Muhammad Nade Ali (23i-0116) Abdul Mateen(23i-2511)

Hanzlah Hassan(23i-0085) Awais Ali(23i-0080)

Bachelors of Artificial Intelligence

Submission Date: 09th May, 2025

Project Report

AI Powered Game Bot

Objective:

The purpose of this project is to design a bot that has the ability to play Street Fighter II Turbo in a real-time manner through the functional use of the machine learning aspect. The bot uses human gameplay data to train to predict best button presses for each game frame, and ends up replacing rule based logic with data driven approach.

Part 1: Data Collection:

Objective: Collect gameplay data that records human decision making under many different scenarios for training the machine learning model.

Implementation:

- Modified bot.py to capture game data in real time.
- Game states and actions were logged into a CSV file called **GameState.csv**.
- Manual gameplay in single-player mode was performed to create diverse data, including attacking, defensive, and movement strategies.

Captured Data Included:

- Player coordinates: x_coord, y_coord
- Health values

- Round timer
- Fight results
- Button states: A, B, X, Y, L, R, Up, Down, etc.

Part 2: Data Preprocessing

Cleaning:

- Removed rows with invalid or missing fight results (e.g., "P1", "P2" issues).
- Dropped irrelevant or redundant columns like Player1_move_id and fight_result.
- Converted all **boolean values** (True/False) to **binary (1/0)**.

Normalization:

- Scaled numerical values (coordinates, health, timer) to a **[0, 1]** range.
- Normalized player IDs by dividing them by **100** for categorical handling.

Target Variables:

- Defined output columns as all Player 1 button states:
  Player1_player_buttons_*
  (one column per button).

Part 3: Model Selection & Training

Model Architecture:

A **Multilayer Perceptron (MLP)** was selected for its efficiency and performance in multi-label classification.

- **Layers:**
  4 hidden layers with units: 128 → 64 → 32 → 16
  Activation: ReLU
- **Output Layer:**
  12 units (each corresponding to a game button)
  Activation: Sigmoid

Training Configuration:

- **Loss Function:** Binary Cross-Entropy (multi-label suitable)
- **Optimizer:** Adam
- **Validation Split:** 80% training / 20% testing
- **Regularization:** Early Stopping to prevent overfitting

Results:

- Achieved **~90% accuracy** on the test set (varies by data size).
- Plotted training vs. validation loss and accuracy curves for evaluation.
- Saved trained model as game_model.h5.

Part 4: Integration Into Gameplay

Changes to bot.py:

Replaced traditional rule-based logic with real-time model prediction.

Workflow:

1. **Input Collection:**
   Current game state is captured through the GameState object.
2. **Preprocessing:**
   Input features are normalized using the saved scaler (data_scaler.pkl).

3. **Prediction:**
   model.predict() returns probabilities for each button.
4. **Thresholding:**
   If probability > 0.5 → press button, else don't.
5. **Command Execution:**
   Predicted button states are applied via the Command object.

Manual Control Backup:

For debugging, some manual keyboard controls (like Start/Select) were retained.

Part 5: Testing & Deployment

Testing Steps:

- Emulator: **BizHawk**
- Game: **Street Fighter II Turbo**
- Commands:
  - python controller.py 1 → run bot as Player 1
  - python controller.py 2 → run bot as Player 2
- Connected the bot using the **Gyroscope Bot** overlay.

Observed Performance:

- Bot **reacted intelligently** to opponent movement.
- Pressed appropriate buttons depending on health, proximity, and context.
- Worked across various characters due to **scaled categorical inputs**.

SCREENSHOTS:

Screenshots of important libraries:

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow.keras.optimizers import Adam

from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
```

Code screenshots:

```python
path = "GameState.csv"
df = pd.read_csv(path)

print("Original data shape:", df.shape)
print(df.info())
# print("Null values before cleaning:", df.isnull().sum().sum())

# # data cleaning
```

```python
# # data cleaning
df = df[~df['fight_result'].isin(['P2 ', 'P1'])]
print(df.head(4))
df.drop(columns=['fight_result', 'Player1_move_id', 'Player2_move_id'], inplace=True)
for col in df.columns:
    if df[col].dtype == 'object':
        try:
            df[col] = pd.to_numeric(df[col])
        except ValueError:
            print(f"Skipped column: {col}")


df = df.replace(True, 1)
df = df.replace(False, 0)
print(df.info())
```

```python
df = df.replace(True, 1)
df = df.replace(False, 0)
print(df.info())


# handle missing values before proceeding
print("null values after initial cleaning:", df.isnull().sum().sum())
df.dropna(inplace=True)
print("final data shape after cleaning:", df.shape)

# Feature scaling
columns = df.columns
for col in columns:
    if col in ['Player1_player_id', 'Player2_player_id']:
        df[col] = df[col] / 100
        continue

    maxVal = df[col].max()
    if maxVal > 1:
        df[col] = df[col] / maxVal

# output columns
```

```python
# Build the MLP model
model = Sequential()

# Input layer + Hidden layers
model.add(Dense(units=128, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=16, activation='relu'))

model.add(Dense(units=12, activation='sigmoid'))

# Compile with proper loss function
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

# model.compile(optimizer='adam', loss='binary_crossentropy',metrics=['accuracy'])

# add early stopping to prevent overfitting
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)

history = model.fit(X_train, y_train, epochs=30, batch_size=64,validation_split=0.2,callbacks=[early_stopping])
```

