COP528 AI and Applied Machine Learning Coursework

Task 1:

Introduction

Machine learning has become a transformative technology with applications in computer vision, NLP, and healthcare. A key application is image classification, which categorizes images based on visual content, with real-world uses in autonomous vehicles, medical imaging, and object recognition. This coursework focuses on image classification using a dataset containing objects like parachutes, oil boxes, and trucks. We develop a Convolutional Neural Network (CNN) to accurately classify images, leveraging CNNs' ability to extract hierarchical features. We also explore data augmentation and model enhancements to improve performance. The report covers the problem background, CNN model design, experimental evaluation, and key findings, demonstrating a practical application of machine learning.

Approach

A Convolutional Neural Network (CNN) was chosen for image classification due to its ability to automatically learn spatial hierarchies of features from raw pixel data, making it highly effective for object recognition.

Justification for Using CNNs

- **Feature Extraction**: CNNs eliminate the need for manual feature engineering by learning relevant features directly from images.
- **Spatial Invariance**: Pooling layers enable recognition of objects regardless of position or orientation.
- **Hierarchical Learning**: CNNs capture low-level features (edges, textures) in early layers and high-level features (object parts, full objects) in deeper layers, aligning well with the dataset's complexity.

Key Design Choices

- **Input Size (224x224x3)**: Standardized size for consistency and efficient feature extraction; RGB channels retained for colour information.
- **Convolutional Layers**:
    - **Conv1 (32 filters, 3x3, ReLU)**: Captures low-level features (edges, textures).
    - **Conv2 (64 filters, 3x3, ReLU)**: Extracts higher-level features (object parts).
- **Pooling Layers (MaxPooling2D, 2x2)**: Reduces spatial dimensions, lowers computational cost, and ensures translation invariance.
- **Flatten Layer**: Converts 2D feature maps into a 1D vector for classification.

- **Fully Connected Layers**:
    - **Dense (128 units, ReLU)**: Learns complex feature combinations.
    - **Output Layer (num_classes, linear activation)**: Produces logits for classification.
- **Loss Function (Sparse Categorical Crossentropy)**: Suitable for multi-class classification with integer labels.
- **Optimizer (Adam)**: Efficient learning with adaptive rates and momentum for faster convergence.

Layer Type	Parameters	Output Shape
Input Layer	Image size: 224x224x3	(224, 224, 3)
Conv2D	32 filters, 3x3 kernel, ReLU activation	(222, 222, 32)
MaxPooling2D	2x2 pool size	(111, 111, 32)
Conv2D	64 filters, 3x3 kernel, ReLU activation	(109, 109, 64)
MaxPooling2D	2x2 pool size	(54, 54, 64)
Flatten	-	(54 * 54 * 64 = 186624)
Dense	128 units, ReLU activation	(128)
Output Layer	`num_classes` units, linear activation	(`num_classes`)

Table 1. Layer type, it's parameter and the shape of its output

Evaluation protocol:

1. **Dataset Splitting**: The dataset was already split into **training** (for learning) and **validation** (for tuning hyperparameters) sets, ensuring performance is assessed on unseen data to prevent overfitting.

2. **Performance Metrics**: The following metrics were used to evaluate the model:

- Accuracy: The proportion of correctly classified. Primary metric for assessing overall performance.
- Loss: Tracks how well predictions align with true labels.
- Confusion Matrix: Identifies misclassified classes.
- Classification Report: Provides precision, recall, and F1-score for a detailed class-wise evaluation.

Accuracy alone can be misleading, so additional metrics ensure a comprehensive assessment.

3. **Learning Curves**

- Training vs. Validation Loss
- Training vs. Validation Accuracy

This Detects overfitting (validation loss increasing while training loss decreases) or underfitting (both losses remain high).

Experiment

```
Epoch 1, Loss: 1.7640, Accuracy: 41.41%, Validation Loss: 1.5563, Validation Accuracy: 49.63%
Epoch 2, Loss: 0.9697, Accuracy: 68.81%, Validation Loss: 1.2769, Validation Accuracy: 60.56%
Epoch 3, Loss: 0.4219, Accuracy: 86.98%, Validation Loss: 1.6784, Validation Accuracy: 56.89%
Epoch 4, Loss: 0.1540, Accuracy: 95.53%, Validation Loss: 2.1175, Validation Accuracy: 55.44%
Epoch 5, Loss: 0.0947, Accuracy: 97.26%, Validation Loss: 2.2474, Validation Accuracy: 55.95%
```

