Artificial Neural Networks
Final Project: Analysis of Coin Landing Outcomes Based on Initial Conditions

Ege Demir

January 12, 2024

Summary

This report details an experimental study examining the outcomes of a coin dropped from a height of 1 meter, with a focus on both the final state (heads or tails) and the landing position on an A3 paper. Over 123 trials, outcomes were recorded under various initial conditions. A neural network with dual outputs was employed for analysis: one for classification (predicting heads or tails) and another for regression (estimating landing position). The network showed a preference for predicting 'heads', which is rational when considering the data: 'heads' occurred slightly more frequently in each initial condition (24 heads from tails, 21 heads from heads, and 23 heads from vertical drops). This trend and the tendency to favor frequently occurring landing spots underscore the challenges in capturing randomness with limited features and highlight the need for more complex data in predictive modeling.

1 Introduction

This study delves into the randomness of coin toss outcomes, examining both the final state (heads or tails) and the landing position on an A3 paper. The experiment involved 123 trials, where a coin was dropped from 1 meter, varying the initial condition and recording the outcomes. A neural network with classification and regression outputs was developed to analyze the data. The network predominantly predicted 'heads', a preference explained by the data distribution: 'heads' slightly outnumbered 'tails' in each initial condition, leading to a natural bias in the model's learning. This pattern highlights the limitations of using basic features like initial conditions and demonstrates the importance of incorporating more detailed factors to capture the true randomness of physical phenomena.

An Overview of the Observations

- The analysis revealed a consistent preference for 'heads' in the neural network's predictions, in line with the data showing 24 heads from tails, 21 heads from heads, and 23 heads from vertical drops.
- The neural network's structure, designed for both classification and regression, reflected its capability to learn from the slight biases in the training data, such as the more frequent occurrence of 'heads'.

- Factors like the precise technique of dropping the coin and environmental conditions, while potentially influential, were not included in the neural network's training, pointing to the need for more comprehensive data in future studies.

In conclusion, this study illustrates the complexities of modeling the randomness of a coin toss using a neural network. The network's tendency to reflect biases in the training data underscores the need for richer datasets and more sophisticated models to truly capture the random nature of such events .

2 Experimental Setup

The experimental setup was meticulously designed to maintain a consistent drop height of exactly 1 meter. Initial methods employing a push mechanism off a platform resulted in unwanted angular momentum, as evidenced by the trials shown in Figures 1a and 1b. The refined method involved free-dropping the coin by hand, using the platform as a reference to ensure the coin's drop was perpendicular, as depicted in Figures 2a through 2c. A total of 123 trials were conducted, encompassing 40 heads, 40 tails, and 43 vertical initial drops. The setup is shown in Figures 3a, 3b and 3c. The setup's accuracy was verified with images confirming the height measurement (Figures 4 and 5.

(a) Before the push.

(b) During the push.

Figure 1: The push mechanism that was not used.

3 Data Collection and Documentation

Precision in data collection was paramount, with outcomes marked on an A3 paper at the base after each coin drop. The initial state was indicated by color-coding—red for heads, blue for tails, and black for vertical drops—and outcomes were notated with 'H' for heads and 'T' for tails next to the coin's landing spot. This meticulous approach facilitated a clear visual record of the experiment, which was digitized for model training and analysis. Photographic documentation was integral to the data collection and documentation process. After each set of trials, high-resolution images were taken to capture the precision of the outcomes marked on the A3 paper. The sequence of marking outcomes can be followed through Figures 6a, 6b, 6c, and 6d, depicting the progression of 'H' and 'T' markings. The culmination of the data collection is shown in Figures 6e, 6f, and

(a) Just before dropping the coin.

(b) After the coin is dropped.

(c) Coin dropped and landed on paper.

Figure 2: The refined method of dropping the coin by hand using the platform as a reference.

6h, illustrating the final compilation of results. The final images were later processed in Adobe Lightroom to enhance clarity, particularly for analyzing the pixel coordinates of the markings, with the post-processing results presented in Figure 7.

The Excel sheet compiled from this data included the initial condition, the final outcome, and the precise landing position of the coin, enabling the training of a neural network to predict landing outcomes based on the initial drop parameters. The network's predictive insights were gleaned especially from areas with a higher frequency of drops, providing valuable information about the coin's landing patterns, as highlighted in Figure 8.

(a) Wide view of the experimental setup.

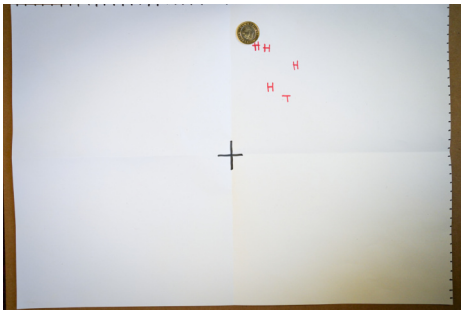
(b) Top view of the coin drop mechanism.

(c) Another top view of the coin drop mechanism.

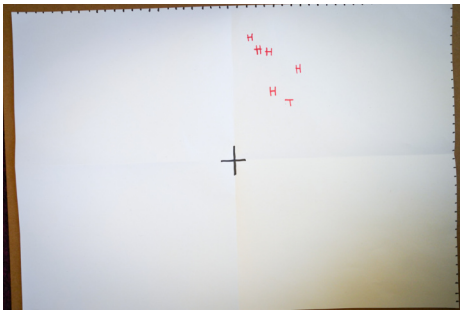
Figure 3: Experimental setup.

Figure 4: Setup height being measured as one meter, close up.

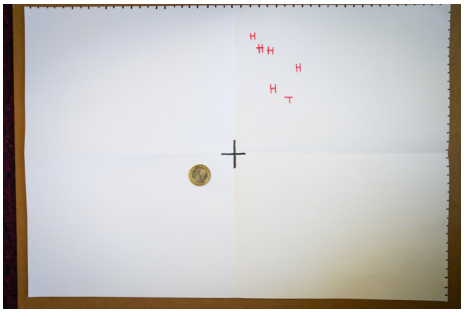
Figure 5: Setup height being measured as one meter, wider view.



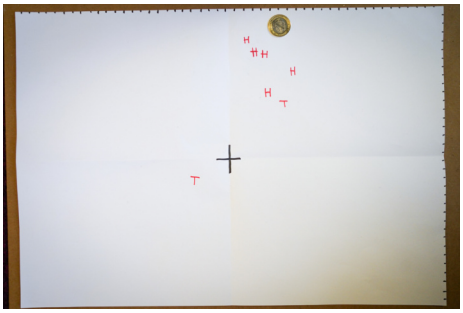
(a) Coin on the paper - heads.



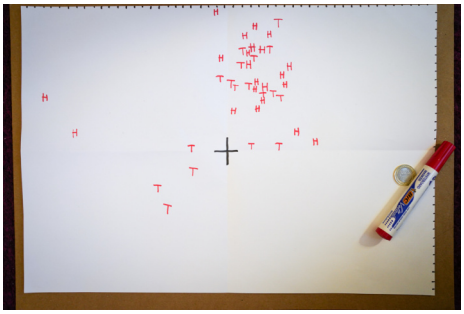
(b) Marking 'H' with red marker.



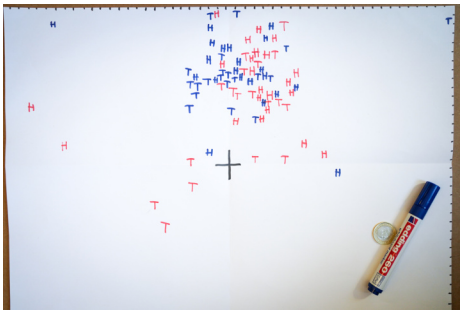
(c) Coin on the paper - tails.



