Task 1

Introduction

Task 1 aims to train and test the model with the given image data to classify the data to the corresponding class. To build the optimising Convolutional Neural Networks (CNN) model, this report has processed tests on the model to progress the best model performance and discussed the comparison.

The model testing and comparison are divided into three steps. First, progress is made on the simple CNN model to understand the gap in the model.
Second, data augmentation (DA), batch normalisation (BN), and hyperparameter adjustment have been adopted.
Lastly, adapted a new model to supplement the model from the previous steps.

Method

This section of the report describes the method used to create the models for each step, helping to clarify the result.

- Preprocessing

Before inputting the data into the model, min-max preprocessing resolves the difference in the scale of the input data. This essential step can increase learning speed and stabilise the gradient descent.

- Step 1: Shallow CNN model

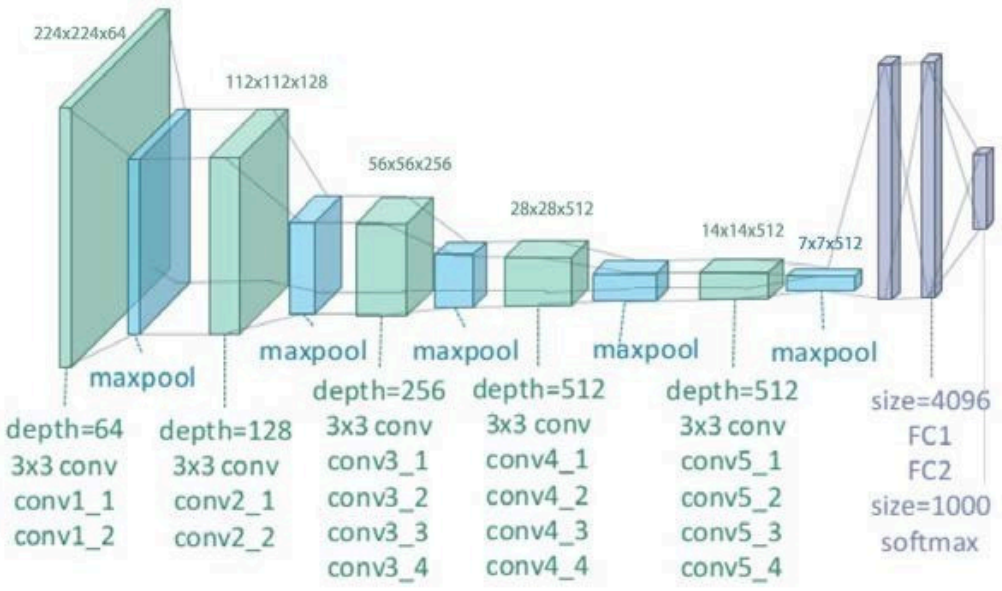
The initial CNN model is a shallow model containing a single layer for each significant phase of CNN (Input layer, Convolutional layer, Pooling layer, Flatten layer, fully connected layer, and Output layer). MyModel includes the init method. The role of each layer is displayed, and the call method explains the Forward Propagation process.
Followed by the loss function and optimiser have been set. As the Target labels are in integer format, the sparse_categorical_crossentropy is applied. The Adam is set as it applies an adaptive Learning Rate due to its effective weight modification.
During the training step, it accomplishes Backpropagation. The tf.GradientTape() practices auto-difference. After producing the prediction, the loss will be calculated using loss_object. Based on this, GradientTape produces the gradient so the optimiser.apply_gradients() can update the weight. During the test step, the model is not allowed to learn but is focused on evaluating its generalisation performance.
The last part was adapted from Longils (2020). In this part, the coding runs 20 epochs of the model to run its learning and evaluation, displaying loss and accuracy. Each epoch's train and test accuracy has been visualised to help understand the model's performance.

- Step 2: Apply of Data Augmentation(DA), Batch Normalization(BN) and adjustment in hyperparameter

The code was adapted from Osetrov(2024). In the first half of the code, the image was converted to float32 to get the model process stable learning. In addition, applied AUTOTUNE to get data loading and preprocessing automatically optimised.
DA was applied to reduce overfitting by modifying the image data (Osetrov, 2024).
Experiments were performed on the activation function and hyperparameters to find the highest-performing model.
In model 1, the Relu and 3x3 kernel size. Meanwhile, model 2 has applied the Selu and 2x2 Kernel size. According to Olamendy (2023), the Relu is efficient while avoiding the vanishing and exploding gradient issues. However, the Relu can cause a dying ReLU problem. It is caused when the neuron only receives negative input, leading to inactivation.
In contrast to ReLUs, SELUs cannot die. In addition, Selu learns faster and better through its internal-normalisation. The 2x2 kernel size has been adapted to check if the smaller kernel could find features in more detail and improve the performance. Padding
The second half of the coding runs each model 1 and 2 20 epochs and visualises the results into graphs displaying the loss and accuracy to make adequate for comparison.

- Step 3:Visual Geometry Group (VGG)-19

Figure 1. VGG-19



Adapted from Lim(2020).

The VGG-19 model was adapted from Bhamre (2025). It is a deep architecture model with 19 layers that accurately identifies diverse objects within images (Grigoryan, 2023). However, according to Grigoryan (2023), “VGGs are resource-intensive computations and the demand for computational resources limit its scalability, making it less practical for deployment on resource-constrained devices.” To prevent this, transfer learning has been applied.
In the coding, the VGG-19 model trained by the imagenet was loaded. In addition, the Fully Connected Layer and Dense Layer were removed to customise into layers suitable for the current dataset. In custom layers, the Fully Connected Layer with 256 neurons with Relu activation function was added, and the output layer with 10 neurons was added as the model

needed to classify the data into 10 classes. The activation function softmax will allow the probability production for each class. The BatchNormalization() and Dropout(0.5) are applied to avoid the overfitting. All of the above settings were complied with the loaded VGG-19 by model.compile(). In the following coding, by using the ImageDataGenerator(), the preprocessing and augmentation are applied to improve model performance. After running the model, the result has been plotted to compare the train and validation accuracy.