Figure 1 - Accuracy and loss of each epoch for both training and validation

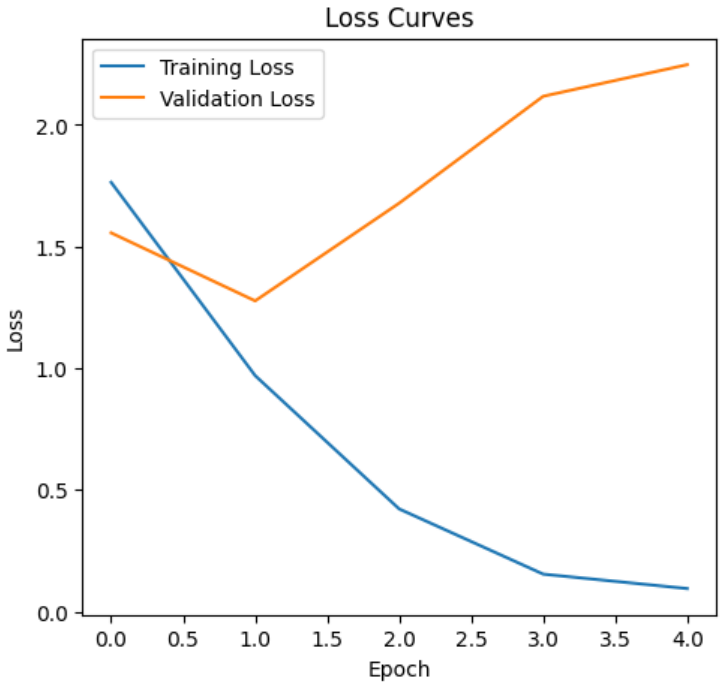

Figure 2 - Epoch loss curve for visualisation

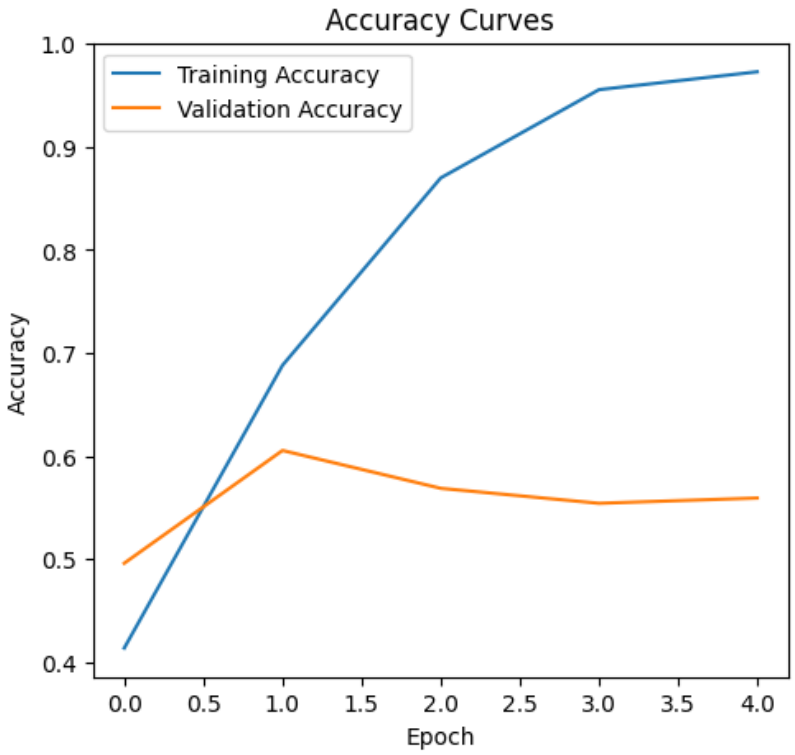

Figure 3 – Epoch accuracy curve for visualisation

Classification Report:

	precision	recall	f1-score	support
n01440764	0.62	0.66	0.64	387
n02102040	0.53	0.63	0.57	395
n02979186	0.58	0.60	0.59	357
n03000684	0.34	0.35	0.34	386
n03028079	0.61	0.67	0.64	409
n03394916	0.50	0.61	0.55	394
n03417042	0.61	0.59	0.60	389
n03425413	0.53	0.40	0.46	419
n03445777	0.54	0.52	0.53	399
n03888257	0.80	0.59	0.68	390
accuracy			0.56	3925
macro avg	0.57	0.56	0.56	3925
weighted avg	0.57	0.56	0.56	3925

Figure 4 - Classification Report

Key Observations:

- The model achieved high training accuracy but struggled to generalize to the validation set, as evidenced by the increasing validation loss and stagnant validation accuracy.
- Overfitting became evident after the second epoch, as the model memorized the training data instead of learning generalizable features.

2. Classification Report Key Observations:

- The model performed best on class n03888257 (F1-score: 0.68) and worst on class n03000684 (F1-score: 0.34).
- The low precision and recall for some classes (e.g., n03000684) suggest that the model struggled to distinguish these classes from others.

Discussion of Findings

- **Overfitting**: The model exhibited clear signs of overfitting, as evidenced by the high training accuracy and low validation accuracy. This suggests that the model memorized the training data instead of learning generalizable features.
- **Class Imbalance**: The variability in precision, recall, and F1-scores across classes indicates potential class imbalance or insufficient representation of certain classes in the training data.
- **Model Complexity**: The CNN architecture, while effective, may be too complex for the dataset, leading to overfitting. Simplifying the model or adding regularization techniques (e.g., dropout, weight decay) could improve generalization.

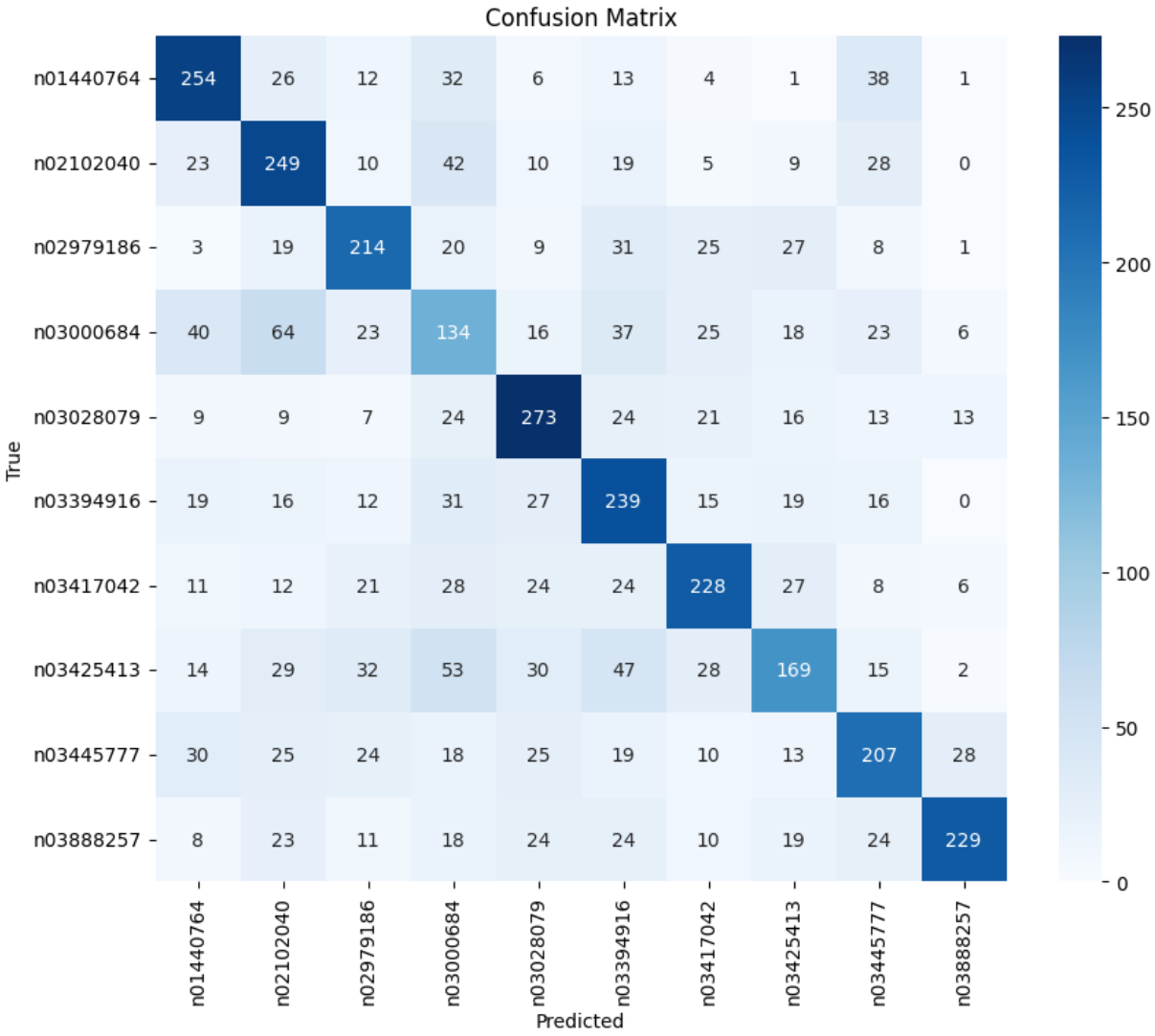

Figure 5 – Confusion matrix at each class

Further improvements to models' performance

To enhance CNN performance, data augmentation, Dropout, L2 regularization, and early stopping were implemented. Each method is justified and its impact analysed.

