SubwAI: AI Agent for Subway Surfers using CNN

Subject: CS502 – Advanced Pattern Recognition

1.Abstract

This project, titled *SubwAI*, focuses on developing an artificial intelligence (AI) agent capable of autonomously playing the popular endless-runner game **Subway Surfers**.
The agent is trained using **Convolutional Neural Networks (CNNs)** and **imitation learning** — a form of supervised learning where the model learns from human gameplay rather than trial and error.

A dataset of gameplay screenshots paired with corresponding user actions (jump, roll, left, right, no-op) is collected.
The CNN learns to map visual frames to the best possible in-game action.
Once trained, the model can play the game autonomously in real time by observing the screen and pressing the correct keys.

The goal of this project is to explore the potential of computer vision–based imitation learning in developing real-time game-playing AI systems.

---

2. Introduction

Video game AI has traditionally relied on rule-based systems or reinforcement learning.
However, imitation learning allows AI to learn human-like decision-making by observing and imitating real players.
SubwAI applies this principle to a visually dynamic game, *Subway Surfers*, which involves continuous decision-making in rapidly changing environments.

The project leverages **supervised deep learning** through CNNs to predict the player's next move given a screenshot.
Unlike reinforcement learning, where the agent learns from reward signals, imitation learning uses human demonstrations as labeled examples.

This method reduces training time, avoids complex reward engineering, and enables faster generalization for vision-based tasks.

---

3. Objectives

1. Develop an AI agent capable of playing *Subway Surfers* automatically.
2. Build and train a Convolutional Neural Network to predict actions from screenshots.
3. Collect and preprocess a labeled dataset of gameplay frames.
4. Evaluate model accuracy and real-time decision performance.
5. Explore the use of data augmentation for better generalization.

6. Demonstrate the feasibility of imitation learning in dynamic gaming environments.

---

4. Methodology

The methodology for SubwAI consists of the following stages:

4.1 Data Collection

- The player plays *Subway Surfers* manually.
- Screenshots of the game are captured at each frame along with the corresponding action (jump, roll, left, right, or no action).
- Images are saved in structured folders by action label.

