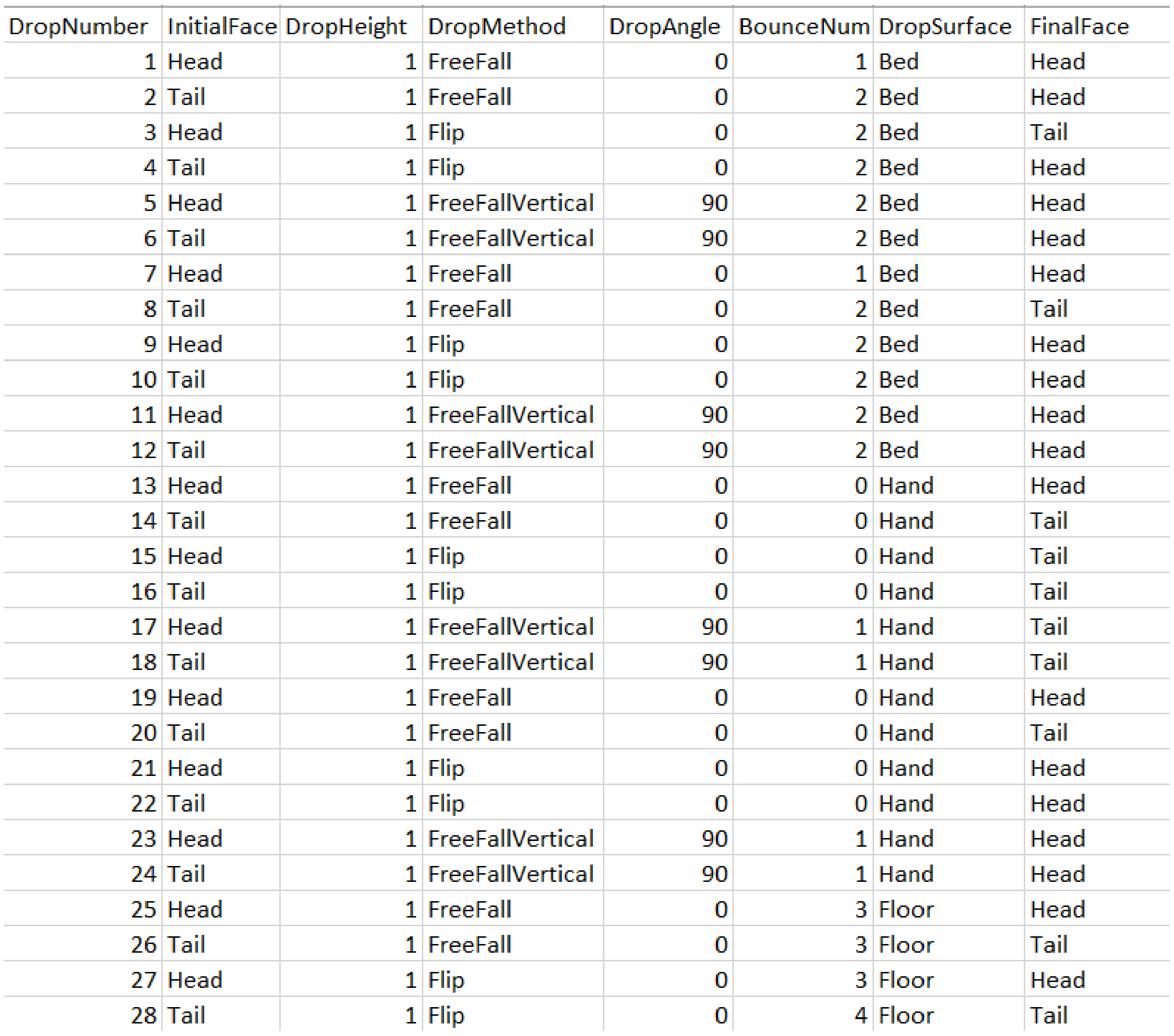
Artificial Neural Networks

Final Exam – Coin Drop Prediction

Dündar Emre Özbirecikli
040210761

1) Data Collection

Data is collected by dropping the 1 TL coin from 1 meter for 100 times to different surfaces with different drop methods and different initial states. Drop methods are dropping vertically, flipping and dropping with free fall. Initial state shows which side is facing up before dropping. Drop surfaces are chosen as bed, hand and floor and “BounceNum” variable shows how many times the coin bounces after it dropped into the surface. Structure of the excel file is shown in Figure 1.1.