1. Data Augmentation

Enhancements:

- RandomFlip (horizontal/vertical) – simulates orientation changes.
- RandomRotation (±20%) – accounts for object rotation.
- RandomZoom (±20%) – adjusts scale variations.

Justification: Expands dataset diversity, improving generalization by exposing the model to varied input conditions.

Impact: Reduced overfitting, improved validation accuracy, and enhanced robustness to image variations.

2. Dropout Layers

Implementation: Added a Dropout layer (0.5) after the dense layer, randomly deactivating 50% of neurons during training.

Impact: Improved generalization, with a smaller training-validation accuracy gap.

3. L2 Regularization

Implementation: Applied L2 regularization (λ=0.001) to convolutional and dense layers, penalizing large weights.

Impact: Lower validation loss, indicating improved generalization.

4. **Early Stopping**

Implementation: Used Early Stopping (monitoring validation loss) with patience = 3 and best weight restoration.

Impact: Balanced training time and model performance, avoiding overfitting.

Experiment evaluation for improved version

```
Epoch 1, Loss: 1.8981, Accuracy: 35.32%, Validation Loss: 1.6623, Validation Accuracy: 44.20%
Epoch 2, Loss: 1.5433, Accuracy: 47.91%, Validation Loss: 1.4781, Validation Accuracy: 51.36%
Epoch 3, Loss: 1.4356, Accuracy: 51.70%, Validation Loss: 1.4577, Validation Accuracy: 51.59%
Epoch 4, Loss: 1.3724, Accuracy: 54.25%, Validation Loss: 1.4579, Validation Accuracy: 52.89%
Epoch 5, Loss: 1.3116, Accuracy: 55.47%, Validation Loss: 1.3894, Validation Accuracy: 55.29%
Epoch 6, Loss: 1.2558, Accuracy: 58.31%, Validation Loss: 1.4983, Validation Accuracy: 52.48%
Epoch 7, Loss: 1.2157, Accuracy: 59.48%, Validation Loss: 1.3534, Validation Accuracy: 55.03%
Epoch 8, Loss: 1.1848, Accuracy: 60.23%, Validation Loss: 1.3471, Validation Accuracy: 55.87%
Epoch 9, Loss: 1.1619, Accuracy: 60.90%, Validation Loss: 1.2334, Validation Accuracy: 58.32%
Epoch 10, Loss: 1.1426, Accuracy: 61.93%, Validation Loss: 1.4233, Validation Accuracy: 54.29%
Epoch 11, Loss: 1.1324, Accuracy: 62.44%, Validation Loss: 1.2575, Validation Accuracy: 58.47%
Epoch 12, Loss: 1.0903, Accuracy: 63.61%, Validation Loss: 1.2682, Validation Accuracy: 57.71%
Epoch 13, Loss: 1.0866, Accuracy: 63.93%, Validation Loss: 1.3222, Validation Accuracy: 57.38%
Epoch 14, Loss: 1.0770, Accuracy: 63.70%, Validation Loss: 1.2323, Validation Accuracy: 59.34%
Epoch 15, Loss: 1.0488, Accuracy: 65.23%, Validation Loss: 1.2535, Validation Accuracy: 58.37%
```

Figure 6 - Accuracy and loss of each epoch for both training and validation of enhanced method

The results from the improved CNN model show significant changes compared to the previous model.

- The previous model achieved a training accuracy of **97.26%** but a validation accuracy of only **55.95%**, with a large gap between training and validation metrics, indicating severe overfitting.
- The improved model achieved a lower training accuracy (**65.23%**) but a higher validation accuracy (**58.37%**), with a smaller gap between training and validation metrics. This suggests that the regularization techniques (dropout, L2 regularization) and data augmentation effectively reduced overfitting.

