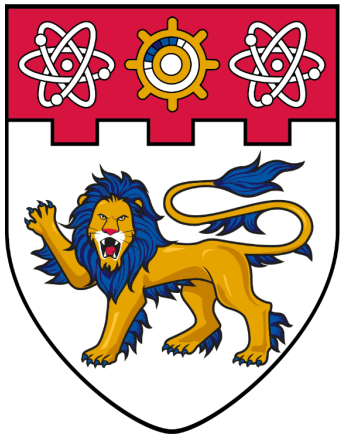

NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

SC4001 / CE4042 / CZ4042
Neural Network and Deep Learning

Project Report

AY 2023-2024

Prepared by:

Gucon Nailah Ginylle Pabilonia (U2021643H)

Nur Dilah Binte Zaini (U2022478K)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

NANYANG TECHNOLOGICAL UNIVERSITY

Table of Content

1. Introduction **3**

2. Literature Review **4**

3. Methodology **6**

3.1 Environment Set Up 6

3.2 RunLah Pygame Application 6

3.3 Development of RunLahAI Game Bot 7

4. Experiments and Results **10**

4.1 Qualitative Results 10

4.1 Quantitative Results 10

5. Discussion **11**

5.1 Conclusion 11

5.2 Future Works 11

6. Installation Guide **12**

References **13**

1. Introduction

The popularity of online gaming has increased significantly over the years. The widespread appeal of online gaming can be attributed to its unparalleled accessibility and convenience. In the dynamic landscape of online gaming, maintaining a competitive edge and keeping up with the trends is a necessity for game companies to thrive. They can elevate the performance in gameplay experience by making their games more dynamic, responsive and immersive. Artificial Intelligence (AI) algorithms play a pivotal role by offering valuable insights, which assists developers in pinpointing specific areas of the gameplay mechanics that can be refined and improved.

Chrome Dino Game is one of the most popular online games, achieving over 270 million games played every month [1]. To add our own personal touch to the well-loved game, we decided to build our very own endless runner PyGame game called “RunLah”, that uses similar game mechanics as Chrome Dino Game. The objective of our game is to assist the main character, Wizard Lah, in running as far as it can without hitting any obstacles. This project focuses on incorporating the popular deep-learning framework, EfficientNetV2 into our AI game bot to learn how to play the custom endless runner game. Due to time constraints and hardware resource limitations, traditional game neural networks with reinforcement learning will not be used in this project.

The rest of the report is structured as follows. In Section 2, previous works done in the field of artificial intelligence in games are discussed. Section 3 describes the methodology used to develop our application, dataset preparation & processing, and model development & training so that the AI game bot can be trained and learn how to play the game. Section 4 discusses the experiments conducted and its findings. Section 5 serves as the concluding remark of the report. Section 6 provides detailed instructions in setting up the python environment to execute the application and run the AI game bot.

2. Literature Review

Reinforcement learning is a machine learning algorithm that penalizes undesirable behavior and rewards favorable behavior which matches the concept of gaming. However, requirements for completely observed, low-dimensional state spaces and handcrafted characteristics have limited the use of reinforcement learning. Therefore, existing reinforcement learning applications are commonly combined with deep learning techniques to solve complex tasks such as playing games.

TD-gammon [2], a backgammon-playing program, used a model-free reinforcement learning algorithm and utilized a multi-layer perceptron with one hidden layer to approximate the value function to achieve an extraordinary degree of play. However, the TD-gammon approach is effective only in backgammon. The approach does not perform well in other games.

Using TD-gammon as a starting point, Mnih et al [3] connected a reinforcement learning algorithm to a deep neural network which operates directly on RGB images and efficiently processes training data by using stochastic gradient updates, achieving state-of-the-art results in six out of seven Atari 2600 games. However, it is computationally demanding due to the reinforcement learning algorithm used.

In alignment with the objectives of our project, Marwah et al. [4] compared 3 algorithms which are Deep Q Network (DQN), Double Deep Q-Network (DDQN) and Expected SARSA to train an agent to play Chrome Dino Run. For these algorithms, visual input is created by taking 4 consecutive screenshots of the game frames and preprocessing them to reduce their dimensionality. The 3 Convolutional layers resized these images before flattening them into a dense layer. This forward feeds into the output layer which gives 2 values of Q as output, 1 for each action state (jump and do nothing) which is DQN that acts as the baseline for the other algorithms. DDQN learns the best and finds better policies even though it is slower than DQN. Expected SARSA is the slowest to converge and performs the worst among the algorithms. Even though the DDQN shows promising results, the overall workflow for the algorithm is complicated and is computationally demanding as the model is required to constantly update the q value till it reaches the optimal q value.