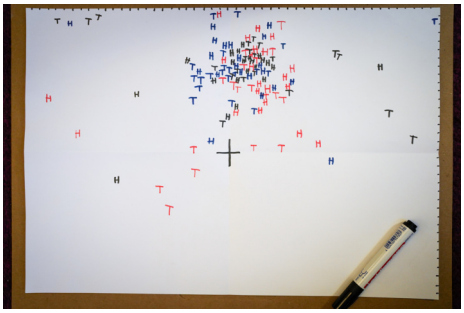
(d) Marking 'T' with red marker.



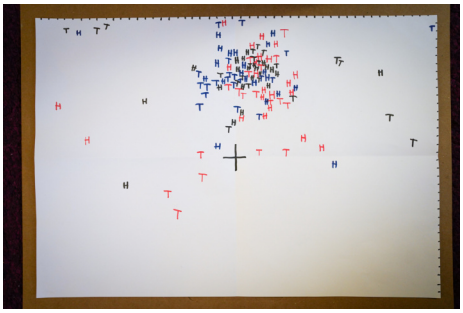
(e) Tails initial state in red.



(f) Adding the heads initial state trials in blue.



(g) Adding the vertical initial state trials in black.



(h) Final paper with all markings.

Figure 6: Data collection procedure.

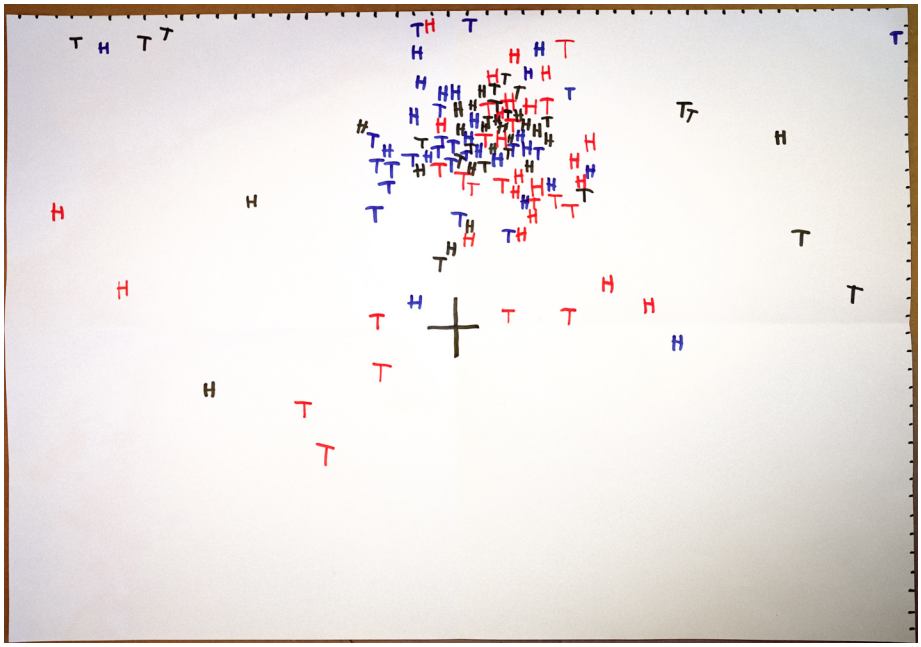


Figure 7: Enhanced image post Lightroom processing for clear analysis of markings.

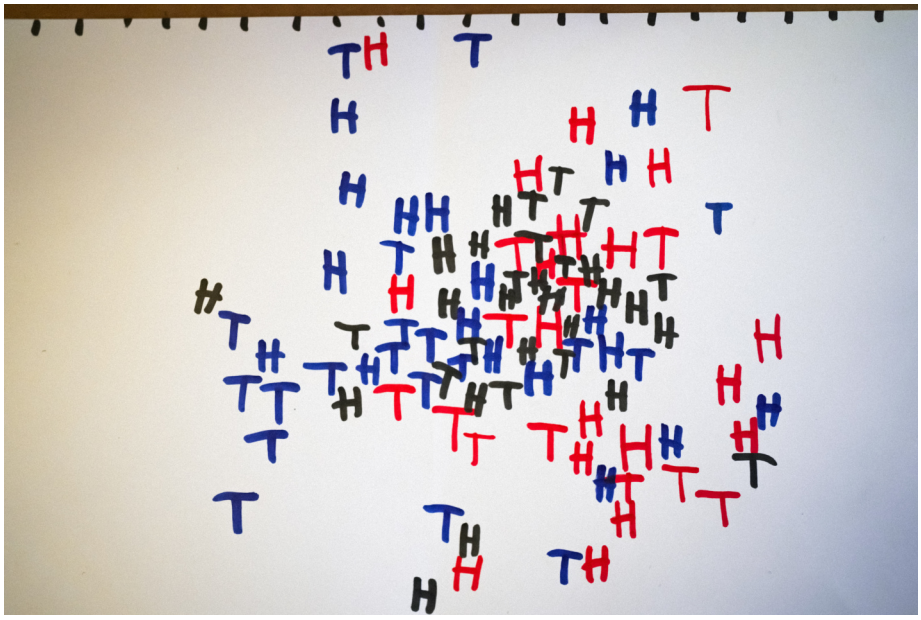


Figure 8: Close-up of the area with the highest drop frequency after image enhancement.

4 The Methodology After the Data Collection

4.1 Data Visualization

Prior to data preprocessing, an initial visualization was created to understand the distribution of the coin's landing positions. This step was crucial for gaining insights into the spatial characteristics of the dataset.

The visualization process involved the following steps:

1. Loading the experimental data from a CSV file into a Pandas DataFrame.
2. Creating a figure with subplots to display the coin landing positions.
3. Adjusting the axes limits to match the pixel dimensions of the landing area and inverting the y-axis to align with the image coordinate system.
4. Mapping the initial conditions and outcomes to specific colors and labels, and plotting each data point accordingly.
5. Adding a custom legend to differentiate between the initial conditions and final outcomes.
6. Saving the plot as a PDF file.

This visualization provided a foundational understanding of the data, aiding in the subsequent preprocessing and analysis phases.

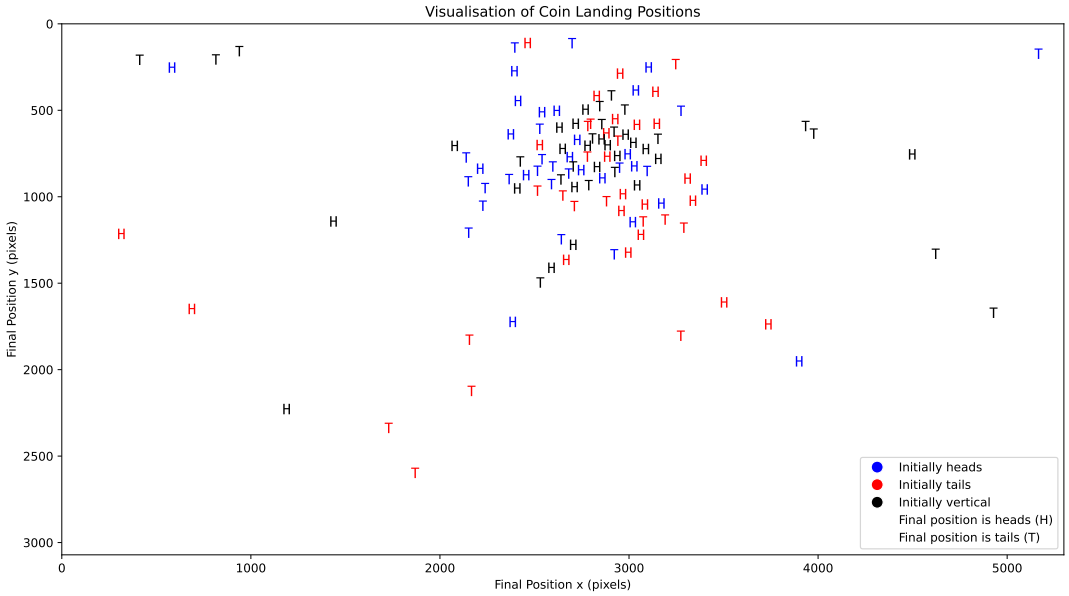


Figure 9: Initial visualization of coin landing positions.