```python
# Train the model with smaller batch size
history = model.fit(X_train, y_train, epochs=30, batch_size=64,validation_split=0.2,callbacks=[early_stopping])

test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")

plt.figure(figsize=(12, 5))

# Plot accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
```

Below are some screenshots attached for your reference which were taken during model training:

```
1246/1246 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.5401 - loss: 0.1069 - val_accuracy: 0.5
Epoch 30/30
1246/1246 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.5341 - loss: 0.1067 - val_accuracy: 0.5
779/779 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5286 - loss: 0.1122
Test Loss: 0.1119
Test Accuracy: 0.5331
```

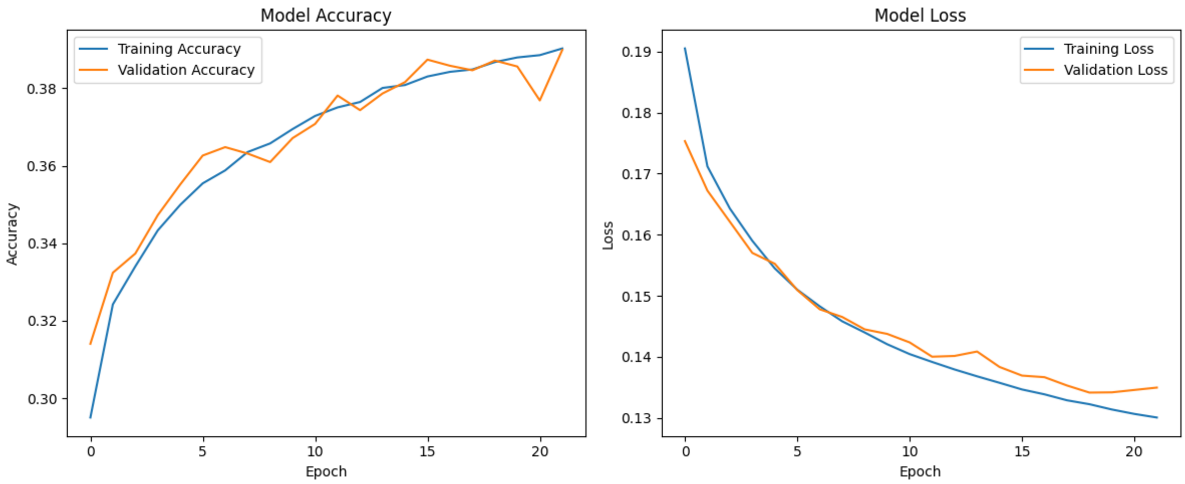

```
1760/1760 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.3799 - loss: 0.1353
Test Loss: 0.1348
Test Accuracy: 0.3816
1760/1760 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model
end using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(

--- Prediction 1 ---
```

```
rmance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow wit
Epoch 1/30
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 70s 23ms/step - accuracy: 0.2624 - loss: 0.2268 - val_accuracy: 0.3140 - val_loss:
Epoch 2/30
```

```
Epoch 19/30
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 41s 10ms/step - accuracy: 0.3850 - loss: 0.1338 - val_accuracy: 0.3830 - val_loss: 0.1380
Epoch 20/30
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 38s 9ms/step - accuracy: 0.3861 - loss: 0.1328 - val_accuracy: 0.3882 - val_loss: 0.1384
Epoch 21/30
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 26s 9ms/step - accuracy: 0.3880 - loss: 0.1319 - val_accuracy: 0.3866 - val_loss: 0.1370
Epoch 22/30
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 37s 8ms/step - accuracy: 0.3870 - loss: 0.1314 - val_accuracy: 0.3852 - val_loss: 0.1372
Epoch 23/30
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 40s 7ms/step - accuracy: 0.3873 - loss: 0.1315 - val_accuracy: 0.3928 - val_loss: 0.1369
Epoch 24/30
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 20s 7ms/step - accuracy: 0.3892 - loss: 0.1300 - val_accuracy: 0.3851 - val_loss: 0.1361
Epoch 25/30
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 19s 7ms/step - accuracy: 0.3922 - loss: 0.1298 - val_accuracy: 0.3930 - val_loss: 0.1361
Epoch 26/30
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 23s 8ms/step - accuracy: 0.3906 - loss: 0.1296 - val_accuracy: 0.3924 - val_loss: 0.1366
Epoch 27/30
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 40s 7ms/step - accuracy: 0.3907 - loss: 0.1285 - val_accuracy: 0.3883 - val_loss: 0.1348
Epoch 28/30
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 21s 8ms/step - accuracy: 0.3928 - loss: 0.1285 - val_accuracy: 0.3851 - val_loss: 0.1360
Epoch 29/30
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 28s 10ms/step - accuracy: 0.3911 - loss: 0.1275 - val_accuracy: 0.3824 - val_loss: 0.1352
Epoch 30/30
2816/2816 ━━━━━━━━━━━━━━━━━━━━ 72s 21ms/step - accuracy: 0.3951 - loss: 0.1268 - val_accuracy: 0.3927 - val_loss: 0.1341
1760/1760 ━━━━━━━━━━━━━━━━━━━━ 20s 11ms/step - accuracy: 0.3920 - loss: 0.1315
Test Loss: 0.1313
Test Accuracy: 0.3931
1760/1760 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step
```

A screenshot of graphs representing curves between model accuracy and model loss:



GAME PICTURES:


KO
RYU
91
ZANGIEF


1P
50000
START
KO
RYU
93
ZANGIEF