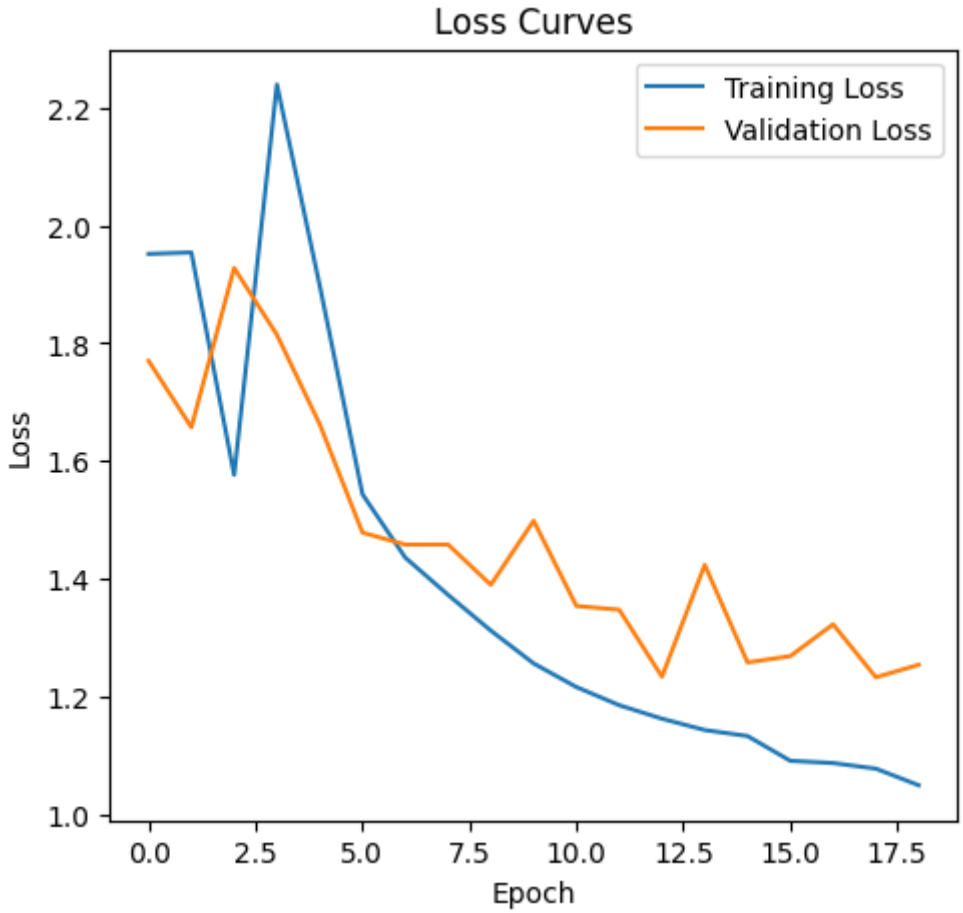

Figure 7 -Loss vs Epoch curve for improved version

2. Classification Report

- Precision: Ranged from 0.44 (n03425413) to 0.94 (n02102040), showing improved precision for some classes compared to the previous model.
- Recall: Ranged from 0.36 (n02102040) to 0.81 (n03888257), indicating better recall for certain classes.
- F1-score: Ranged from 0.41 (n03000684) to 0.73 (n01440764), reflecting a better balance between precision and recall for most classes.
- Overall Accuracy: The model achieved an accuracy of 58% on the validation set, slightly higher than the previous model's 56%.

Comparison to Previous Model:

- The previous model had an F1-score range of **0.34** to **0.68**, while the improved model achieved a range of **0.41** to **0.73**, indicating better overall performance.
- The improved model showed higher precision and recall for several classes, such as **n01440764** (F1-score: **0.73**) and **n03888257** (F1-score: **0.69**), compared to the previous model.

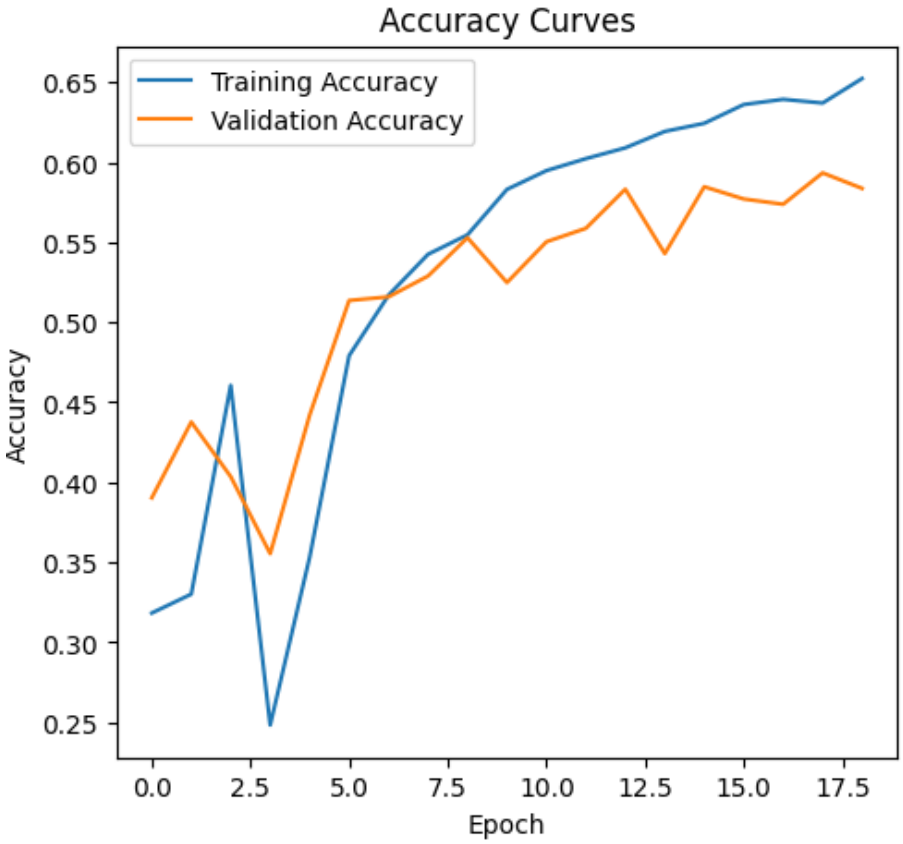

Figure 8 – Accuracy vs Epoch curve for improved version

Classification Report:

	precision	recall	f1-score	support
n01440764	0.81	0.66	0.73	387
n02102040	0.94	0.36	0.52	395
n02979186	0.71	0.59	0.65	357
n03000684	0.48	0.36	0.41	386
n03028079	0.56	0.64	0.60	409
n03394916	0.63	0.58	0.60	394
n03417042	0.49	0.66	0.56	389
n03425413	0.44	0.55	0.49	419
n03445777	0.53	0.63	0.58	399
n03888257	0.61	0.81	0.69	390
accuracy			0.58	3925
macro avg	0.62	0.58	0.58	3925
weighted avg	0.62	0.58	0.58	3925