The goal of this project is to explore simpler techniques, in contrast to existing approaches that rely on complex methodologies and demand high computational power for playing an endless runner game. Moreover, these current methods heavily depend on large training datasets for algorithm training. However, in practical scenarios, the availability of limited data poses a challenge for learning optimal q values.

Instead of utilizing reinforcement learning combined with convolutional neural networks (CNN), exclusively employing CNNs, such as EfficientNetV2-S, can be used in training an AI game bot to play a game. EfficientNetV2-S [5] is a convolutional neural network which outperforms earlier models in terms of parameter efficiency and training speed.

Stage	Operator	Stride	#Channels	#Layers
0	Conv3x3	2	24	1
1	Fused-MBConv1, k3x3	1	24	2
2	Fused-MBConv4, k3x3	2	48	4
3	Fused-MBConv4, k3x3	2	64	4
4	MBConv4, k3x3, SE0.25	2	128	6
5	MBConv6, k3x3, SE0.25	1	160	9
6	MBConv6, k3x3, SE0.25	2	256	15
7	Conv1x1 & Pooling & FC	-	1280	1

Figure 1: EfficientNetV2-S architecture

It uses Fused-MBConv and MBConv in the early stages. As compared to EfficientNet, EfficientNetV2-S uses a smaller expansion ratio in MBConv which offers less memory access overhead. It does not have the last stride-1 stage unlike EfficientNet due to the large parameter size and memory access overhead.

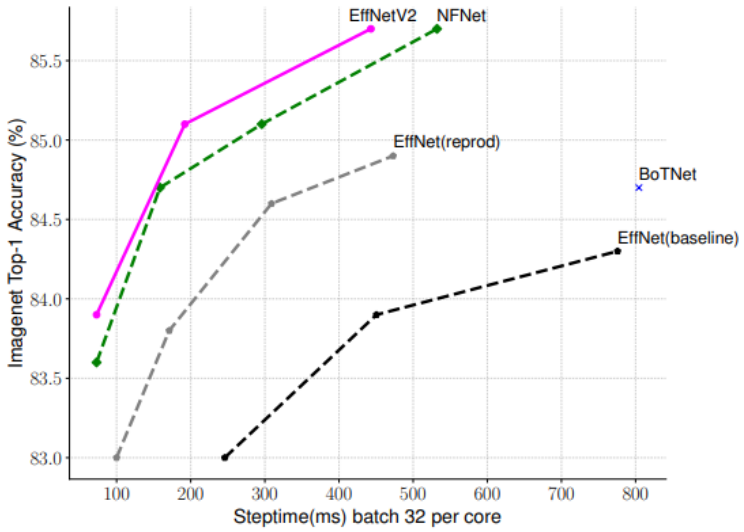

Figure 2: Accuracy scores and training step time on TPU v3 for models trained on ImageNet dataset

Apart from NFNets which was trained on 360 epochs, the other models in Figure 2 were trained with 350 epochs. All models utilized a similar number of training steps. In terms of training speed step time, the EfficientNetV2 model, with the training-aware NAS and scalling, achieves the shortest training time among the other models. The performance of EfficientNetV2 is better than state-of-the-art models while being up to 6.8x smaller. Therefore, we chose EfficientNetV2-S to be incorporated into our model that our game is trained on.

3. Methodology

3.1 Environment Set Up

Please refer to Section 6 for the Installation Guide, it provides detailed instructions on configuring the Python environment in PyCharm to streamline the execution process.

3.2 RunLah Pygame Application

High-Level Game Description

RunLah is an endless runner game developed from scratch with PyGame, where the player-controlled game avatar continuously moves forward through a side-scrolling landscape, while dodging obstacles of varying difficulty to achieve the best possible score.

Figure 3: Character Introduction

Additionally, the game features a starting screen that provides gameplay instructions and an end screen that showcases the player's final score.

Figure 4: Start and End Screens

3.3 Development of RunLahAI Game Bot

Data Collection and Processing

Data collection is carried out by simultaneously running the RunLah PyGame application and executing `capture_runlah_gameplay.py` which takes screenshots of the game screen while a human player engages in gameplay. The resulting screenshots are stored in a designated “captures” folder, with filenames containing the current exact date time and its associated keyboard event.