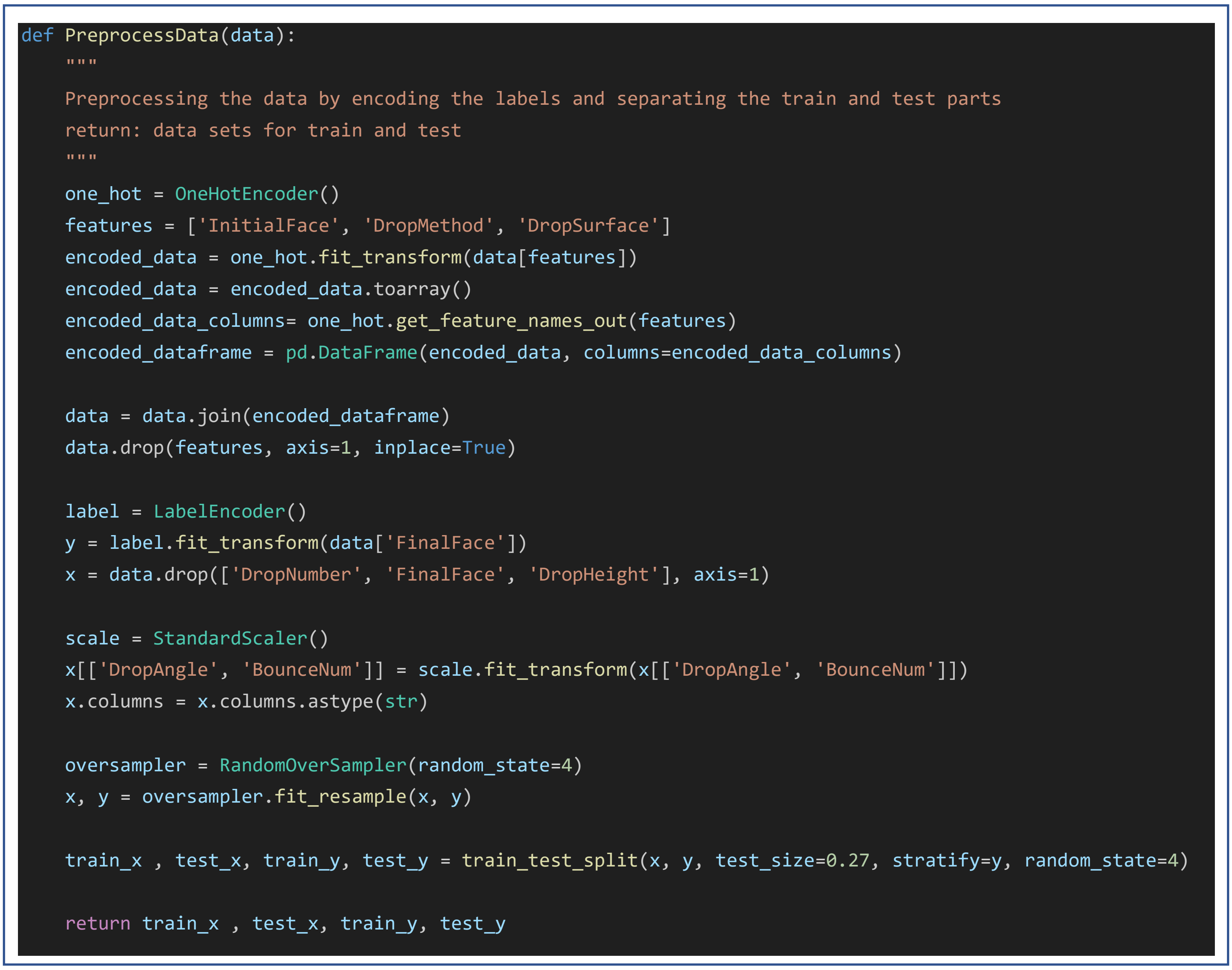


DropNumber	InitialFace	DropHeight	DropMethod	DropAngle	BounceNum	DropSurface	FinalFace
1	Head	1	FreeFall	0	1	Bed	Head
2	Tail	1	FreeFall	0	2	Bed	Head
3	Head	1	Flip	0	2	Bed	Tail
4	Tail	1	Flip	0	2	Bed	Head
5	Head	1	FreeFallVertical	90	2	Bed	Head
6	Tail	1	FreeFallVertical	90	2	Bed	Head
7	Head	1	FreeFall	0	1	Bed	Head
8	Tail	1	FreeFall	0	2	Bed	Tail
9	Head	1	Flip	0	2	Bed	Head
10	Tail	1	Flip	0	2	Bed	Head
11	Head	1	FreeFallVertical	90	2	Bed	Head
12	Tail	1	FreeFallVertical	90	2	Bed	Head
13	Head	1	FreeFall	0	0	Hand	Head
14	Tail	1	FreeFall	0	0	Hand	Tail
15	Head	1	Flip	0	0	Hand	Tail
16	Tail	1	Flip	0	0	Hand	Tail
17	Head	1	FreeFallVertical	90	1	Hand	Tail
18	Tail	1	FreeFallVertical	90	1	Hand	Tail
19	Head	1	FreeFall	0	0	Hand	Head
20	Tail	1	FreeFall	0	0	Hand	Tail
21	Head	1	Flip	0	0	Hand	Head
22	Tail	1	Flip	0	0	Hand	Head
23	Head	1	FreeFallVertical	90	1	Hand	Head
24	Tail	1	FreeFallVertical	90	1	Hand	Head
25	Head	1	FreeFall	0	3	Floor	Head
26	Tail	1	FreeFall	0	3	Floor	Tail
27	Head	1	Flip	0	3	Floor	Head
28	Tail	1	Flip	0	4	Floor	Tail

Figure 1.1 – Part of the Coin Drop Data Set

2) Preprocessing