Result and Discussion

This part of the report discusses each model's results and explains how implementing approaches such as DA and model architecture changes complemented the former model's fault.

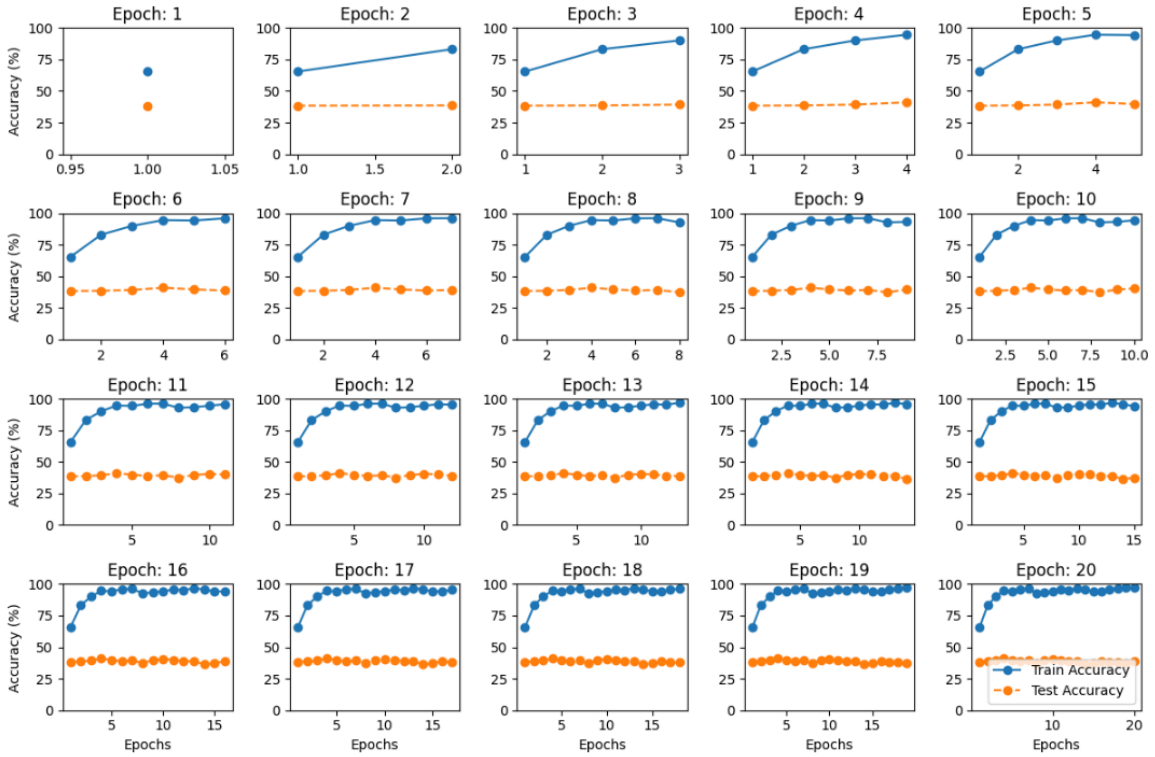
- Define evaluation protocol

In task 1, each model is evaluated by comparing the train's and validation's loss and accuracy. By visualising the evaluation, the results were understood readily.

- Step 1 model evaluation

The Step 1 model is shallow, having a single convolutional and pooling layer.
Figure 2 shows significant differences between the train and test accuracy, indicating overfitting on training data. Throughout the 20 epochs, the training accuracy consistently increased, while the test accuracy was under 50%.

Figure 2.

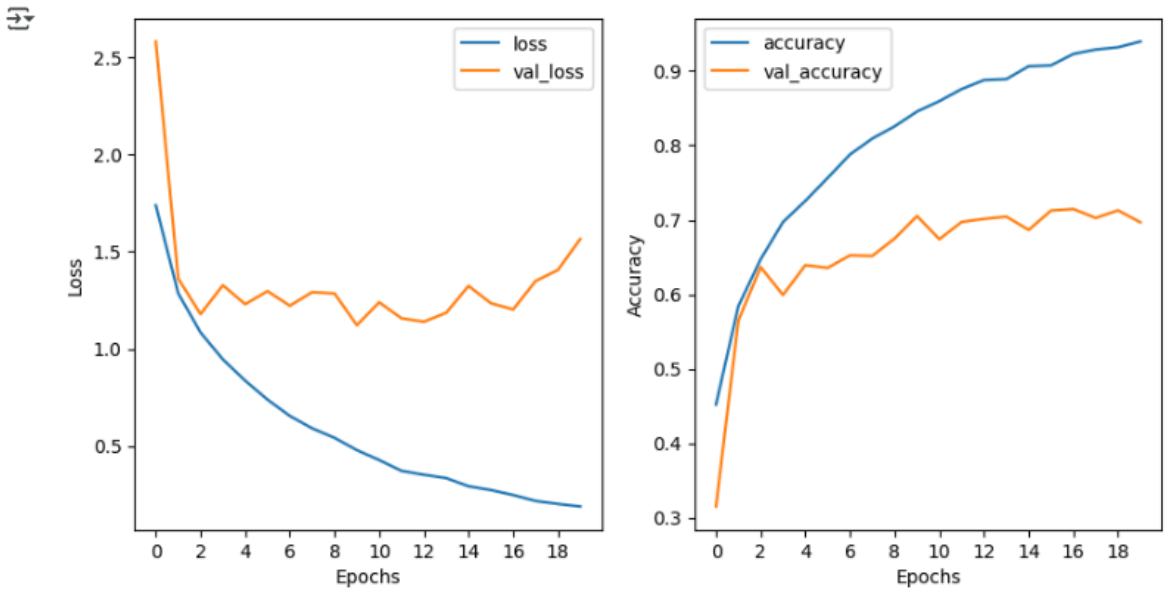


- Step 2 model evaluation

To correct the overfitting in the former model, the models in Step 2 included DA.