4.2 Data Preprocessing

Data preprocessing is a critical step in preparing the dataset for training the neural network. The raw data collected from the experiment was loaded into a Pandas DataFrame from a CSV file. Categorical variables were encoded using LabelEncoder, and features and targets were separated accordingly. The 'Initial' column was transformed into a format suitable for the neural network, and the position data was normalized using MinMaxScaler to ensure that all inputs were on a similar scale.

```
data_path = 'experiment_data.csv'
coin_data = pd.read_csv(data_path)

# Encode categorical data
encoder = LabelEncoder()
coin_data['Initial'] = encoder.fit_transform(coin_data['Initial'])
coin_data['Final Heads or Tails'] = encoder.fit_transform(coin_data['Final Heads
 or Tails'])

# Separate features and targets
X = coin_data[['Initial']].values
# Only use 'Initial' as feature to avoid data leakage
y_class = coin_data['Final Heads or Tails'].values
# Classification target

# Scale the target variables for regression separately
scaler = MinMaxScaler()
scaler_y = MinMaxScaler()
y_reg = scaler.fit_transform(coin_data[['Final Position x',
 'Final Position y']].values)
# Regression targets
```

4.3 Neural Network Architecture

The architecture of the neural network is defined using TensorFlow and Keras. The model consists of an input layer followed by dense layers with ReLU activation functions. Two separate output layers are used: one for classification with a sigmoid activation, and one for regression with a linear activation. The code defining the model is as follows:

```
def create_model():
    input_layer = Input(shape=(X.shape[1],))
    shared_dense1 = Dense(64,
    activation='relu')(input_layer)
    shared_dense2 = Dense(64,
    activation='relu')(shared_dense1)
    classification_output = Dense(1, activation='sigmoid',
    name='class_output')(shared_dense2)
```

```
    regression_output = Dense(2, activation='linear',
    name='reg_output')(shared_dense2)
    model = Model(inputs=input_layer,
    outputs=[classification_output, regression_output])
    model.compile(optimizer='adam',
              loss={'class_output':'binary_crossentropy',
              'reg_output': 'mean_squared_error'},
              metrics={'class_output':
              ['accuracy', AUC(name='auc')],
              'reg_output': ['mse']})
    return model
```

4.4 Training and Evaluation

The neural network was trained using 10-fold cross-validation to ensure that the model's performance was consistent across different subsets of the data. Each fold of the cross-validation involved training the model and evaluating it on a separate test set. The model's performance was assessed using a variety of metrics, including accuracy, mean squared error (MSE), and the area under the receiver operating characteristic (ROC) curve.

```
# 10-Fold Cross-Validation with Enhanced Metrics and Visualization
fold_no = 1
for train, test in kfold.split(X):  # Split based only on the feature matrix X
    print(f"\n--- Processing Fold {fold_no} ---")
    model = create_model()
    print("Training the model (model.fit)...")

    # Fit the model using both classification and regression targets
    history = model.fit(X[train], {'class_output':
    y_class[train], 'reg_output': y_reg[train]},
                        validation_data=(X[test],
                        {'class_output': y_class[test],
                        'reg_output': y_reg[test]}),
                        epochs=100, batch_size=8, verbose=0)

    print("Evaluating the model (model.evaluate)...")
    scores = model.evaluate(X[test], {'class_output':
    y_class[test], 'reg_output': y_reg[test]}, verbose=0)
```

4.5 Visualizations, Metrics, and Predictions

A suite of Python functions was implemented to visualize various aspects of the neural network's training process and performance metrics. The functions include:

- `visualize_data`: To generate scatter plots visualizing the distribution of the training and test data across the folds.

- `visualize_roc_curve`: To plot the Receiver Operating Characteristic (ROC) curve and calculate the Area Under Curve (AUC) for classification outcomes.
- `plot_regression_metrics`: To illustrate the regression model's performance by plotting Mean Squared Error (MSE) over training epochs.
- `plot_classification_accuracy`: For displaying the accuracy of the classification model over the epochs.
- `plot_auc`: To show the change in the AUC metric over the epochs.
- `generate_classification_report` and `print_confusion_matrix`: To provide detailed classification metrics and confusion matrix data.
- `compare_classification_predictions` and `compare_regression_predictions`: To contrast the predicted results with the actual values, offering a side-by-side comparison.

Each of these functions plays a crucial role in interpreting the model's learning behavior and its predictive performance, ensuring a robust analysis of the results.

4.6 Additional Visuals for a Specific Fold

In addition to the metrics and graphs that are given in every iteration, a specific fold in our cross-validation process was selected for in-depth visual analysis to illustrate the neural network's predictive capabilities with greater clarity. The visualizations for this fold were particularly important for presentations and discussions, highlighting the comparison between predicted and actual results.

1. **Regression Predictions Visualization:** The visualize_regression_comparisons function was used to graphically compare the actual landing positions of the coin with the positions predicted by the neural network. This function plots each predicted versus actual position for regression targets, allowing us to visually assess the model's performance in predicting the landing coordinates of the coin.

   ```
   visualize_regression_comparisons(regression_comparison_df,
   X, y_reg, test, scaler, fold_no=4)
   ```

2. **Classification Predictions Visualization:** Similarly, the visualize_classification_comparisons function provided side-by-side images of the actual and predicted outcomes of the coin toss (heads or tails). This comparative visualization serves to demonstrate the classification accuracy of the neural network.

   ```
   visualize_classification_comparisons(classification_comparison_df, fold_no=4)
   ```

These functions, tailored to the specific fold under examination, play a crucial role in the detailed analysis of the neural network's predictions. They enable a clear visual representation of the accuracy of the network's predictions in both classification and regression aspects of the coin toss outcomes.

5 Code Implementation

Preparing the code, Jupyter notebook platform was used. The dataset, photos taken, .ipynb file together with all the outputs are available via this drive link.

Library Dependencies

The project utilized a number of Python libraries to facilitate the implementation of the neural network, data preprocessing, visualization, and statistical analysis. Key libraries included:

- `matplotlib.pyplot` and `matplotlib.image` for creating visualizations and processing images.
- `pandas` for data manipulation and analysis.
- `numpy` for numerical computing.
- `seaborn` for advanced statistical graphics.
- `scikit-learn` for machine learning tools like cross-validation, data scaling, and performance metrics.
- `tensorflow.keras` for building and training the neural network model, defining the architecture, and compiling the model.

These libraries were integral to the project, each contributing specific functionalities that, when combined, allowed for the successful execution of the experiment and subsequent analysis.