Figure 9 – Classification report for improved version

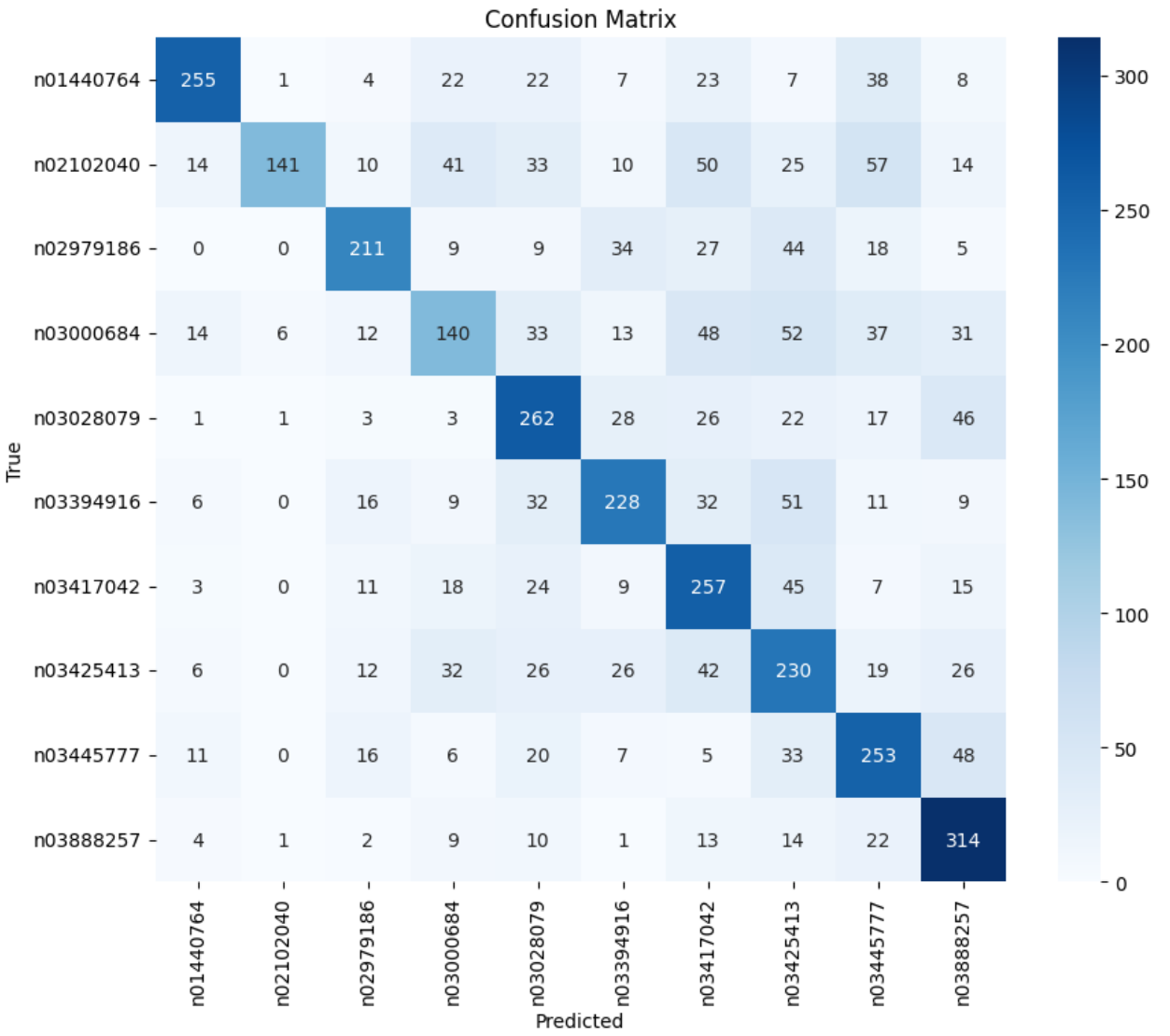

Figure 10 – Confusion matrix for improved version

Discussion of Findings

- Reduced Overfitting: The improved model showed a smaller gap between training and validation metrics, indicating that the regularization techniques and data augmentation effectively reduced overfitting.
- Better Generalization: The improved model achieved higher validation accuracy (58.37%) compared to the previous model (55.95%), demonstrating better generalization to unseen data.
- Class-Specific Performance: The improved model showed higher precision, recall, and F1-scores for several classes, indicating that it learned more robust features and performed better on challenging classes.
- Trade-offs: While the improved model achieved better generalization, it required more epochs to converge and had a lower training accuracy, reflecting the trade-off between model complexity and generalization.

Future work could focus on further fine-tuning hyperparameters, exploring advanced augmentation techniques, or using more complex architectures to achieve even better results.

Task 2:

Introduction

This task involves solving a Gridworld problem using two reinforcement learning methods: Value Iteration and Q-Learning. The Gridworld contains walls (w), obstacles (o), and a goal (g). The agent starts at the top-left corner and must navigate to the goal while avoiding obstacles. The goal is to find an optimal policy that maximizes cumulative rewards.

- Value Iteration: A model-based method that computes the optimal value function and derives the policy. It requires knowledge of the environment's dynamics.
- Q-Learning: A model-free method that learns the optimal policy by updating a Q-table through exploration and exploitation. It does not require prior knowledge of the environment.

Both methods are applied to the Gridworld, and their performance is evaluated based on their ability to find the optimal policy.

Methods

1. Value Iteration

How It Works: Value Iteration is an iterative algorithm that computes the optimal value function V∗(s)$V*(s)$ for each state ss. The value function represents the expected cumulative reward when starting from state ss and following the optimal policy. The algorithm updates the value function using the Bellman Optimality Equation.

Implementation Details:

1. **Initialization**: The value function VV is initialized to zero for all states.
2. **Iteration**: The value function is updated iteratively until convergence (when the change in VV is below a small threshold ϵ=10−7ϵ=10−7).
3. **Policy Extraction**: Once the value function converges, the optimal policy ππ$\pi\pi$ is derived by selecting the action that maximizes the expected cumulative reward for each state.

Justification:

- Value Iteration is chosen because it is a model-based method that guarantees convergence to the optimal policy. It is suitable for environments where the transition probabilities and rewards are known.

2. Q-Learning

How It Works: Q-Learning is a model-free RL method that learns the optimal policy by iteratively updating a Q-table. The Q-table Q(s,a)$Q(s,a)$ represents the expected cumulative reward for taking action aa in state ss and following the optimal policy thereafter. The algorithm uses the Bellman Equation to update the Q-values.

Implementation Details:

1. **Initialization**: The Q-table is initialized to zero for all state-action pairs.
2. **Exploration vs. Exploitation**: The agent follows an epsilon-greedy policy to balance exploration and exploitation. With probability ϵ=0.4ϵ=0.4, the agent takes a random action (exploration); otherwise, it takes the action with the highest Q-value (exploitation).

3. **Q-Value Update**: The Q-values are updated iteratively using the observed rewards and the maximum Q-value of the next state.
4. **Policy Extraction**: After training, the optimal policy $\pi\pi$ is derived by selecting the action with the highest Q-value for each state.

Justification:

- Q-Learning is chosen because it is a model-free method that does not require prior knowledge of the environment's dynamics. It learns the optimal policy through trial and error, making it suitable for environments where the transition probabilities and rewards are unknown.
- The epsilon-greedy policy ensures a balance between exploration and exploitation, allowing the agent to discover the optimal policy while avoiding suboptimal solutions.