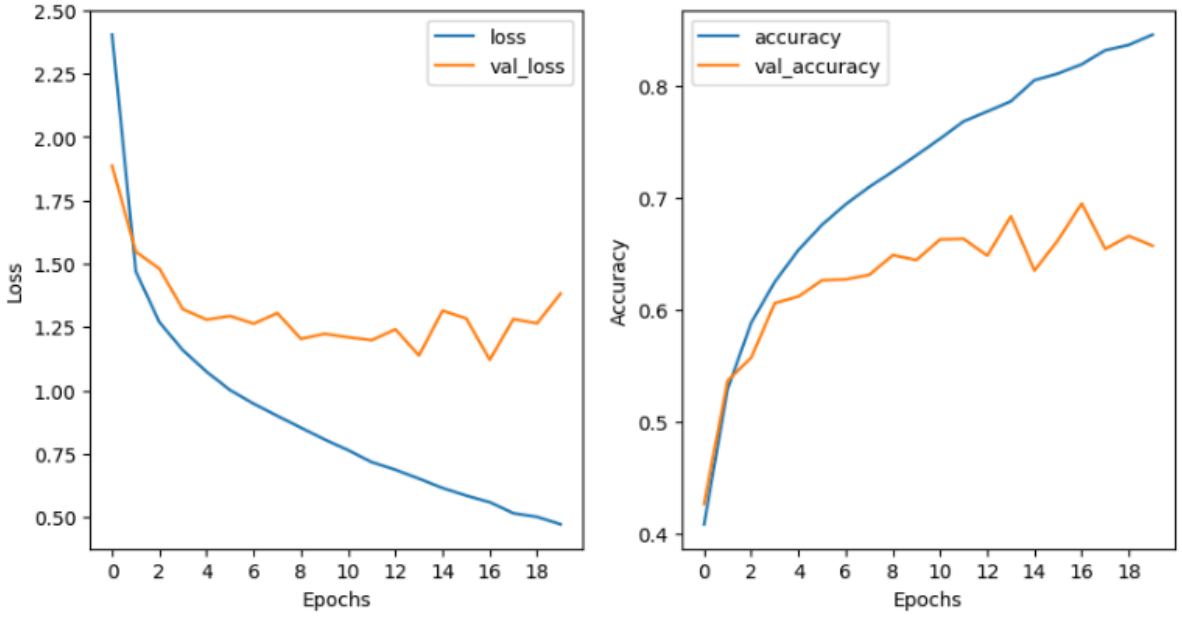
Furthermore, it deepened the convolutional layer by five, and the pooling layers by three, and BN was applied to increase its running speed. The adapted approaches were expected to help it learn more complicated features and prevent overfitting.
However, Figures 3 and 4 show that neither model can narrow the meaningful gap between the training and test results, indicating the possibility of overfitting. To compare models 1 and 2, as previously referred, the model applied Relu and kernel size set as 3x3 showed slightly higher performance by scoring accuracy around 71% while the other was around 69%. Even though it was not evident, it is assumed that the difference came from the activation function, as the 3x3 kernel application to model 2 has not changed the performance.

Figure 3. Model 1




```
Best Validation Loss for Model 1: 1.1222
Best Validation Accuracy for Model 1: 0.7146
```

Figure 4. Model 2



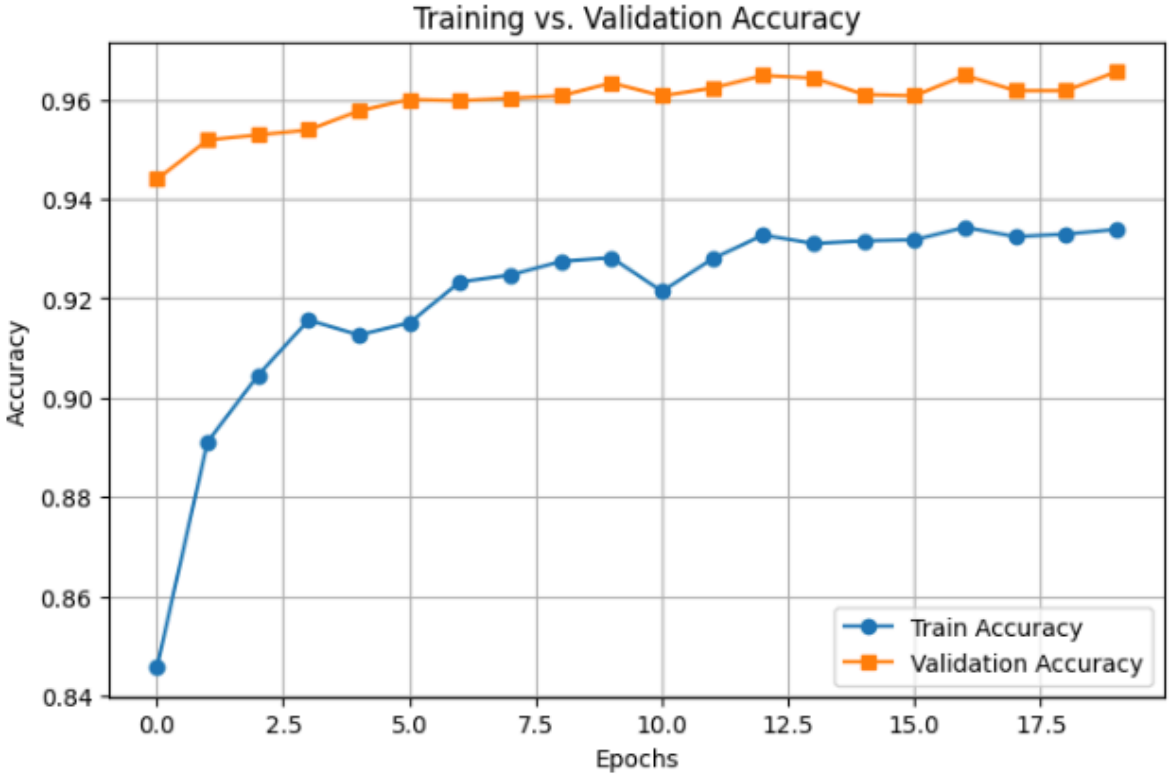

```
Best Validation Loss for Model 2: 1.1207
Best Validation Accuracy for Model 2: 0.6948
```