Full Code

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import matplotlib.pyplot as plt
import pandas as pd
from matplotlib.lines import Line2D
import os

# Load your data
# Make sure to update the path to where your CSV file is located
data_path = 'experiment_data.csv'
coin_data = pd.read_csv(data_path)

# Define the directory for saving the PDF
pdf_path = 'general_visualizations_of_data'
```

```
# Create the directory if it doesn't exist
os.makedirs(pdf_path, exist_ok=True)


# Create a figure and a set of subplots
fig, ax = plt.subplots(figsize=(15, 8))

# Adjusting the axes limits to match the photo's pixel dimensions
ax.set_xlim(0, 5300)
ax.set_ylim(0, 3071)

# Inverting the y-axis to match the image coordinate system
ax.invert_yaxis()

# Color and label mapping
color_mapping = {'Tails': 'red', 'Heads': 'blue', 'Vertical': 'black'}
label_mapping = {'H': 'H', 'T': 'T'}

# Plotting each data point with the appropriate color and label as text
for _, row in coin_data.iterrows():
    ax.text(row['Final Position x'], row['Final Position y'],
            label_mapping[row['Final Heads or Tails']],
            color=color_mapping[row['Initial']],
            fontsize=12, ha='center', va='center', family='monospace')

# Creating custom legends for initial conditions and final outcomes
legend_elements = [
    Line2D([0], [0], marker='o', color='w',
    label='Initially heads', markerfacecolor='blue', markersize=10),
    Line2D([0], [0], marker='o', color='w',
    label='Initially tails', markerfacecolor='red', markersize=10),
    Line2D([0], [0], marker='o', color='w',
    label='Initially vertical', markerfacecolor='black', markersize=10),
    Line2D([0], [0],  color='w', label='Final position
     is heads (H)', markerfacecolor='w', markeredgecolor='black', markersize=15),
    Line2D([0], [0],  color='w', label='Final position
    is tails (T)', markerfacecolor='w', markeredgecolor='black', markersize=15)
]

# Adding the legend to the plot
ax.legend(handles=legend_elements, loc='lower right')

# Set the plot labels and title
ax.set_xlabel('Final Position x (pixels)')
ax.set_ylabel('Final Position y (pixels)')
ax.set_title('Visualisation of Coin Landing Positions')
```

```
# Save the plot as a PDF file
plt.savefig(f'{pdf_path}/coin_landing_positions.pdf')

# Display the plot
plt.show()


# In[2]:


import os
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import mean_squared_error,
mean_absolute_error, r2_score, roc_curve, auc, classification_report, confusion_matrix
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.metrics import AUC
from tensorflow.keras.callbacks import EarlyStopping

# Define paths for saving the plots
classification_plot_path = 'classification_plots'
regression_plot_path = 'regression_plots'
general_visualizations_path = 'general_visualizations_of_folds'

# Create directories if they don't exist
os.makedirs(classification_plot_path, exist_ok=True)
os.makedirs(regression_plot_path, exist_ok=True)
os.makedirs(general_visualizations_path, exist_ok=True)

def visualize_data(y_reg, train, test, fold_no):
    plt.figure(figsize=(10, 8))

    # Visualize the normalized position data directly from y_reg
    plt.scatter(y_reg[train, 0], y_reg[train, 1],
    marker='x', color='black', label='Train Data')
    plt.scatter(y_reg[test, 0], y_reg[test, 1],
    marker='o', color='blue', label='Test Data')

    plt.title(f'Train vs Test Data Distribution - Fold {fold_no}')
    plt.xlabel('Final Position x (Normalized)')
```

```
    plt.ylabel('Final Position y (Normalized)')
    plt.gca().invert_yaxis()
    plt.legend()
    plt.savefig(f'{general_visualizations_path}/
    train_test_data_fold_{fold_no}.pdf')
    plt.show()

def visualize_roc_curve(model, y_class, test, fold_no):
    # Obtain probabilistic outputs from the model for the test set
    y_pred_class_prob = model.predict(X[test])[0]

    # Compute the ROC curve and AUC
    fpr, tpr, thresholds = roc_curve(y_class[test], y_pred_class_prob)
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve
    plt.figure(figsize=(10, 8))
    plt.plot(fpr, tpr, color='red',
    label=f'ROC Curve (area = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='black', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver Operating Characteristic - Fold {fold_no}')
    plt.legend(loc="lower right")
    plt.savefig(f'{classification_plot_path}/roc_curve_fold_{fold_no}.pdf')
    plt.show()

def plot_regression_metrics(history, fold_no):
    # Plot MSE over epochs for both training and validation sets
    plt.figure(figsize=(10, 8))
    plt.plot(history.history['reg_output_mse'],
    color='red', label='Train MSE')
    plt.plot(history.history['val_reg_output_mse'],
    color='blue', label='Validation MSE')
    plt.title(f'Regression - MSE Over Epochs (Fold {fold_no})')
    plt.xlabel('Epochs')
    plt.ylabel('Mean Squared Error')
    plt.legend()
    plt.savefig(f'{regression_plot_path}/mse_over_epochs_fold_{fold_no}.pdf')
    plt.show()


# Plotting Classification Accuracy Over Epochs
def plot_classification_accuracy(history, fold_no):
    plt.figure(figsize=(10, 8))
    plt.plot(history.history['class_output_accuracy'],
```

```
    color='red', label='Train Accuracy')
    plt.plot(history.history['val_class_output_accuracy'],
    color='blue', label='Validation Accuracy')
    plt.title(f'Classification - Accuracy Over Epochs (Fold {fold_no})')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.savefig(f'{classification_plot_path}/accuracy_over_epochs_fold_{fold_no}.pdf')
    plt.show()

# Plotting AUC Over Epochs
def plot_auc(history, fold_no):
    plt.figure(figsize=(10, 8))
    plt.plot(history.history['class_output_auc'], color='red',
     label='Train AUC')
    plt.plot(history.history['val_class_output_auc'], color='blue',
    label='Validation AUC')
    plt.title(f'Classification - AUC Over Epochs (Fold {fold_no})')
    plt.xlabel('Epochs')
    plt.ylabel('AUC')
    plt.legend()
    plt.savefig(f'{classification_plot_path}/auc_over_epochs_fold_{fold_no}.pdf')
    plt.show()

# Generating and Displaying Classification Report
def generate_classification_report(model, y_class, test, fold_no):
    y_pred_class_prob = model.predict(X[test])[0]
    y_pred_class = (y_pred_class_prob > 0.5).astype(int)
    class_report = classification_report(y_class[test], y_pred_class,
    target_names=['Heads', 'Tails'])
    print(f'Fold {fold_no} - Classification Report:\n{class_report}')

# Displaying Confusion Matrix
def print_confusion_matrix(model, X, y_class, test, fold_no):
    y_pred_class = (model.predict(X[test])[0] > 0.5).astype(int)
    cm = confusion_matrix(y_class[test], y_pred_class)
    print(f'Fold {fold_no} - Confusion Matrix:\n{cm}')


def compare_classification_predictions(model, X, y_class, test, fold_no):
    y_pred_class_prob = model.predict(X[test])[0]
    y_pred_class_keras = (y_pred_class_prob > 0.5).astype(int)