Experiments

Figure 11 – Value iteration policy graph

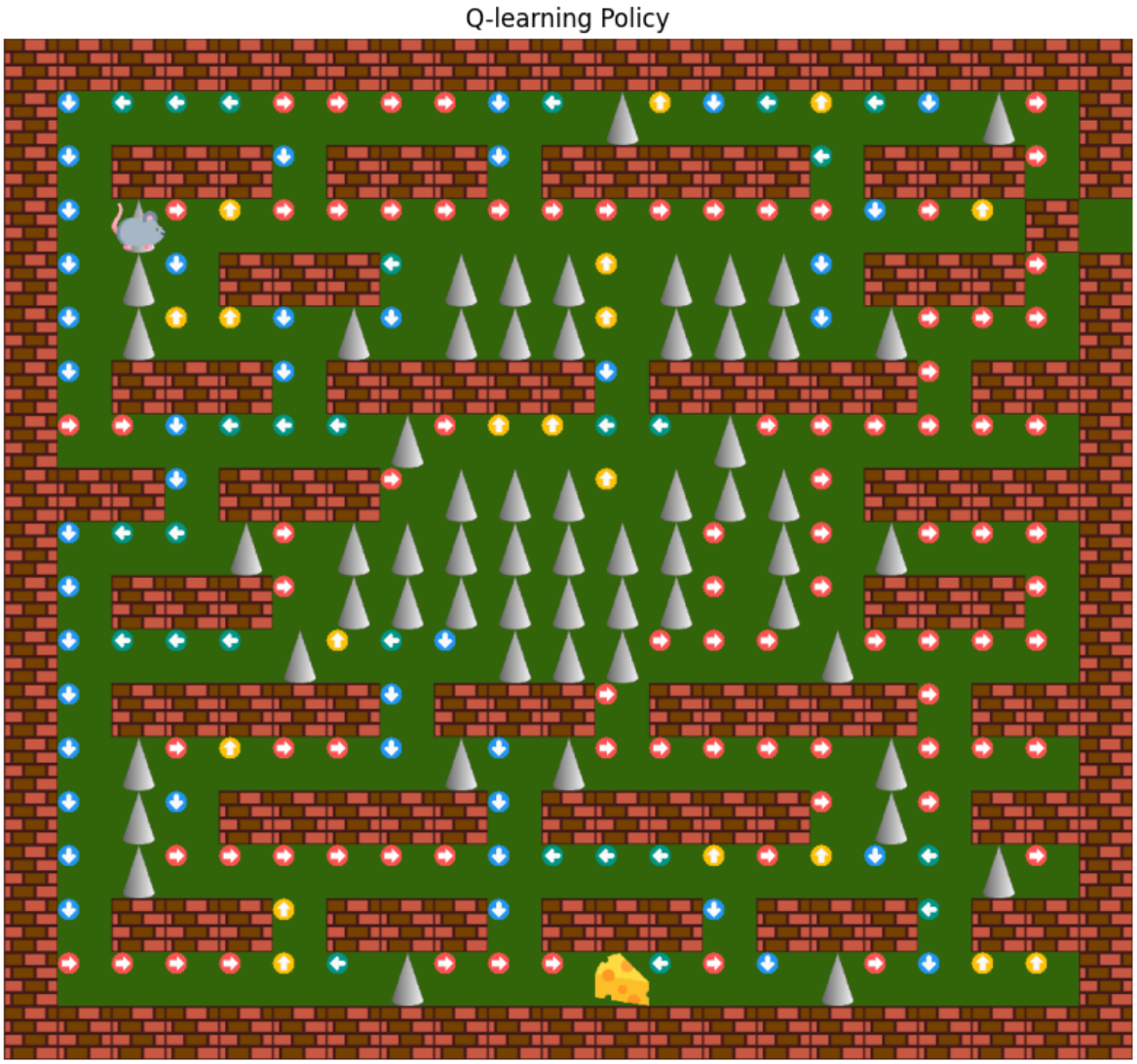

Figure 12 – Q-learning policy graph

1. Value Iteration

- **Convergence**: The Value Iteration algorithm converged in **250 iterations** with a convergence time of **0.0836 seconds**.
- **Policy**: The derived policy was visualized on the Gridworld, showing the optimal path from the start to the goal while avoiding obstacles and walls.
- **Performance**: The average reward over 100 episodes was **-125.00**, indicating that the policy needs further refinement to improve performance.

2. Q-Learning

- **Convergence**: The Q-Learning algorithm converged after **10,000 episodes** with a convergence time of **22.1670 seconds**.
- **Policy**: The derived policy was visualized on the Gridworld, showing the optimal path from the start to the goal while avoiding obstacles and walls.
- **Performance**: The average reward over 100 episodes was **67.00**, demonstrating better performance compared to Value Iteration.

Aspect	Value Iteration	Q-Learning
Convergence Time	0.0836 seconds	22.1670 seconds
Iterations/Episodes	250 iterations	10,000 episodes
Average Reward	-125.00	67.00
Policy Quality	Suboptimal, needs refinement	Optimal, achieves higher rewards
Exploration	Not required (assumes full knowledge)	Required (epsilon-greedy policy)
Use Case	Known environments	Unknown or complex environments

Table 2 – Method comparison

Analysis:

- **Convergence Time**: Value Iteration converged significantly faster than Q-Learning. This is expected because Value Iteration is a model-based method that directly computes the optimal value function using known environment dynamics.
- **Policy Quality**: Q-Learning achieved a higher average reward (67.00) compared to Value Iteration (-125.00). This suggests that Q-Learning's exploration strategy (epsilon-greedy) allowed it to discover a more effective policy.
- **Exploration**: Q-Learning's ability to explore the environment and learn from interactions made it more robust in finding an optimal policy, especially in complex or unknown environments.
- **Performance**: The negative average reward for Value Iteration indicates that the derived policy may not be optimal or that the reward structure needs adjustment. In contrast, Q-Learning's positive average reward demonstrates its effectiveness in maximizing cumulative rewards.

Discussion

Value Iteration converged quickly (250 iterations, 0.0836s) but yielded a suboptimal policy with an average reward of -125. The policy graph suggests ineffective obstacle avoidance, likely due to misaligned rewards or transition probabilities. Adjusting the reward function, such as increasing penalties for moving toward obstacles, could improve performance.

Q-Learning, though slower, achieved a significantly higher average reward of 67.00. Its policy graph shows better pathfinding, with the agent reliably avoiding obstacles and reaching the goal. This improvement stems from Q-Learning’s model-free learning and epsilon-greedy exploration, allowing for better policy discovery. The cumulative reward graph confirms steady learning progress. In summary, Q-Learning outperformed Value Iteration in this task, achieving better rewards through exploration and adaptability. While Value Iteration's speed makes it suitable for well-defined problems, its reliance on accurate modelling can limit performance, as seen in the suboptimal policy graph. Future work could refine the reward structure for Value Iteration and optimize Q-Learning's hyperparameters for further improvements.

Part C:

To investigate the effects of the discount factor (γ) and exploration rate (ε) on the performance of Q-learning, a systematic experimental approach was implemented. The goal was to analyse how different combinations of these parameters influence the agent's learning process, policy quality, and overall

1. Parameter Selection

- Discount Factor (γ): Five values of γ were tested: [0.1, 0.5, 0.7, 0.9, 0.99]. These values represent a range from short-term to long-term reward focus.
- Exploration Rate (ε): Five values of ε were tested: [0.1, 0.3, 0.5, 0.7, 0.9]. These values represent a range from low to high exploration.