- Step 3 model evaluation

In the formal stage, even though model layers were added and reinforcement of the overfitting prevention was reinforced by applying DA, the model still could not perform well. Therefore, Step 3 implemented a further complex model through transfer learning. As previously referred, the VGG-19 model is adequate as it accurately identifies diverse objects within images and adds more variety for recognising data than the VGG-16 (Simonyan & Zisserman, 2015).

In Figure 5, the test accuracy has been steady between 94~96%, which is approximate to training accuracy. It can be interpreted that there is no overfitting, and it has excellent generalisation performance.

Figure 5.




```
123/123 ━━━━━━━━━━━━━━━━━━━━ 28s 223ms/step - accuracy: 0.9708 - loss: 0.2371
Test accuracy: 0.9656
```

Conclusion

Progressing to Task 1 has developed the models through the Steps. Implementing the DA and BN, adding layers, and setting the hyperparameter have gradually reinforced the model. The process emphasised the effectiveness and importance of each implementation. Adding referred implementations enabled it to optimise the VGG-19 model and achieve excellent performance.

Task 2

Introduction

In Task 2, it is split into three parts. The Methods section explains two approaches to Reinforcement Learning. Value Iteration and the Q-Learning approach will be implemented in actual coding and draw a result. The results are compared and analysed with references. The Experiment section highlights the discount factor and exploration rate. It explains these concepts and analyses how they impact the model's performance by observing the experiments. Lastly, the report concludes by reflecting on the task.

Methods

This part of the report explains each Value Iteration and Q-learning. Then, based on the comparison of the two methods, the results of the actual cases of both methods are discussed.

- Value Iteration

Referring to Carl(2023), the Markov Decision Process(MDP) is a mathematical model representing a sequential decision-making process involving agents' interactions with the environment. Reinforcement Learning (RL) wants to solve the MDP problem to find the optimal policy. The solution process differs depending on whether a model exists. Information about the transition and reward functions should be included to have a model(Environment). The problem will be solved with Dynamic Programming(DP) using the Value Iteration (VI) algorithm if a model exists.
In the VI, the Bellman equation updates the value estimates for each state(value function). Iterating over the update provides the optimal value function that leads to policy (Carl, 2023).

- Q-Learning

In contrast, Q-Learning is a Model-Free algorithm that does not require a predefined model(Environment). Instead, the agent learns the optimal actions by trial and error by exploring the given environment (Watkins & Dayan, 1992).
The algorithm maximises the long-term reward of each action in the environment, which varies between exploration and exploitation. Specifically, the agent evaluates every action as a Q-value(action-value) using Bellman's equation and updates it based on the feedback to get the optimal policy (Saxena, 2024). The Greedy Policy balances these actions. According to Saxena (2024), this policy allows agents to explore new strategies in the first half of learning and gradually decrease the exploration rate to follow the optimised action in the second half.

To sum up, the value Iteration algorithm rapidly and accurately draws the optimal policy as there is a predefined model. In contrast, Q-learning allows agents to learn when there is no information about the environment. However, it needs enough learning and could show a move to the unoptimised path in the exploration process.

As referred, the difference was also spotted in the actual case.
Figures 5 and 6 are each Value Iteration and Q-Learning algorithm applied models.

By comparing the two figures, it was found that the Value Iteration algorithm applied model result performed better than the Q-Learning algorithm applied model. Figures 6 and 7 show that the arrow points to the shortest path to avoid the wall and obstacle. However, the difference comes from the other path used once the agent falls into the wrong path. The marked path in Figure 7 is inefficient or pointing in the wrong direction compared to Figure 6.

This finding allowed us to assume that the model-free feature of Q-learning caused the difference. As previously explained, the model learns through trial and error. If there is a lack of learning, the Q-value in the Q-table might be inaccurate.
In another case, it might occur through the greed policy. Applying epsilon = 0.4(exploration rate) might lead to less exploration on other paths. Therefore, according to the previous explanation, the model was less accurate and optimised in less learned paths.