    # Decode the initial states from numeric to categorical
    initial_states = (X[test, 0].astype(int))
```

```
    manual_accuracy = np.mean(y_pred_class_keras == y_class[test])
    comparison = np.column_stack((initial_states, y_class[test],
     y_pred_class_keras, y_pred_class_prob))
    comparison_df = pd.DataFrame(comparison, columns=['Initial State',
    'Actual Label', 'Predicted Label', 'Probabilistic Output'])

    print(f'Fold {fold_no} - Classification Predictions Comparison:
    \n{comparison_df.head(12)}')

    return comparison_df

def compare_regression_predictions(model, X, y_reg, test, scaler, fold_no):
    y_pred_reg = model.predict(X[test])[1]

    # Scale back the predictions to the original scale
    y_pred_reg_rescaled = scaler.inverse_transform(y_pred_reg)
    y_reg_rescaled = scaler.inverse_transform(y_reg[test])

    # Decode the initial states from numeric to categorical
    initial_states = (X[test, 0].astype(int))

    # Display side-by-side comparison
    comparison = np.column_stack((initial_states, y_reg_rescaled, y_pred_reg_rescaled))
    comparison_df = pd.DataFrame(comparison, columns=['Initial State',
    'Actual X', 'Actual Y', 'Predicted X', 'Predicted Y'])

    print(f'\nFold {fold_no} - Creating the Regression Predictions Comparison Table...\n')
    print(f'Fold {fold_no} - Regression Predictions Comparison:\n', comparison_df.head(12))

    return comparison_df


# In[3]:


import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import KFold, train_test_split
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import mean_absolute_error, r2_score, roc_curve,
auc, classification_report, confusion_matrix
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.metrics import AUC

plot_path = 'fold_4_visualisations_of_predictions'
os.makedirs(plot_path, exist_ok=True)

# Function to visualize regression comparisons one by one
def visualize_regression_comparisons(comparison_df, X, y_reg,
test, scaler, fold_no):
    # Convert normalized regression targets back to pixel coordinates
    y_test_pixels = scaler.inverse_transform(y_reg[test])

    for i in range(len(comparison_df)):
        plt.figure(figsize=(10, 8))

        # Extract actual and predicted coordinates for the current instance
        actual_x, actual_y = comparison_df.iloc[i]['Actual X'],
        comparison_df.iloc[i]['Actual Y']
        predicted_x, predicted_y = comparison_df.iloc[i]
        ['Predicted X'], comparison_df.iloc[i]['Predicted Y']

        # Plot all test data points with lower alpha
        plt.scatter(y_test_pixels[:, 0], y_test_pixels[:, 1],
        color='grey', alpha=0.5, label='Other Test Data')

        # Highlight the current point of interest
        plt.scatter(actual_x, actual_y, color='blue', label='Actual', zorder=5)
        plt.scatter(predicted_x, predicted_y, color='red', label='Predicted', zorder=5)

        plt.xlabel('X Position (pixels)')
        plt.ylabel('Y Position (pixels)')
        plt.title(f'Regression Prediction Comparison - Fold {fold_no}, Instance {i+1}')
        plt.xlim(0, 5300)
        plt.ylim(0, 3071)
        plt.gca().invert_yaxis()  # To match the original image coordinate system
        plt.legend()
        plt.savefig(f'{plot_path}/regression_comparison_fold_{fold_no}_instance_{i+1}.pdf')
        plt.show()

def visualize_classification_comparisons(comparison_df, fold_no):
    # Load images for heads and tails
    heads_img = mpimg.imread('coin_heads_tails/heads.jpg')
    tails_img = mpimg.imread('coin_heads_tails/tails.jpg')

    # Plotting the first few comparisons
```

```
    fig, axs = plt.subplots(nrows=comparison_df.shape[0], ncols=2, figsize=(5, 20))

    for i in range(comparison_df.shape[0]):
        actual_label = comparison_df.iloc[i]['Actual Label']
        predicted_label = comparison_df.iloc[i]['Predicted Label']

        axs[i, 0].imshow(heads_img if actual_label == 0 else tails_img)
        axs[i, 0].set_title(f"Actual: {'Heads' if actual_label == 0 else 'Tails'}")
        axs[i, 0].axis('off')

        axs[i, 1].imshow(heads_img if predicted_label == 0 else tails_img)
        axs[i, 1].set_title(f"Predicted: {'Heads' if predicted_label == 0 else 'Tails'}")
        axs[i, 1].axis('off')

    plt.tight_layout()
    plt.savefig(f'{plot_path}/classification_comparison_fold_{fold_no}.pdf')
    plt.show()


# In[4]:


import os
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import mean_absolute_error,
r2_score, roc_curve, auc, classification_report, confusion_matrix
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense

# Load and preprocess the dataset
data_path = 'experiment_data.csv'
coin_data = pd.read_csv(data_path)

# Encode categorical data
encoder = LabelEncoder()
coin_data['Initial'] = encoder.fit_transform(
coin_data['Initial'])
coin_data['Final Heads or Tails'] = encoder.fit_transform(
coin_data['Final Heads or Tails'])

# Separate features and targets
```

```
X = coin_data[['Initial']].values
# Only use 'Initial' as feature to avoid data leakage
y_class = coin_data['Final Heads or Tails'].values
# Classification target

# Scale the target variables for regression separately
scaler = MinMaxScaler()
scaler_y = MinMaxScaler()
y_reg = scaler.fit_transform(coin_data[['Final Position x',
'Final Position y']].values)  # Regression targets


# Define K-Fold Cross Validator and Model Creation Function
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

def create_model():
    # Model architecture
    print("Creating the model...")
    input_layer = Input(shape=(X.shape[1],))
    shared_dense1 = Dense(64, activation='relu')(input_layer)
    shared_dense2 = Dense(64, activation='relu')(shared_dense1)
    classification_output = Dense(1, activation=
    'sigmoid', name='class_output')(shared_dense2)
    regression_output = Dense(2, activation='linear',
    name='reg_output')(shared_dense2)
    # Compile the model
    model = Model(inputs=input_layer, outputs=[classification_output,
     regression_output])
    print("Compiling the model (model.compile)...")
    model.compile(optimizer='adam',
              loss={'class_output': 'binary_crossentropy',
               'reg_output': 'mean_squared_error'},
              metrics={'class_output': ['accuracy',
               AUC(name='auc')], 'reg_output': ['mse']})
    return model

# 10-Fold Cross-Validation with Enhanced Metrics and Visualization
fold_no = 1
for train, test in kfold.split(X):  # Split based only on the feature matrix X
    print(f"\n--- Processing Fold {fold_no} ---")
    model = create_model()
    print("Training the model (model.fit)...")