2. Q-learning Implementation

For each combination of γ and ε, the Q-learning algorithm was executed with the initialization and training Loop

1. Metrics Calculation:
    - Convergence Time: The time taken to complete 10,000 episodes.
    - Average Reward: The mean cumulative reward across all episodes.
    - Success Rate: The percentage of episodes where the agent reached the goal.
    - Average Episode Length: The mean number of steps taken per episode.
    - Final Q-value Variance: The variance of the Q-values at the end of training, indicating the stability of the learned policy.

3. Visualisation

- Success Rate vs. Episodes: The cumulative success rate over episodes was plotted for each combination of γ and ε to observe how quickly the agent learned to reach the goal.
- Average Episode Length vs. Episodes: The average episode length over episodes was plotted to analyse how efficiently the agent navigated the Gridworld.

γ	ε	Convergence Time (s)	Average Reward	Success Rate (%)	Avg Episode Length (steps)	Final Q-value Variance
0.1	0.1	47.16	-175.89	0.11	79.75	394.24
0.1	0.3	41.43	-158.83	0.05	63.19	455.29
0.1	0.5	29.11	-138.72	0.03	40.53	453.88
0.1	0.7	27.15	-131.58	0.01	32.60	442.63
0.1	0.9	31.37	-136.85	0.00	37.85	320.95
0.5	0.1	22.48	25.47	79.78	35.09	410.26
0.5	0.3	22.63	-42.25	44.18	31.61	506.47
0.5	0.5	20.75	-96.53	14.76	27.05	520.43

Table 3 - numeric results for effects analysis

1. Effect of Discount Factor (γ)

The discount factor determines the importance of future rewards. A higher γ values future rewards more heavily, while a lower γ focuses on immediate rewards.

- **Low γ (0.1)**: Poor performance across all ε values, with average rewards ranging from **-175.89** to **-131.58** and success rates below **0.11%**. A low γ causes the agent to prioritize immediate rewards, which is ineffective in environments where long-term planning is required to reach the goal. The agent fails to learn a meaningful policy.
- **Moderate γ (0.5, 0.7, 0.9)**: Significant improvement in performance, especially for low ε values. For example, with γ=0.5 and ε=0.1, the average reward is **25.47**, and the success rate is **79.78%**. A moderate γ balances immediate and future rewards, enabling the agent to learn effective policies. The agent can navigate the Gridworld efficiently, as shown by the higher success rates and lower episode lengths.
- **High γ (0.99)**: Like moderate γ values, with high success rates (e.g., **80.34%** for γ=0.99 and ε=0.1) and low episode lengths. A high γ emphasizes long-term rewards, which is beneficial in this environment. However, the performance is comparable to moderate γ values, suggesting diminishing returns for very high γ.

γ	ε	Convergence Time (s)	Average Reward	Success Rate (%)	Avg Episode Length (steps)	Final Q-value Variance
0.5	0.7	24.00	-129.73	0.10	30.93	484.04
0.5	0.9	32.07	-138.93	0.00	39.93	328.86
0.7	0.1	22.95	26.44	80.22	35.00	416.33
0.7	0.3	22.58	-42.30	43.98	31.26	532.28
0.7	0.5	21.28	-94.30	15.99	27.28	556.58
0.7	0.7	22.90	-127.68	0.37	29.42	548.47
0.7	0.9	30.92	-136.22	0.00	37.22	321.15
0.9	0.1	22.99	25.50	79.55	34.60	586.86
0.9	0.3	21.89	-41.98	44.06	31.10	726.47
0.9	0.5	20.33	-94.38	15.84	27.06	757.76
0.9	0.7	22.75	-125.65	0.99	28.63	724.78
0.9	0.9	30.19	-136.13	0.00	37.13	336.13
0.99	0.1	22.73	26.93	80.34	34.75	1328.53
0.99	0.3	22.35	-41.25	44.52	31.29	1503.13
0.99	0.5	20.93	-93.96	16.18	27.32	1595.17
0.99	0.7	21.77	-125.89	0.86	28.61	1494.32
0.99	0.9	31.62	-137.84	0.00	38.84	343.01

Table 4 – numeric results for effects analysis

Effect of Exploration Rate (ε)

The exploration rate controls the balance between exploration (trying new actions) and exploitation (choosing the best-known action).

- **Low ε (0.1)**: High success rates (e.g., **79.78%** for γ=0.5 and ε=0.1) and low episode lengths (e.g., **35.09 steps**). A low ε prioritizes exploitation, allowing the agent to follow the best-known policy. This works well when the agent has already learned a good policy but may fail if the initial policy is poor.
- **Moderate ε (0.3, 0.5)**: Mixed results. For γ=0.5, ε=0.3 yields a success rate of **44.18%**, while ε=0.5 yields **14.76%**. Moderate ε balances exploration and exploitation. While it helps the agent discover better policies, excessive exploration (e.g., ε=0.5) can reduce performance by diverting the agent from optimal paths.
- **High ε (0.7, 0.9)**: Poor performance, with success rates close to **0%** and high episode lengths. High ε prioritizes exploration, causing the agent to take random actions frequently. This prevents the agent from converging to an optimal policy, as it spends too much time exploring suboptimal paths.

Interaction Between γ and ε

- **Low γ and High ε**: The worst performance is observed, as the agent focuses on immediate rewards and explores excessively, failing to learn a meaningful policy.
- **Moderate/High γ and Low ε**: The best performance is achieved, as the agent balances long-term rewards with exploitation of the learned policy.
- **Moderate/High γ and High ε**: Performance degrades due to excessive exploration, even though the agent values long-term rewards.

Discussion

The results demonstrate that the **discount factor (γ)** and **exploration rate (ε)** significantly impact the performance of Q-learning in the Gridworld environment. There is a clear trade-off between exploration and exploitation; while some exploration is necessary to discover good policies, excessive exploration prevents the agent from converging to an optimal policy. For similar environments, it is recommended to use a moderate to high γ (e.g., 0.7 to 0.9) and a low ε (e.g., 0.1 to 0.3) to achieve the best performance. In conclusion, the results highlight the importance of carefully selecting γ and ε in Q-learning, with moderate to high γ and low ε generally yielding the best results in this Gridworld scenario. Future improvements could involve fine-tuning hyperparameters, implementing epsilon decay, exploring advanced strategies like Boltzmann exploration, and refining the reward structure.

This study provided key insights into optimizing Q-learning performance.