Figure 6. Value Iteration

```
Pi: [1 2 2 2 1 2 2 2 1 2 2 0 0 0 1 2 2 2 0 1 1 1 1 0 1 2 1 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 1 2 1 3 2 3 3 3 2 3 3 3 0 1 2 0 0 1 2 3 2 2 0 3 2 2 0 3 2 0 0 0
 1 1 3 0 0 0 1 2 2 2 2 0 0 0 3 2 2 0 0 0 0 0 0 0 1 2 3 3 0 3 2 1 0 0 1 2 2 2
 1 2 3 1 3 0 3 0 1 0 1 0 0 0 1 1 2 1 1 1 1 0 1 0 1 0 0 1 1 2 2 2 0 0 1 2 2
 0 1 0 0 1 0 0 1 2 2 1 1 1 1 1 0 1 2 2 2 2 0 1 2 0 0 0 0 1 2 1 2 2 1 0 1 1
 1 1 1 1 0 0 0 0 0 0 0 0 1 2 0 0 1 2 2 2 2 2 2 0 1 3 1 1 3 0 0 0 0 3 2 0 0 0 0
 1 2 2 2 2 0 3 2 2]
```

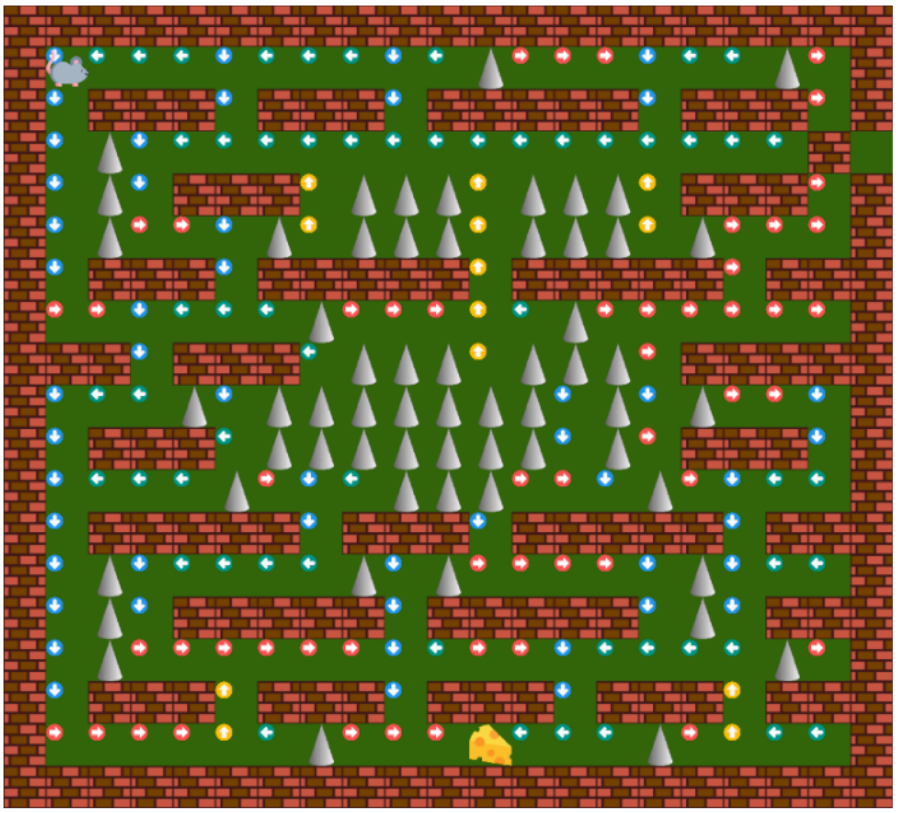
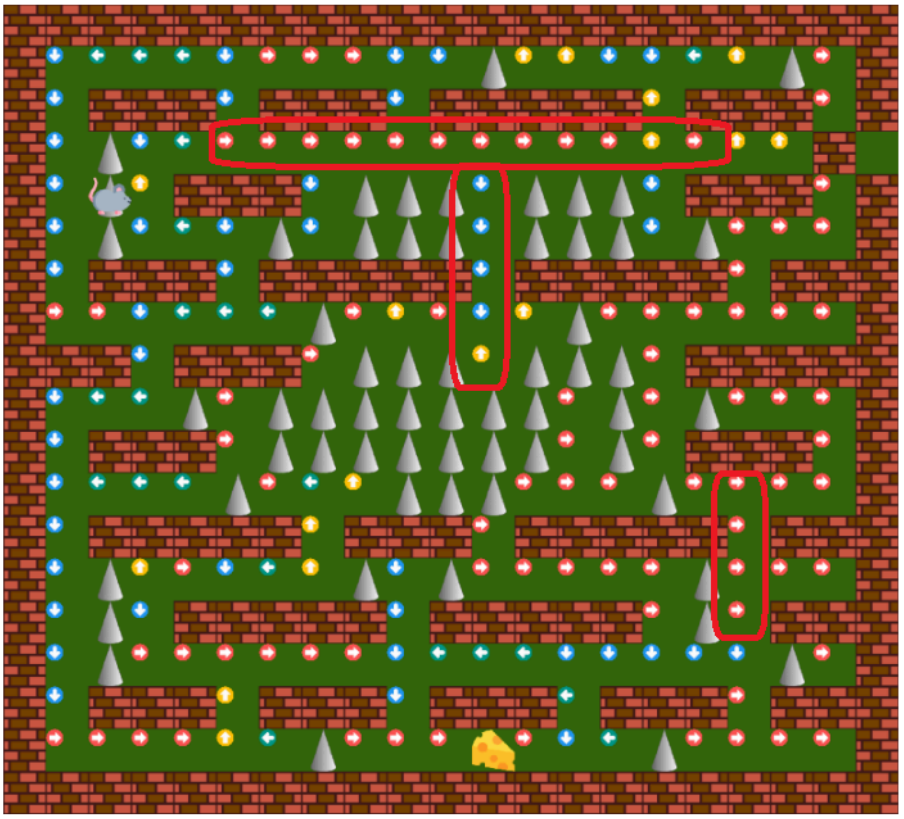