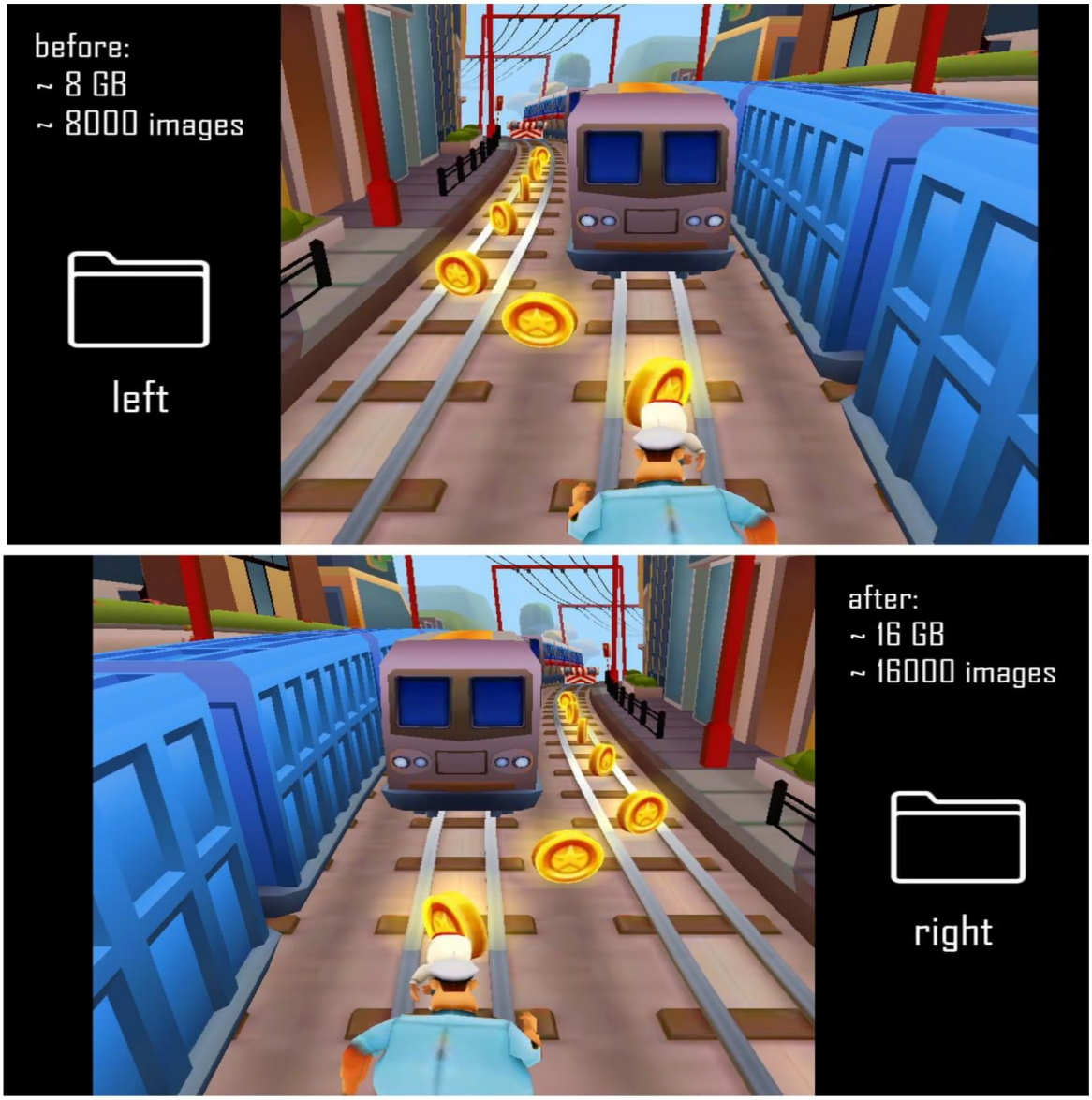
4.2 Data Preprocessing

- Images are resized and normalized.
- Noise is reduced, and images are converted into arrays suitable for CNN input.
- The dataset is split into training, validation, and testing sets.

4.3 Data Augmentation

To increase dataset diversity and avoid overfitting:

- Horizontal flips
- Random cropping
- Rotation and brightness adjustments
  These augmentations help simulate unseen scenarios and improve robustness.

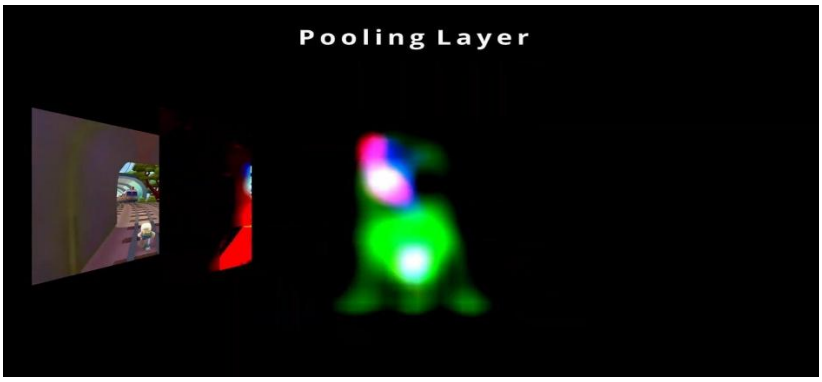


4.4 Model Architecture

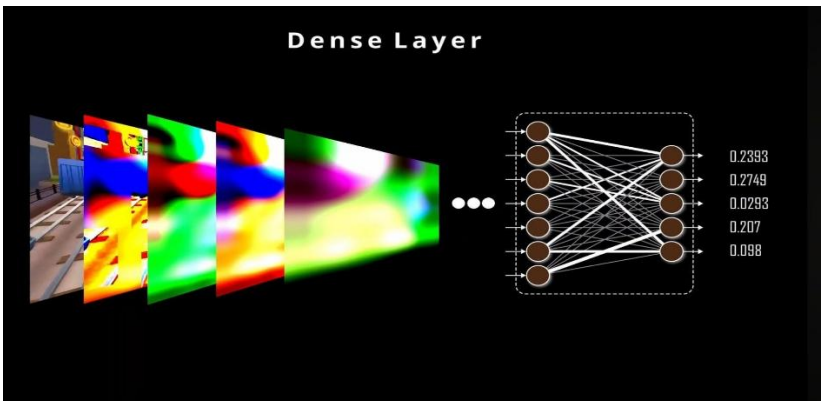
A CNN is trained to classify each frame into one of the possible actions.