Filenames featuring the suffix "space" signify scenarios when the character should "jump," and those with the suffix "down" indicate when the character should "duck". Files marked with the suffix "n" denote scenarios when the character should “do nothing” or during the start and end screens of the game.

A total of 10275 game screenshots were gathered for training and testing of our RunLahAI game bot.

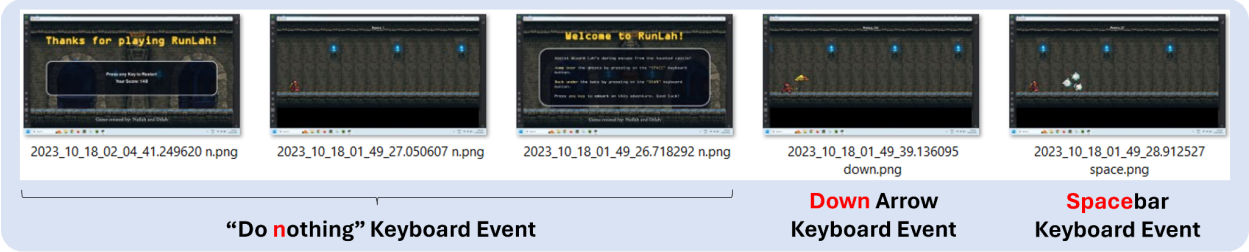

Figure 5: Types of files in captures folder

> See capture_runlah_gameplay.py code file to view the full code implementation with comments.

The images in the “captures” folder are then processed by extracting the filename and employing string manipulation to isolate the suffix. This suffix plays a crucial role in determining the image label corresponding to the keyboard event that controls the game avatar’s actions. The complete list of suffixes and their corresponding classes are listed in Table 1.

Table 1: suffixes and classes

Suffix	Label
n	0
space	1
down	2

Subsequently, the filenames and the associated labels for each of the captured images are written to a CSV file, as depicted in Figure 6.

labels.csv

```
file_name,class
2023_10_18_01_49_26.718292 n.png,0
2023_10_18_01_49_26.802651 n.png,1
2023_10_18_01_49_26.884420 n.png,0
2023_10_18_01_49_27.050607 n.png,2
```

Figure 6: sample of labels.csv

> See process_dataset.py code file to view the full code implementation with comments.

Model Development and Training

Prior to training the RunLahAI game bot model, a specialised RunLahDataset class is defined to facilitate image data processing, enabling seamless integration into the PyTorch data pipelines. This class is designed to work harmoniously with a custom image transformation pipeline for EfficientNetV2-S.

After partitioning the dataset into an 80:20 ratio for training and testing, the train and test data loaders are established using the RunLahDataset class. These loaders play a vital role in supplying data to the neural network during training and testing, along with the defined batch size.

In this project, a comprehensive model evaluation will be conducted by varying the number of epochs and batch sizes to identify the optimal hyperparameter combination that yields the highest performance of a game bot.

Table 2: epochs and batch size combinations

Combinations (epochs, batch size)		
10, 2	10, 4	10, 8
20, 2	20, 4	20, 8
30, 2	30, 4	30, 8

After the necessary prerequisites have been defined, an instance of the EfficientNetV2 model is initialised using the torchvision library, which takes 1280 input features and produces 3 output features. This model plays a pivotal role in enhancing the RunLahAI game bot's understanding of the game environment without relying on reinforcement learning. Its efficiency in extracting relevant features allows the model to discern intricate patterns and nuances within the game, aiding the bot in making informed decisions. Other necessary components such as the criterion and optimiser are also initialised.

The Cross Entropy Loss, our chosen criterion, excels in classification tasks like playing the RunLah game. By combining the softmax activation and negative log-likelihood, it efficiently transforms raw model outputs into probabilities. This loss function effectively minimizes the disparities between high probabilities for the correct class and lower probabilities for incorrect ones, training the model to make accurate predictions for each game move.

The Stochastic Gradient Descent (SGD) algorithm, our chosen optimization method, leverages model parameters and updates them through computed gradients from backpropagation. In the context of training our RunLahAI game bot, where adaptability and responsiveness are crucial, SGD's stochastic nature and efficiency make it an ideal choice, enabling the model to swiftly learn and adjust to the dynamic challenges of the game environment.

> See train_RunLahAI.ipynb code file to view the full code implementation with comments.

The training process was conducted over the course of 2 weeks on a laptop equipped with an NVIDIA GeForce RTX 3050 Laptop GPU and an AMD Ryzen 7 5800H with Radeon Graphics. Table 3 shows the time taken for each of the models to complete training.