Figure 7. Q-Learning

```
Pi: [1 2 2 2 1 0 0 0 1 1 0 3 3 1 1 2 3 0 0 1 1 1 3 0 1 0 1 2 0 0 0 0 0 0 0 0 0
 0 3 0 3 3 1 0 3 1 0 0 0 1 0 0 0 1 0 1 0 1 2 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0
 1 1 1 0 0 0 1 2 2 2 0 0 3 0 1 3 0 0 0 0 0 0 0 1 0 0 0 0 3 0 0 0 0 1 2 2 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 2 2 2 0 0 2 3 0
 0 0 0 0 0 0 0 0 0 0 0 1 3 0 0 1 0 3 0 1 2 3 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 1
 0 0 0 1 0 0 0 0 0 0 0 0 1 2 2 2 1 1 1 1 1 0 0 1 3 1 2 0 0 0 0 0 3 2 0 0 0 0
 0 0 1 2 0 0 0 0 0]
```



Experiments

In this part of the report, the discount factor and exploration rate of Q-Learning will be explained and investigated. Furthermore, based on the experiment, applying the discount factor and Q-Learning will be discussed to find the impact on the model performance.

As mentioned, the exploration rate controls how often new actions are tried. The following figures show the results of applying high (epsilon = 0.9) and low(epsilon = 0.1) exploration rates to understand the impact better.

In Figure 8, the model result seems less optimised and accurate when the high exploration rate is applied. Due to the increase in the exploration rate, the exploration action is prioritised over the best-known action. In contrast, according to Figure 9, the low exploration rate resulted in high accuracy and efficiency in the optimised path by focusing on action with a high q-value. However, the accuracy of the other path in Figure 9 decreased more than that of Figure 7, with a medium exploration rate applied due to less exploration of different paths during the learning.

According to Or(2020), the discount factor determines the importance of long-term and short-term rewards. In the model coding, it is set by modifying the gamma. If the gamma increases, it will prioritise long-term rewards. In the opposite setting, short-term rewards will

be prioritised. Figures 10 and 11 show the results of each high and low discount factor. According to the comparison of the results, the high discount factor showed higher performance, while the low discount factor showed poor performance. Prioritising short-term rewards more than necessary led to choosing an inefficient path from a long-term perspective.

Experiments on exploration rate and discount factor revealed that finding a balance in both hyperparameters is crucial by rapidly testing the model to achieve the optimal result.

Figure 8. Exploration Rate: 0.9

```
Pi: [0 0 0 0 1 0 0 0 1 2 0 0 3 1 3 3 1 0 0 1 1 1 3 0 1 0 0 0 0 0 0 0 0 1 0 0
 0 3 2 0 3 1 0 3 3 0 0 0 1 0 0 0 1 0 1 0 3 0 1 0 3 0 0 0 1 0 0 0 1 0 0 0 0
 1 1 1 0 0 0 1 2 2 2 0 0 1 2 1 2 0 0 0 0 0 0 0 1 0 0 0 0 2 0 0 0 0 1 2 2 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 2 2 2 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 2 3 1 1 2 0 0 0 0
 0 0 0 0 0 0 0 0 0]
```