- **Input Layer:** Preprocessed image frames (resized).
- **Convolutional Layers:** Extract visual patterns like obstacles, lanes, or coins.

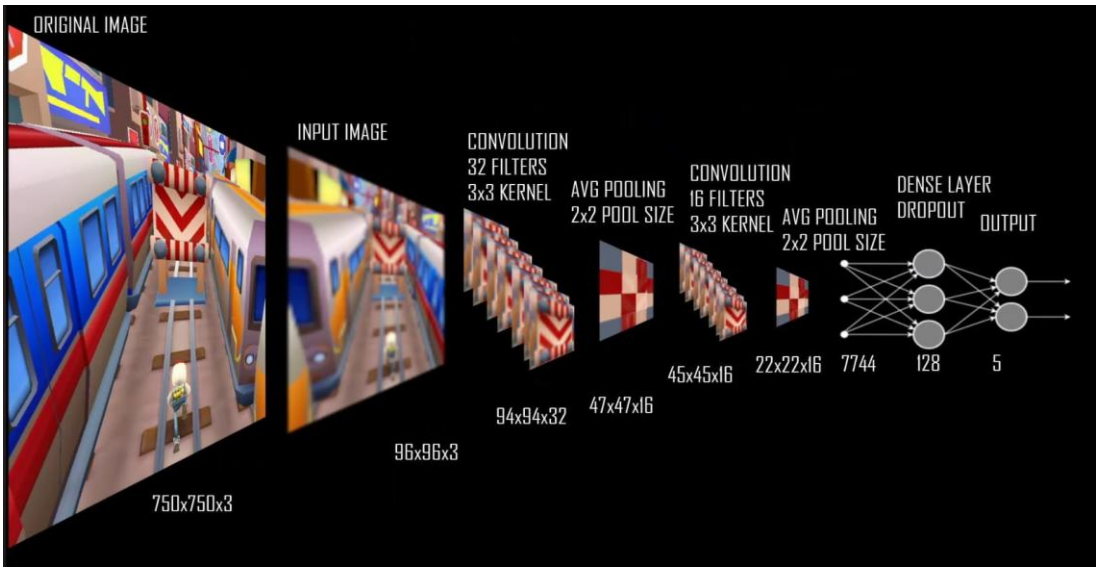
- **Pooling Layers:** Reduce spatial dimensions and retain key features.



- **Fully Connected Layers:** Combine extracted features to make the final action decision.



- **Output Layer:** Softmax activation for 5 classes (up, down, left, right, noop).



4.5 Training and Evaluation

- The model is trained using **categorical cross-entropy loss** and the **Adam optimizer**.