    # Fit the model using both classification and regression targets
    history = model.fit(X[train], {'class_output': y_class[train],
    'reg_output': y_reg[train]},
```

```
                    validation_data=(X[test], {'class_output':
                     y_class[test], 'reg_output': y_reg[test]}),
                    epochs=100, batch_size=8, verbose=0)

print("Evaluating the model (model.evaluate)...")
scores = model.evaluate(X[test], {'class_output': y_class[test],
'reg_output': y_reg[test]}, verbose=0)


# Define paths for saving the plots
classification_plot_path = 'classification_plots'
regression_plot_path = 'regression_plots'
general_visualizations_path = 'general_visualizations_of_folds'

# Create directories if they don't exist
os.makedirs(classification_plot_path, exist_ok=True)
os.makedirs(regression_plot_path, exist_ok=True)
os.makedirs(general_visualizations_path, exist_ok=True)

# Supporting functions for visualizations and metrics calculations
visualize_data(y_reg, train, test, fold_no)
visualize_roc_curve(model, y_class, test, fold_no)
plot_regression_metrics(history, fold_no)
plot_classification_accuracy(history, fold_no)
plot_auc(history, fold_no)
generate_classification_report(model, y_class, test, fold_no)
print_confusion_matrix(model, X, y_class, test, fold_no)
classification_comparison_df = compare_classification_predictions(model,
X, y_class, test, fold_no)
regression_comparison_df = compare_regression_predictions(model,
X, y_reg, test, scaler, fold_no)

# Print evaluation results
print(f"\nScores for Fold {fold_no}:")
for name, value in zip(model.metrics_names, scores):
    print(f"  - {name}: {value}")
selected_no = 4
if fold_no == selected_no:
    classification_comparison_df = compare_classification_predictions(model,
    X, y_class, test, fold_no=selected_no)
    regression_comparison_df = compare_regression_predictions(model, X,
     y_reg, test, fold_no=selected_no,scaler=scaler)
    visualize_classification_comparisons(classification_comparison_df,
    fold_no=selected_no)
    visualize_regression_comparisons(regression_comparison_df, X,
     y_reg, test, scaler, fold_no=selected_no)
```

```
    fold_no += 1
# In[5]:
model.summary()

# In[6]:
model.save('10_fold_model.h5')

# In[7]:
print(history.history.keys())

# In[8]:
from sklearn.preprocessing import LabelEncoder

labels = ['Heads', 'Tails']
encoder = LabelEncoder()
encoder.fit(labels)