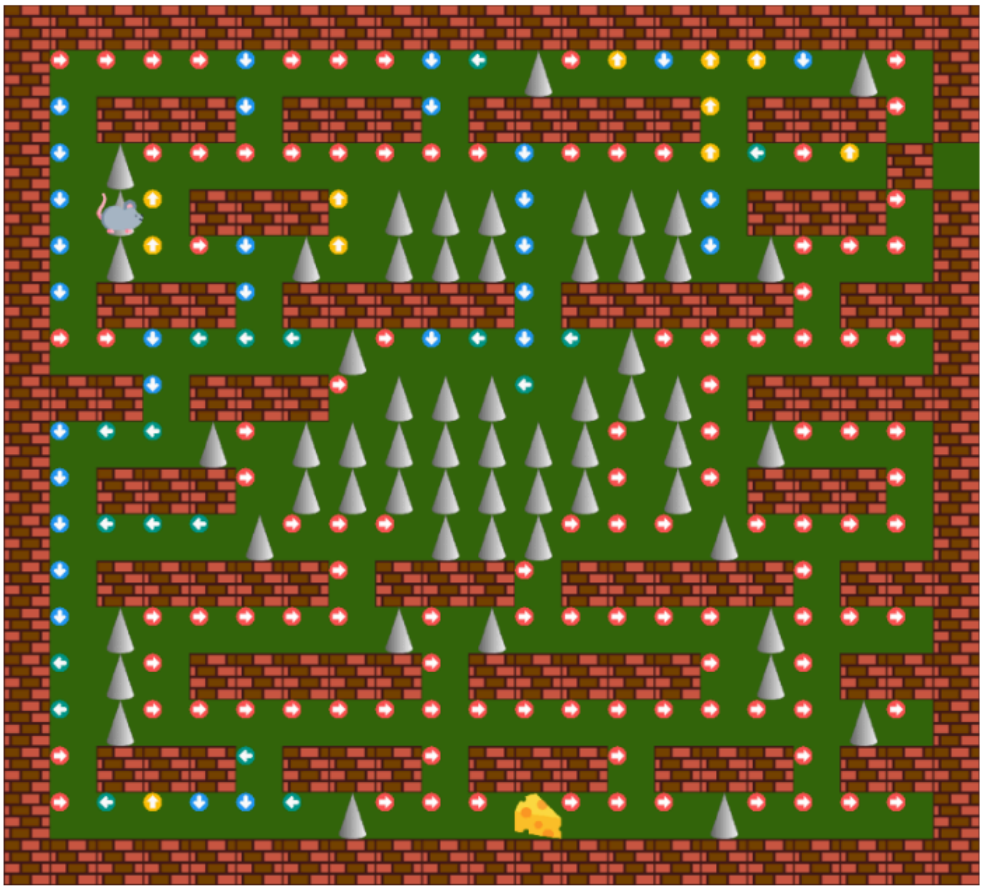


Figure 9. Exploration Rate:0.1

```
Pi: [1 2 2 0 2 0 3 0 2 3 0 1 2 2 3 3 3 0 0 1 2 3 3 0 1 0 3 1 3 1 3 3 2 0 1 2 3
 2 0 1 3 3 1 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1 2 2 0 1 0 0 0 1 0 0 0 1 0 0 0 0
 1 0 2 0 0 0 1 2 2 1 0 3 2 3 2 3 0 0 0 0 0 0 0 1 0 0 0 0 3 0 0 0 0 1 2 2 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 2 2 3 0 1 0 1 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 3 1 3 1 2 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1
 1 0 0 1 0 0 0 0 0 0 0 1 2 2 0 1 2 2 0 1 0 0 1 3 1 0 0 0 0 0 0 3 2 0 0 0 0
 0 2 0 3 0 0 0 0 0]
```

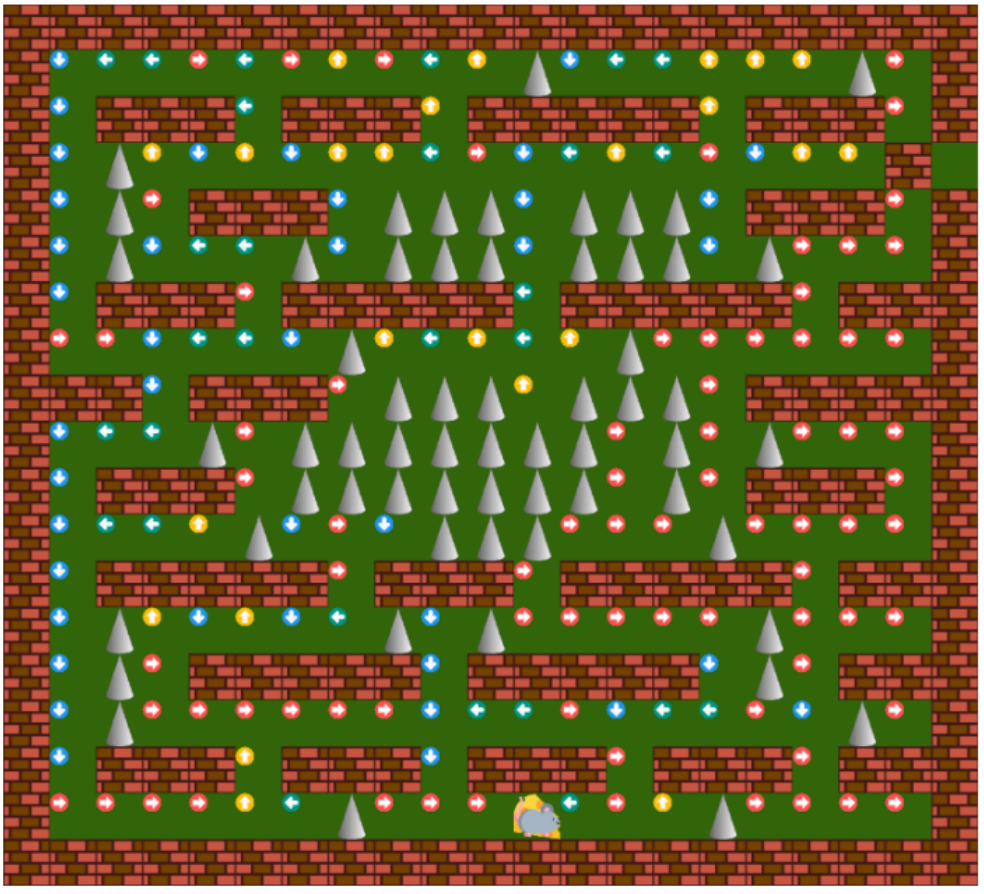


Figure 10. gamma = 0.9

```
Pi: [1 2 2 2 2 0 0 0 1 1 0 1 3 2 3 2 3 0 0 1 1 1 3 0 1 0 3 1 0 0 0 0 0 0 1 0 0
 0 3 0 1 0 1 0 0 2 0 0 0 1 0 0 0 1 0 1 0 1 2 1 0 3 0 0 0 1 0 0 0 3 0 0 0 0
 1 1 0 0 0 0 1 2 2 2 0 3 3 0 3 0 0 0 0 0 0 0 1 0 0 0 0 3 0 0 0 0 1 2 2 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 2 2 2 0 1 2 2 0
 0 0 0 0 0 0 0 0 0 0 1 2 0 0 1 0 0 0 3 2 3 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 1
 1 0 0 1 0 0 0 0 0 0 0 1 2 2 2 2 2 1 1 2 0 0 1 3 1 3 1 0 0 0 0 3 2 0 0 0 0
 0 0 1 1 0 0 0 0 0]
```