Table 3: Time taken for training

Time taken (in hours) for each combination (epochs, batch size) to complete training					
10, 2	2.21	**10, 4**	3.07	**10, 8**	12.89
20, 2	4.82	**20, 4**	12.49	**20, 8**	25.74
30, 2	7.16	**30, 4**	8.31	**30, 8**	42.68

To ensure effective training and prevent overfitting, we closely monitored the training loss, aiming for a consistent decrease throughout the training process.

Following training, the resultant trained models were evaluated on the test dataset, and their test accuracy scores were documented. To evaluate the performance of each model qualitatively, a code solution is devised to load each model to do predictions on real-time screenshots of the RunLah game and simulate the keyboard inputs accordingly.

> See RunLahAIBot.py code file to view the full code implementation with comments.

Execution Guidelines

All commands to be run in the project folder's main directory in the following order.

1. To run RunLah, run the following command in a terminal:

```
python .\RunLahGame\RunLahGame.py
```

2. To run the dataset preparation code solution to obtain the game screenshots, make sure RunLah is running and run the following command in a terminal:

```
python .\capture_runlah_gameplay.py
```

3. To run the dataset processing code solution to obtain the CSV file with the image filenames and their associated labels, run the following command in a terminal:

```
python .\process_dataset.py
```

4. To run the training code solution, run the following command in a terminal:

```
python -m notebook
```

5. Once Jupyter is open, navigate to the `train_RunLahAI.ipynb`, and "Run All Cells".

6. (To just test the provided model, skip steps 2 - 5) To run the RunLahAI bot to play RunLah, make sure RunLah is running and run the following command in a terminal:

```
python .\runlahAIbot.py
```

4. Experiments and Results

Following each training sequence, the model undergoes evaluation on the test dataset to derive the test accuracy across various epochs and batch sizes. This assessment involves applying the PyTorch softmax function to convert model outputs into probabilities, utilizing argmax to identify the predicted class for each image. The total and correct number of predictions are recorded and the test accuracy is obtained with this formula: `100 * correct_predictions // total_predictions`.

> See “Test trained model on test dataset” code cell in train_model.ipynb to view the full code solution.

4.1 Qualitative Results

Click here to watch the YouTube video created that showcases the RunLah gameplay by the AI game bot across 10 games for each model with different epochs and batch sizes.

4.1 Quantitative Results

Tables 4 and 5 illustrate the test accuracies of each model variation and the average scores achieved by the game bot respectively.

Table 4: Test Accuracies for models with different epochs and batch sizes.

	Batch Size of 2	Batch Size of 4	Batch Size of 8
10 epochs	79%	87%	88%
20 epochs	83%	90%	91%
30 epochs	86%	90%	91%

Table 5: Average high scores across 10 games for models with different epochs and batch sizes

	Batch Size of 2	Batch Size of 4	Batch Size of 8
10 epochs	113	100	297
20 epochs	188	351	1313
30 epochs	140	289	1183

The model trained with **batch size of 8** and **20 epochs** outperformed all the other models. It not only attained one of the highest test accuracies but also consistently outperformed others with the top average score across 10 games, highlighting its robustness and effectiveness in playing the game. In terms of test accuracy, although the model with batch size of 8 and 30 epochs attained a similar score, the significant time investment required for its training renders it less favorable than the model trained with a batch size of 8 and 20 epochs. Hence, opting for the model with lesser number of epochs and shorter training duration appears to be the wiser choice, striking an optimal balance between accuracy and efficiency.

5. Discussion

5.1 Conclusion

Based on our experiments with different batch sizes, the performance of EfficientNetV2-S is optimal when trained with a batch size of 8. As compared to batch size of 4, utilizing a batch size of 8 allows the model to have a better generalization, attributed to the larger batch size contributing to enhanced accuracy scores. Training with batch size of 2 can lead to poor generalization to unseen data due to the noisy gradients.This can cause the model to converge to suboptimal or even non-optimal solutions. Hence, even though it reaches convergence faster than other batch sizes, the accuracy scores are the lowest.

With the batch size of 8, the optimal number of epochs is 20. Despite the potential computational expenses associated with prolonged training periods, the computational requirements for a batch size of 8 with 20 epochs are considerably lower compared to a batch size of 8 with 30 epochs, which also imposes higher memory demands. Even though batch size of 8 and 20 epochs can be computationally intensive and may have slow convergence as compared to other batch sizes with fewer epochs, the model has a higher chance to reach the global minimum, leading to a more optimal solution.