# Mapping from label to numeric value
label_mapping = dict(zip(encoder.classes_, encoder.transform(encoder.classes_)))
print("Label to Numeric Mapping:", label_mapping)
```

6 Results

6.1 General Performance Across Folds

During the cross-validation process, each fold was analyzed to evaluate the model's performance. As Fold 4 displayed more variations, implying non-constant prediction of heads, an evaluation focused on visual observation was deemed more relevant than a mere comparison of 'performance'. Given all the simulations and visualizations, opting for a 'better' fold would be illogical, suggesting the model's effectiveness is tied to the specifics of test and train data, an outcome that is undesirable due to its dependency on the folding process. The analysis, while not achieving highly accurate predictions, aligned with the author's initial expectations. The results highlighted the inherent randomness of the coin toss outcomes, particularly in terms of the final condition (heads or tails) and the landing position. It was observed that the model tended to predict landing positions where most coins landed, but with limited correlation to the initial condition of the coin. This outcome emphasizes the need for incorporating more features into the model to capture the complexities of the coin toss process and improve prediction accuracy.

6.1.1 Fold 4 Analysis

Classification Report: The model achieved a precision of 0.62 for Heads and 0.25 for Tails, with an overall accuracy of 0.50, indicating some improvement in classification compared to earlier folds.
Confusion Matrix: This showed a modest improvement in distinguishing outcomes, with a mix of correct classifications and misclassifications, reflecting the challenges in achieving high predictive accuracy.

MSE and Classification Accuracy: The Mean Squared Error (MSE) for the regression output was 0.0288. Given the normalized scale of the data, this suggests somewhat a level of predictive accuracy in estimating the landing positions of the coin, where the predictions are at the place where the coin usually drops, again, little to do with the initial condition. The classification accuracy stood at 50%, indicating the model is guessing randomly.
Regression Predictions Comparison: The comparison showed the model's moderate ability to predict landing positions. Although there was an alignment between actual and predicted positions, the precision was not high.
Classification Predictions Comparison: This comparison revealed a mixed performance by the model, with instances of both correct predictions and misclassifications. The probabilistic outputs further demonstrated the model's varying confidence in its predictions.
ROC Curve and AUC: The Receiver Operating Characteristic (ROC) curve and the Area Under Curve (AUC) provided insights into the model's capability to distinguish between the two classification outcomes (Heads and Tails). An AUC of 0.46875 suggested a performance slightly below randomness.
Visualizations and Metrics: The visualizations created for Fold 4, including ROC curve, regression metrics over epochs, and classification accuracy over epochs, offered a detailed view of the model's learning and performance dynamics.

6.2 Special Visuals for Fold 4

Fold 4 featured several specific instances that offered deeper insights into the model's performance. Detailed visual comparisons for these instances are provided, illustrating the nuanced behavior of the model under varying conditions. Figures 10 to 14 showcases the regression predictions whereas 15 and 16 ilustrates the classification predictions. Given the test data number of 12 for this fold, all the classification predictions for Fold 4 were given in this report. As for the regressions, first 5 was given here, rest is available, again, via the aforementioned link. The appendix includes regular visualizations for Fold 4. Together with the special visualizations, they provide additional context and depth to the textual analysis presented in this section.

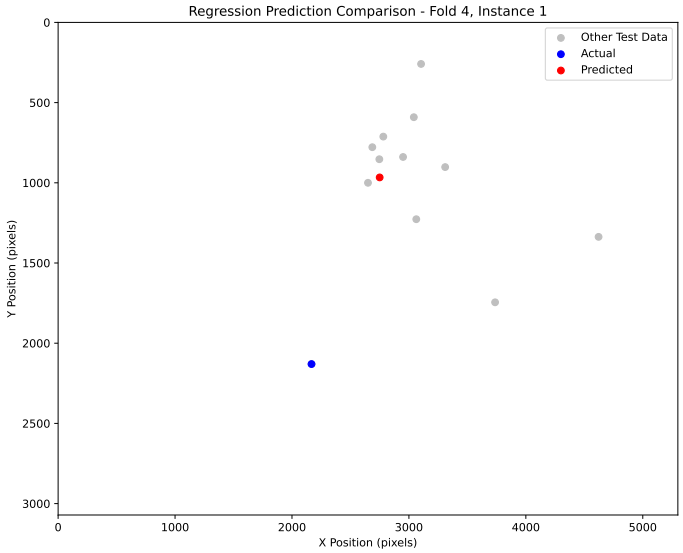


Figure 10: Instance 1: Comparison of actual vs. predicted regression targets for Fold 4.

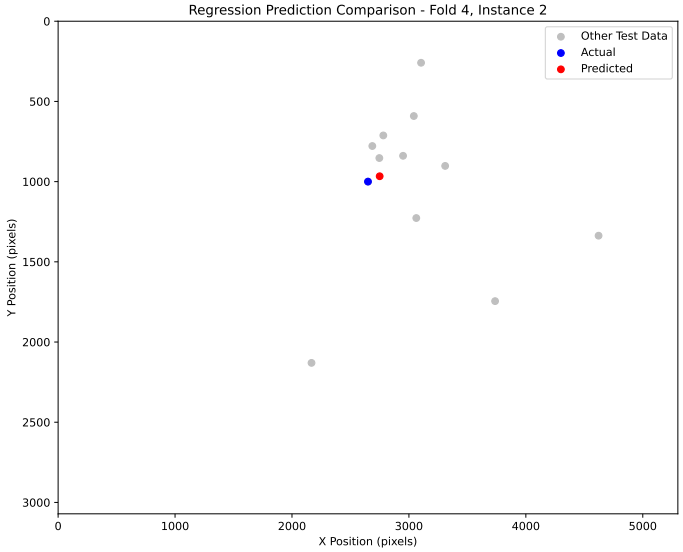


Figure 11: Instance 2: Comparison of actual vs. predicted regression targets for Fold 4.

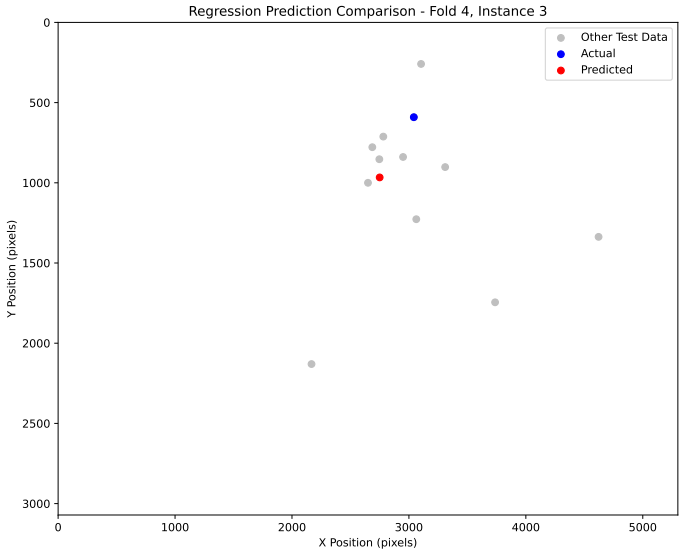


Figure 12: Instance 3: Comparison of actual vs. predicted regression targets for Fold 4.

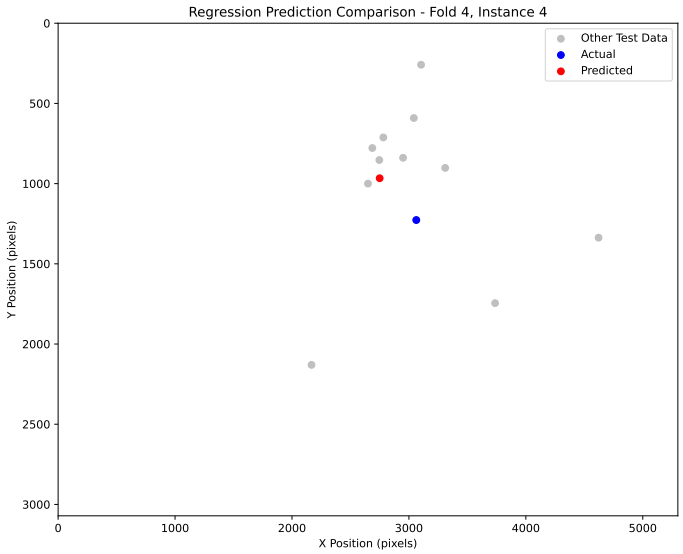


Figure 13: Instance 4: Comparison of actual vs. predicted regression targets for Fold 4.

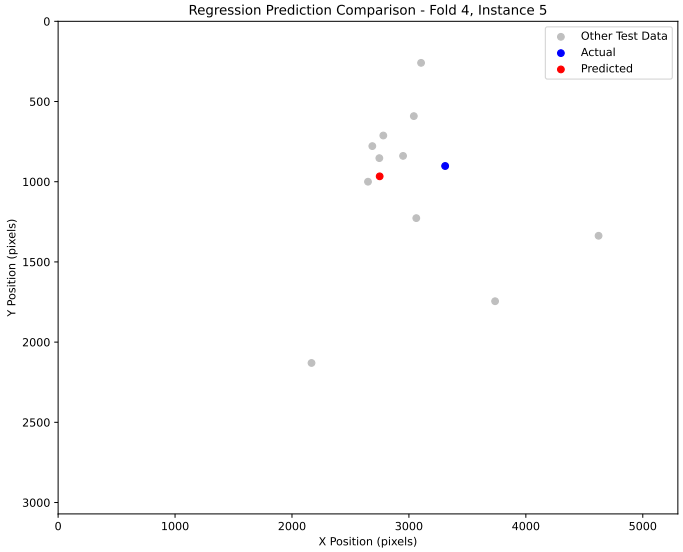