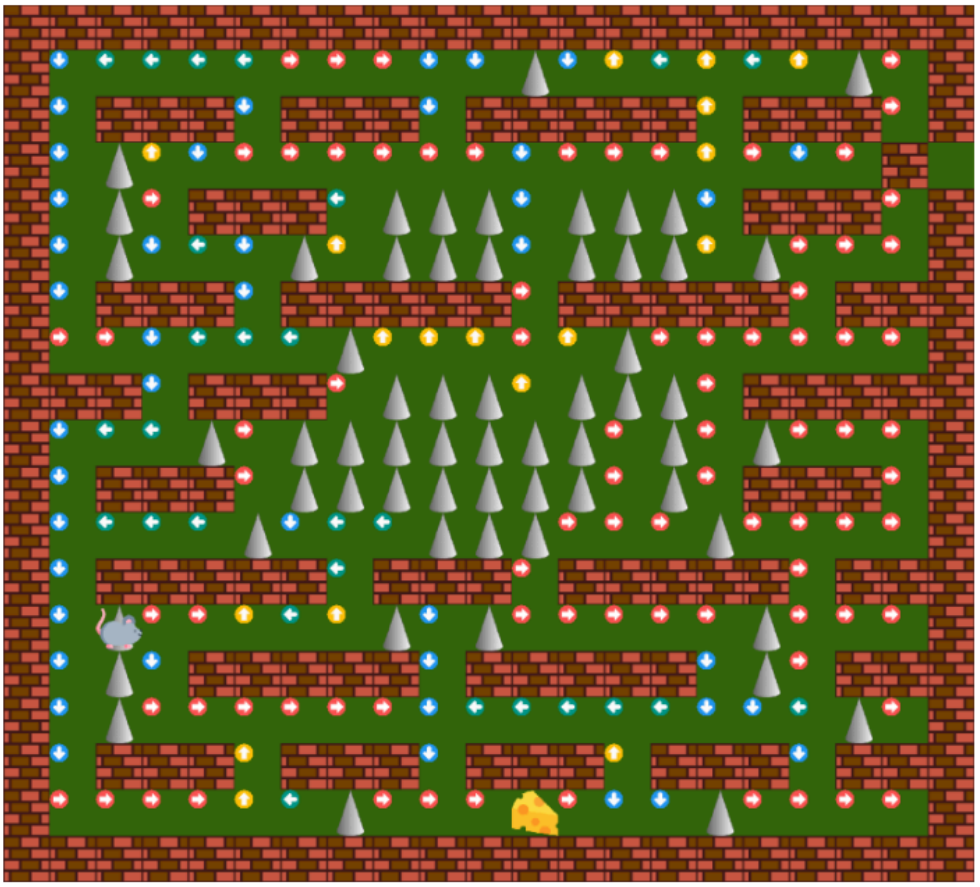


Figure 11. gamma = 0.1

```
Pi: [0 0 0 0 0 0 0 0 0 1 0 3 2 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 1 0 3 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 0 0 2 3 2 3 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 1 0 2 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 1 3 3 2 0 1 0 1 0
 0 0 0 0 0 0 0 0 0 0 2 0 0 0 1 0 3 3 3 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 3 2
 0 0 0 2 0 1 1 0 2 0 1 2 3 1 1 2 1 1 2 1 0 0 3 2 1 2 0 1 3 0 1 3 3 0 1 0 0
 0 0 1 1 0 0 0 0 0]
```

Reflection

This report's comparative analysis of Value Iteration and Q-Learning highlights significant differences in their approaches and effectiveness. With its dependency on a pre-defined model, Value Iteration has demonstrated quicker convergence to the optimal policy, showcased by its superior performance in structured environments. In contrast, the model-free nature of Q-Learning allows it to operate in environments lacking explicit models but at the cost of potential inefficiencies, as seen in the experimental results. The experiments further suggest that the balance of exploration and exploitation, controlled by parameters like the exploration rate and discount factor, plays a critical role in the effectiveness of Q-learning. These findings underscore the importance of parameter tuning and the inherent trade-offs between long-term and short-term rewards in reinforcement learning algorithms. Understanding and optimising these parameters can significantly enhance the performance of RL agents by being robust and adaptable in varied and unpredictable environments.

Reference

Bhamre, S. (2025, March 10). VGG19. https://www.kaggle.com/code/sumedhbhamre/vgg19

Carl. (2023, September 10). *Reinforcement Learning: an Easy Introduction to Value Iteration | by Carl | TDS Archive | Medium*. https://medium.com/towards-data-science/reinforcement-learning-an-easy-introduction-to-value-iteration-e4cfe0731fd5

Grigoryan, A. A. (2023, August 13). *Understanding VGG Neural Networks: Architecture and Implementation | by Anna Alexandra Grigoryan | Medium*. https://thegrigorian.medium.com/understanding-vgg-neural-networks-architecture-and-implementation-400d99a9e9ba

Longils. (2020, July 10). *[*비전공자용*] [Python]* 배치 정규화 *Batch Normalization*. https://huangdi.tistory.com/9

Olamendy, C. J. (2023, December 4). *Understanding ReLU, LeakyReLU, and PReLU: A Comprehensive Guide | by Juan C Olamendy | Medium*. https://medium.com/@juanc.olamendy/understanding-relu-leakyrelu-and-prelu-a-comprehensive-guide-20f2775d3d64

Or, B. (2020, August 21). *Penalizing the Discount Factor in Reinforcement Learning | by Dr Barak Or | TDS Archive | Medium*. https://medium.com/towards-data-science/penalizing-the-discount-factor-in-reinforcement-learning-d672e3a38ffe

Osetrov, E. (2024, December 12). *Data Augmentation in Python: Everything You Need to Know*. https://neptune.ai/blog/data-augmentation-in-python

Saxena, A. (2024, December 12). *Q Learning in Machine Learning [Explained by Experts]*. https://www.appliedaicourse.com/blog/q-learning-in-machine-learning/

Simonyan, K., & Zisserman, A. (2015). Why is the VGG Network Commonly Used? *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. https://nnart.org/why-is-vgg-commonly-used/

towardsdatascience. (n.d.). *Introduction to SELUs*. Towardsdatascience. Retrieved March 12, 2025, from https://towardsdatascience.com/gentle-introduction-to-selus-b19943068cd9

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3–4), 279–292. https://doi.org/10.1007/BF00992698

Figure

Lim, D. (2020, October 14). *[VGGNet] VGGNet* 개념 정리. https://daechu.tistory.com/10