5.2 Future Works

The proposed method in this project can be improved in the areas as stated below.

Introducing Reinforcement Learning

As mentioned in Literature Review, reinforcement learning is required to be combined with deep learning techniques to handle high dimensional inputs such as game frames. By combining reinforcement learning with our approach, the model can learn more complex and optimal strategies to achieve superhuman game play.

Upgrading Hardware

With an increase in GPU performance in the future, training CNN with reinforcement learning techniques can be supported to improve the performance of the model. Larger neural networks can also be implemented to execute reinforcement learning tasks in complex environments.

Large datasets

Utilizing larger training datasets can reduce the possibility of overfitting as the model is more inclined to capture the patterns in the dataset instead of memorizing the training dataset.

The improved model can be used to learn complex games that produce inputs in high dimensional space for future work.

6. Installation Guide

Please follow the guide below carefully to ensure smooth execution of all of the code solutions.

1. Set up a virtual environment in **PyCharm** (Please set the theme to "Dark").
2. Unzip the zip folder called `CZ4042_Project_Nailah_Dilah.zip` and copy the contents to your working directory.
3. In addition to the libraries in the requirements.txt, please install the following prerequisites:

    - Python 3.11
    - CUDA Toolkit 11.8
    - PyTorch 2.0.1 and other related libraries (run code below):

```
pip install torch==2.0.1 torchvision==0.15.2 torchaudio==2.0.2 --index-url https://download.pytorch.org/whl/cu118
```

    It is important to ensure these prerequisites are installed to ensure smooth execution of all of the code solutions.

    Next, run the following command to install the rest of the necessary libraries:

```
pip install -r requirements.txt
```

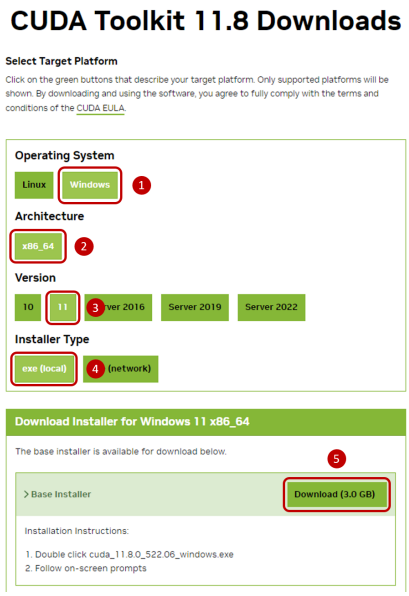

Figure 7: CUDA Toolkit Download Screen

4. Download the model here.

    - Your working directory should look like Figure 8.
    - To **just test the provided model**, run steps 1 and 6 as mentioned in the Execution Guidelines under the Methodology section.
    - To **run through the whole process** (data preparation to model training to model testing), follow all the steps mentioned in the Execution Guidelines under the Methodology section.

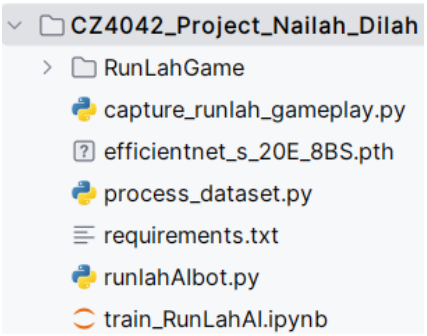

Figure 8: Expected Working Directory

References

[1] M. Hughes, ‘4 years later, Google finally explains the origins of its Chrome dinosaur game’, TNW | Dd. Accessed: Nov. 08, 2023. [Online]. Available: https://thenextweb.com/news/4-years-later-google-finally-explains-the-origins-of-its-chrome-dinosaur-game

[2] G. Tesauro, ‘Temporal difference learning and TD-Gammon’, Commun. ACM, vol. 38, no. 3, pp. 58–68, Mar. 1995, doi: 10.1145/203330.203343.

[3] V. Mnih et al., ‘Playing Atari with Deep Reinforcement Learning’, 2013, doi: 10.48550/ARXIV.1312.5602.

[4] D. Marwah, S. Srivastava, A. Gupta, and S. Verma, ‘Chrome Dino Run using Reinforcement Learning’, 2020, doi: 10.48550/ARXIV.2008.06799.

[5] M. Tan and Q. V. Le, ‘EfficientNetV2: Smaller Models and Faster Training’. arXiv, Jun. 23, 2021. Accessed: Nov. 08, 2023. [Online]. Available: http://arxiv.org/abs/2104.00298