- Accuracy and loss graphs are recorded to monitor learning.
- Around 85% test accuracy was achieved after augmentation.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 94, 94, 32)        896
_________________________________________________________________
average_pooling2d (AveragePooling2D)  (None, 47, 47, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 45, 45, 16)        4624
_________________________________________________________________
average_pooling2d_1 (AveragePooling2D)  (None, 22, 22, 16)        0
_________________________________________________________________
flatten (Flatten)            (None, 7744)              0
_________________________________________________________________
dense (Dense)                (None, 128)               991360
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 645
=================================================================
Total params: 997,525
Trainable params: 997,525
Non-trainable params: 0
_________________________________________________________________
Epochs: 20
Accuracy: 0.8525272011756897% | Loss: 0.5509672164916992
```

4.6 Deployment

- The trained model is deployed in real time to capture the game screen, process frames, run inference, and simulate game controls automatically.
- **Tools Used:**
  - **Screen Capture:** mss, OpenCV
  - Preprocessing: NumPy, OpenCV
  - **Model Inference:** PyTorch (optionally TorchScript/ONNX)
  - **Acceleration:** CUDA/cuDNN (GPU), TensorRT/OpenVINO (optional)
  - **Action Simulation:** pyautogui, or low-latency SendInput (Windows API)
- Captured frames are resized, normalized, and converted to the format expected by the CNN before inference.
- The model predicts an action every frame (or at target FPS), using model.eval() and torch.no_grad() for efficient inference.
- Predicted actions are executed using automated key presses or mouse events, enabling the agent to control the game.

---

5. Implementation

5.1 Tools and Libraries

- **Python**
- **TensorFlow / Keras** for CNN model building
- **OpenCV** for image capture and processing
- **NumPy / Pandas** for data handling
- **PyAutoGUI** for simulating key presses

5.2 Code Overview

- **ai.py** — Main script for data collection, training, and playing.
- **dataset_augmentation.py** — Performs image transformations.
- **image_sort.py / image_check.py** — Manages and verifies image labels.
- **models/ folder** — Stores trained model files and graphs.
- **recordings/** — Stores gameplay videos of AI performance.

The system operates in three modes:

1. **Gather Mode:** Collect new labeled screenshots.
2. **Train Mode:** Train the CNN using the dataset.
3. **Play Mode:** Run the trained model to play automatically.

---

6. Results and Analysis

The trained CNN achieved around **85% test accuracy** on unseen data.
When deployed, the AI successfully played *Subway Surfers* autonomously for over a minute before failure.

Observations

- The model handled lane changes and jumps well.
- Occasionally failed on rare or unseen obstacle patterns.
- Data augmentation improved accuracy by ~10%.
- Performance was limited by screen capture and inference speed.

The results confirmed that imitation learning with CNNs can effectively replicate human gameplay patterns in visually complex environments.

---

7. Challenges Faced

1. **Data Imbalance:** Some actions (like ‘jump’) occurred more frequently, causing class imbalance.
2. **Real-time Delay:** The model needed to make predictions in milliseconds to keep up with gameplay.
3. **Limited Dataset:** Manual data collection constrained model exposure to rare scenarios.
4. **Overfitting:** Without augmentation, the model overfit to specific scenes.
5. **Dynamic Game Speed:** Increasing speed during gameplay made predictions harder.

Solutions included data balancing, regularization, and optimization for faster inference.

---

8. Conclusion

The SubwAI project demonstrates the effectiveness of CNN-based imitation learning for real-time gaming tasks.
By learning directly from human gameplay data, the AI agent can mimic player actions and handle dynamic visual input effectively.

The project showcases:

- The practicality of supervised deep learning for visual control.
- The value of augmentation in small datasets.
- The potential of imitation learning as an alternative to reinforcement learning in fast-paced games.

---

9. Future Scope

- Integrate **Reinforcement Learning (RL)** to allow the agent to learn from its own mistakes and improve over time.
- Use **Recurrent Neural Networks (RNN/LSTM)** to incorporate temporal awareness (past frames influence current action).
- Expand dataset diversity with multiple players and different game environments.
- Improve **real-time inference** using GPU acceleration.
- Apply the same concept to other games or robotic control tasks.

Group Contribution:

1. 2201AI19 – Koppolu Buddha Bhavan – 8.33%
2. 2201CS38 – Komarabatini Vishwa Chaitanya – 8.33%
3. 2201CS12 – Alam Sai Bharath – 8.33%
4. 2201CS71 – Sripathi Surya Teja – 8.33%
5. 2201CS29 – Nikhil Gurram – 8.33%
6. 2201AI44 – Sadupati Vinay – 8.33%
7. 2201AI08 – Boreddy Akshitha Reddy – 8.33%
8. 2201AI43 – Haswanthi – 8.33%
9. 2201CS69 – Sriyash – 8.33%
10. 2201AI49 – Pranav Sai – 8.33%
11. 2201AI37 – Gorantla Sowmya Raj – 8.33%
12. 2201AI36 – Somil Aggarwal – 8.33%