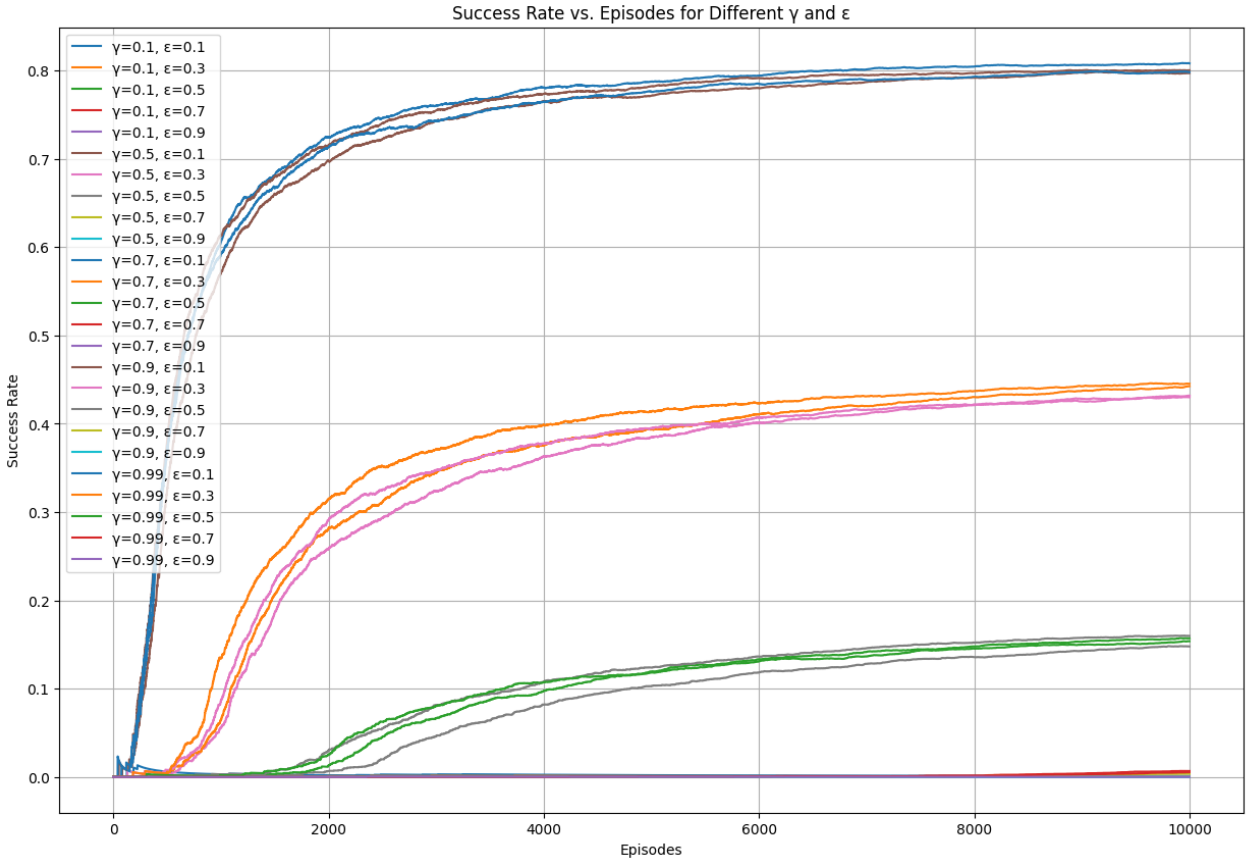

Figure 13 – Success rate vs Episode for different y and e graph

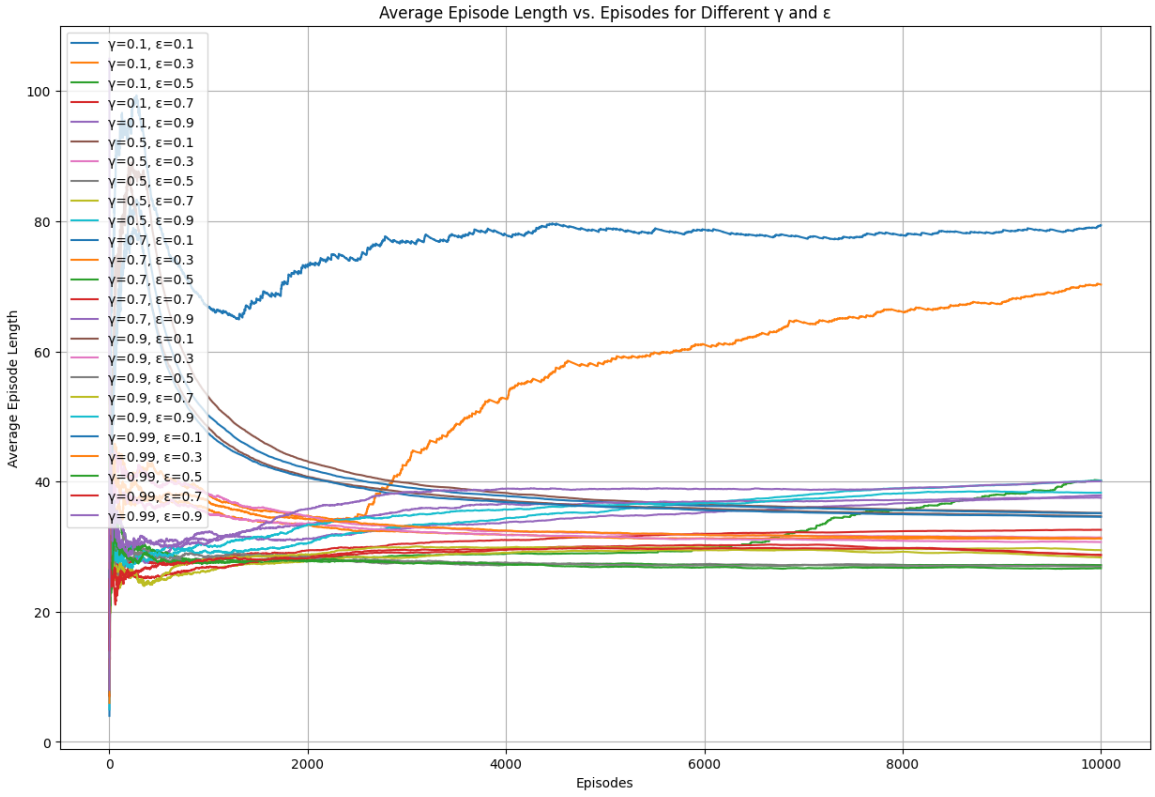

Figure 14 – Average episode length vs Episode for different y and e graph