Figure 14: Instance 5: Comparison of actual vs. predicted regression targets for Fold 4.

Figure 15: Part 1: Side-by-side comparison of actual vs. predicted classification outcomes for Fold 4.

Figure 16: Part 2: Side-by-side comparison of actual vs. predicted classification outcomes for Fold 4.

7 Conclusion

This study has demonstrated the inherent challenges in predicting the outcomes of a coin toss using a neural network with a limited feature set. The analysis across different folds, especially the in-depth examination of fold 4, revealed a tendency of the model to favor 'heads' as the outcome. This bias reflects the higher frequency of 'heads' in the dataset, underscoring the model's reliance on the available data.

However, not surprisingly, no significant connection was observed between the initial condition (heads, tails, or vertical) and the final landing state of the coin. The model's predictions did not consistently correlate with the initial conditions, suggesting that the landing outcome of a coin toss is predominantly random and influenced by factors not captured in the current feature set. Moreover, the spatial analysis of the coin's landing positions revealed that the model tends to predict landing positions within a specific proximity, likely influenced by the frequent landing spots in the training data. This observation indicates the model's limitation in capturing the full range of possible outcomes, highlighting the need for more nuanced features or a different modeling approach. Also, it is observed that the results were very dependent on the folding of that instance.

An interesting aspect of this study emerged when an unintentional deviation in feature selection led to seemingly improved results, particularly for the regression task. In the initial model configuration, the features mistakenly included not only the coin's initial state (heads, tails, or vertical) but also its final coordinates (X, Y positions). This erroneous inclusion of the target variables (final positions) as part of the input features created a scenario where the model had direct access to the information it was supposed to predict. As a result, the model's performance in predicting the final landing positions of the coin appeared exceptionally accurate, bordering on perfection.

However, this was a clear case of data leakage, where the model was inadvertently "informed" about the outcome during training. In practical terms, the model was merely echoing back the provided coordinates rather than making genuine predictions. Interestingly, this flawed approach also led to a slight improvement in classification accuracy, suggesting a potential correlation between the coin's landing position and its final state (heads or tails). This observation might hint at a subtle underlying pattern in the physical dynamics of the coin's toss.

Nonetheless, it's crucial to acknowledge that this improved performance was artificial, stemming from a methodological error rather than genuine predictive capability. In predictive modeling, ensuring the integrity and appropriateness of input features is paramount to avoid misleading results, as vividly demonstrated in this accidental experiment. The key takeaway is that while feature selection can significantly influence model performance, it must be executed with careful consideration to maintain the validity and reliability of the model's predictions.

In conclusion, while the neural network showed some capability in predicting outcomes, the findings emphasize the randomness of a coin toss and the importance of feature selection in predictive modeling. The study serves as a reminder of the complexities in modeling seemingly simple physical phenomena and the need for careful consideration of feature relevance and data quality.

A Appendix

Here are the visualizations generated for Fold 4 of the cross-validation process excluding the special visualizations for Fold 4 as they are given in the results section directly. Rest of the outputs are available in the aforementioned link. Of course, one can run the code himself to see the results.

A.1 Graphs for Fold 4

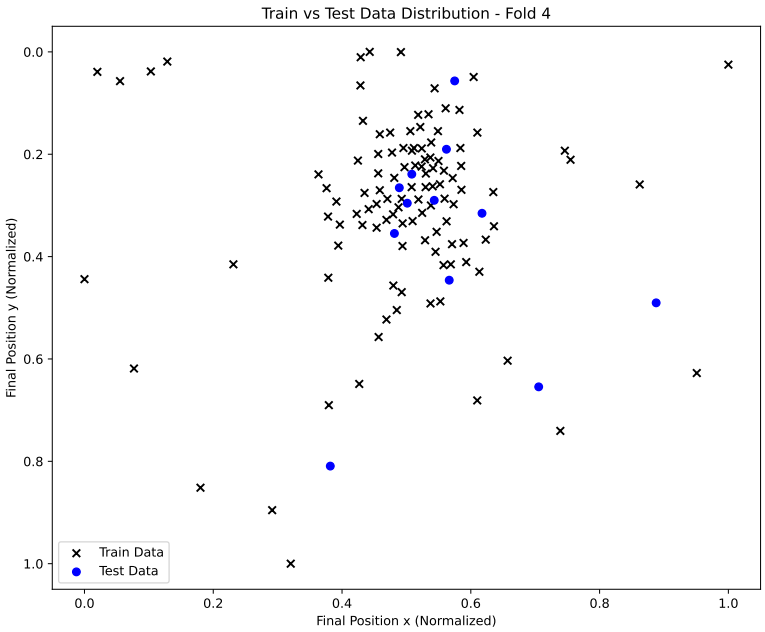


Figure 17: Train vs Test Data Distribution for Fold 4.

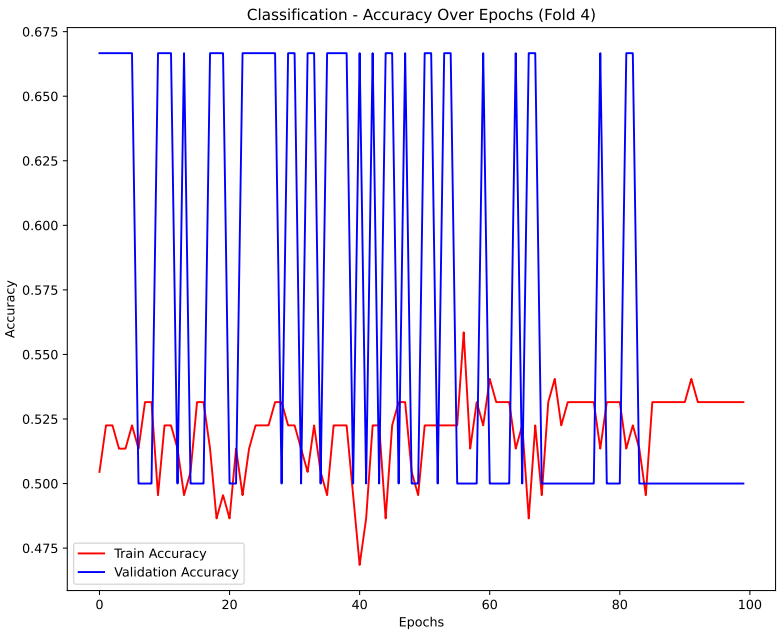


Figure 18: Accuracy over epochs for Fold 4.

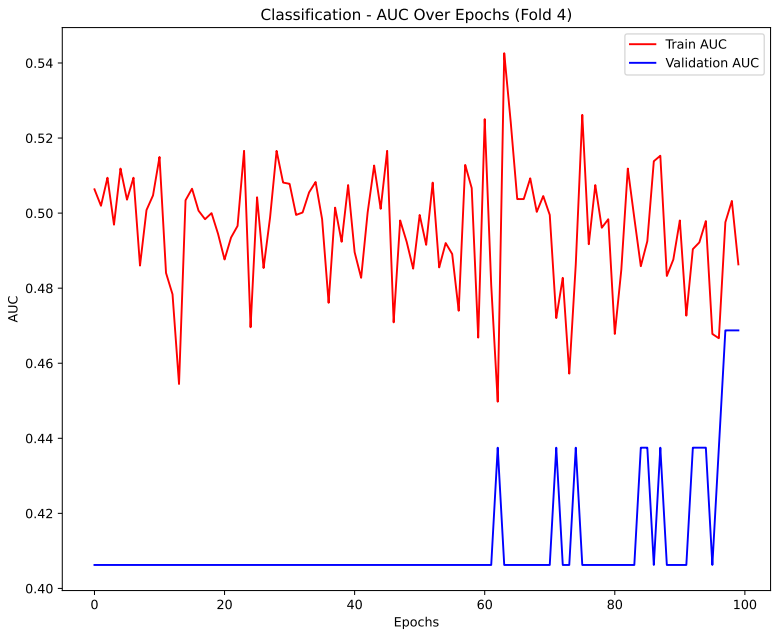


Figure 19: AUC over epochs for Fold 4.

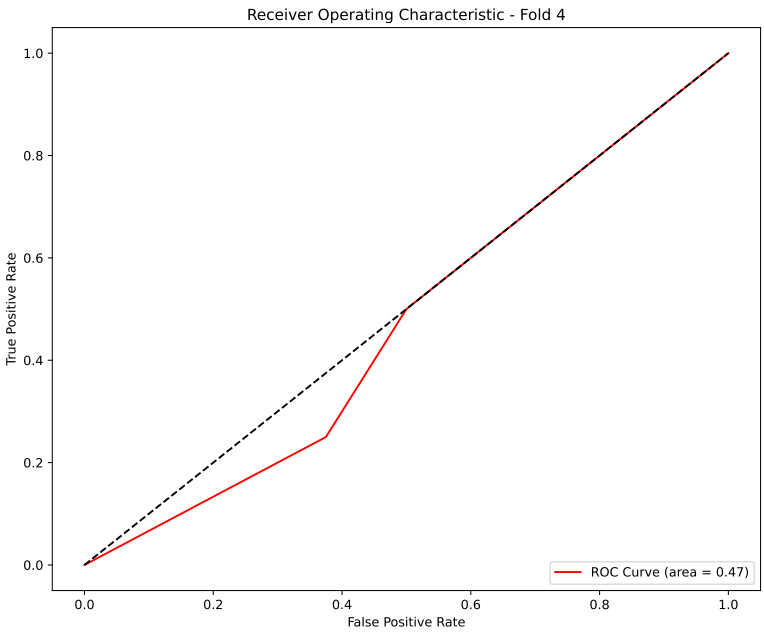


Figure 20: ROC Curve for Fold 4.

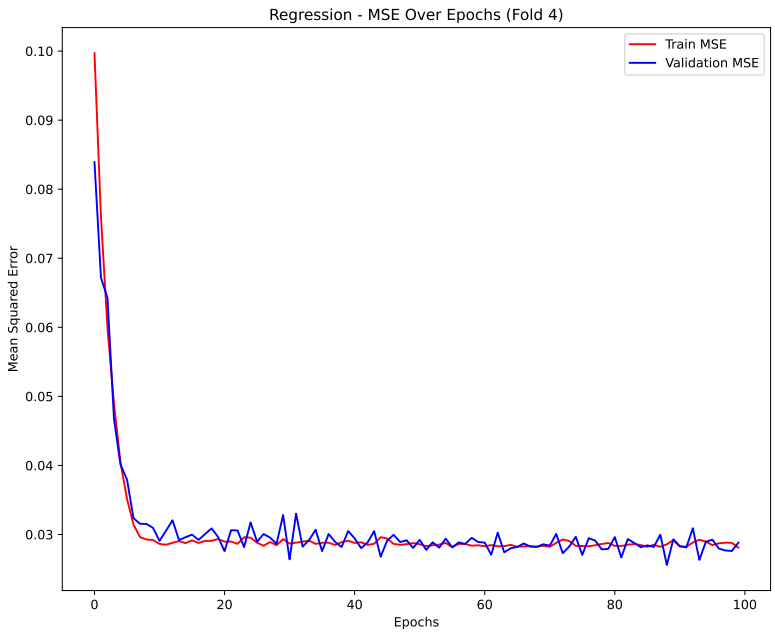


Figure 21: MSE over epochs for Fold 4.