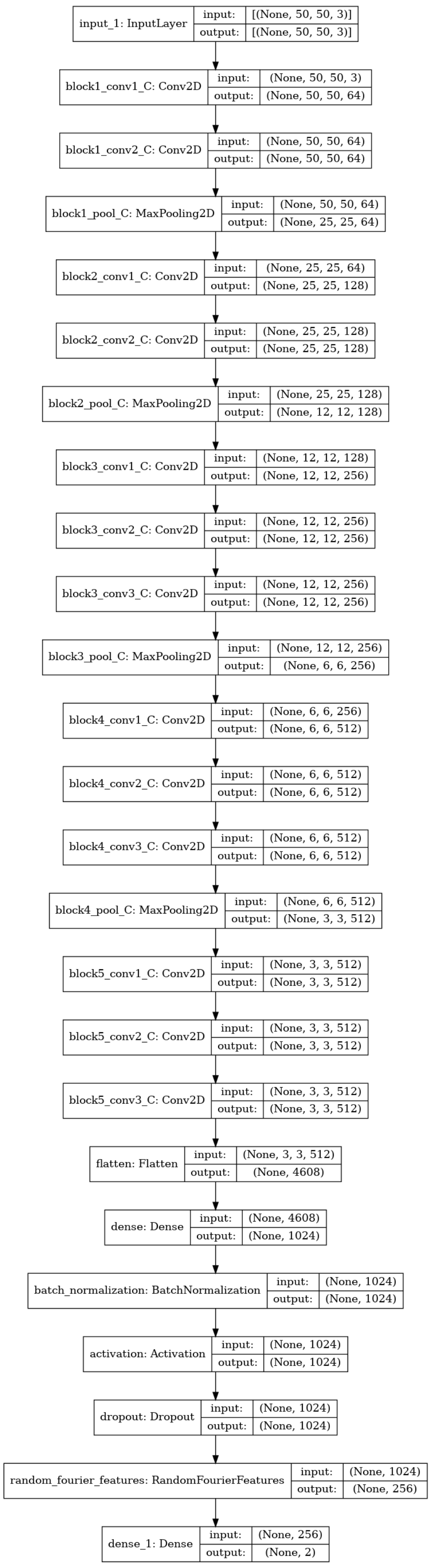

input_1: InputLayer | input: [(None, 50, 50, 3)] | output: [(None, 50, 50, 3)]
block1_conv1_C: Conv2D | input: (None, 50, 50, 3) | output: (None, 50, 50, 64)
block1_conv2_C: Conv2D | input: (None, 50, 50, 64) | output: (None, 50, 50, 64)
block1_pool_C: MaxPooling2D | input: (None, 50, 50, 64) | output: (None, 25, 25, 64)
block2_conv1_C: Conv2D | input: (None, 25, 25, 64) | output: (None, 25, 25, 128)
block2_conv2_C: Conv2D | input: (None, 25, 25, 128) | output: (None, 25, 25, 128)
block2_pool_C: MaxPooling2D | input: (None, 25, 25, 128) | output: (None, 12, 12, 128)
block3_conv1_C: Conv2D | input: (None, 12, 12, 128) | output: (None, 12, 12, 256)
block3_conv2_C: Conv2D | input: (None, 12, 12, 256) | output: (None, 12, 12, 256)
block3_conv3_C: Conv2D | input: (None, 12, 12, 256) | output: (None, 12, 12, 256)
block3_pool_C: MaxPooling2D | input: (None, 12, 12, 256) | output: (None, 6, 6, 256)
block4_conv1_C: Conv2D | input: (None, 6, 6, 256) | output: (None, 6, 6, 512)
block4_conv2_C: Conv2D | input: (None, 6, 6, 512) | output: (None, 6, 6, 512)
block4_conv3_C: Conv2D | input: (None, 6, 6, 512) | output: (None, 6, 6, 512)
block4_pool_C: MaxPooling2D | input: (None, 6, 6, 512) | output: (None, 3, 3, 512)
block5_conv1_C: Conv2D | input: (None, 3, 3, 512) | output: (None, 3, 3, 512)
block5_conv2_C: Conv2D | input: (None, 3, 3, 512) | output: (None, 3, 3, 512)
block5_conv3_C: Conv2D | input: (None, 3, 3, 512) | output: (None, 3, 3, 512)
flatten: Flatten | input: (None, 3, 3, 512) | output: (None, 4608)
dense: Dense | input: (None, 4608) | output: (None, 1024)
batch_normalization: BatchNormalization | input: (None, 1024) | output: (None, 1024)
activation: Activation | input: (None, 1024) | output: (None, 1024)
dropout: Dropout | input: (None, 1024) | output: (None, 1024)
random_fourier_features: RandomFourierFeatures | input: (None, 1024) | output: (None, 256)
dense_1: Dense | input: (None, 256) | output: (None, 2)

Figure A.0.4: A visualisation of the CNN for the SVM without sensor fusion.

Appendix B

DQN graph

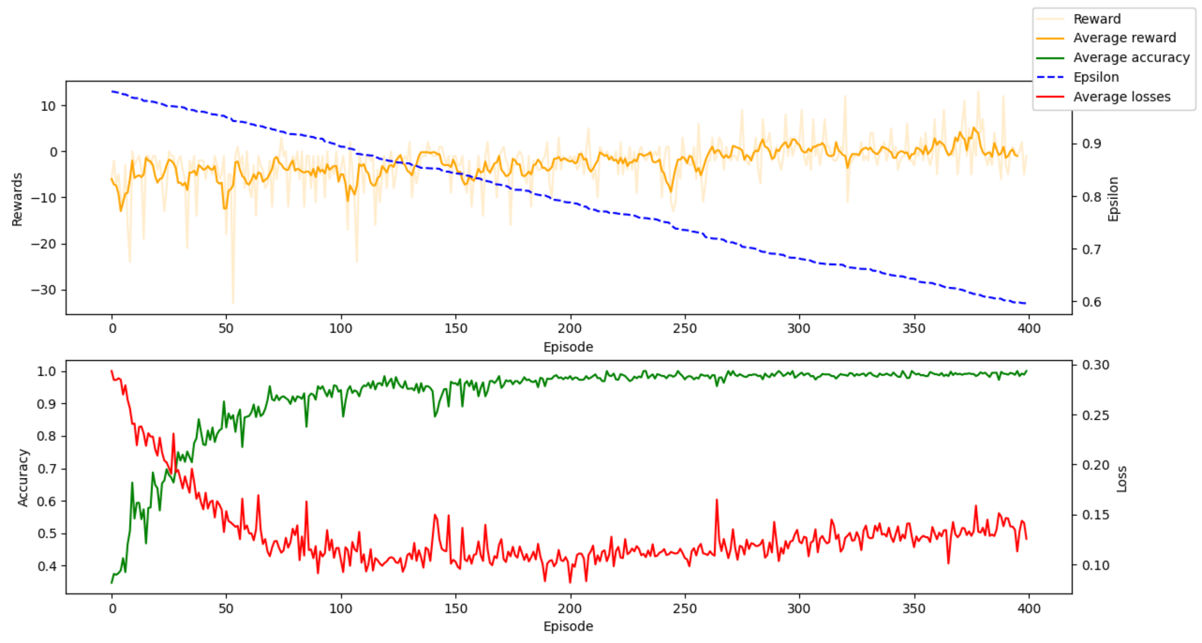


Figure B.0.1: Accuracy, loss, epsilon and reward graph of the DQN algorithm for four classes, 400 episodes when sensor fusion is employed.