Preprocessing is applied to the data to prepare it for the Neural Network. It is made by encoding the “InitialFace”, “DropMethod”, “DropSurface” with one-hot encoding and “FinalFace” with label encoding. After that random oversampling is applied to the data for getting same amount of head and tail. Then putting the data to x and y variables by assigning the “FinalFace” to y and assigning the data to x with dropping the “DropHeight”, “FinalFace” and “DropNumber”. Removing “DropHeight” because it is same for all cases. Then the data is sperated with 27% test and 63% train.



```python
def PreprocessData(data):
    """
    Preprocessing the data by encoding the labels and separating the train and test parts
    return: data sets for train and test
    """
    one_hot = OneHotEncoder()
    features = ['InitialFace', 'DropMethod', 'DropSurface']
    encoded_data = one_hot.fit_transform(data[features])
    encoded_data = encoded_data.toarray()
    encoded_data_columns= one_hot.get_feature_names_out(features)
    encoded_dataframe = pd.DataFrame(encoded_data, columns=encoded_data_columns)

    data = data.join(encoded_dataframe)
    data.drop(features, axis=1, inplace=True)

    label = LabelEncoder()
    y = label.fit_transform(data['FinalFace'])
    x = data.drop(['DropNumber', 'FinalFace', 'DropHeight'], axis=1)

    scale = StandardScaler()
    x[['DropAngle', 'BounceNum']] = scale.fit_transform(x[['DropAngle', 'BounceNum']])
    x.columns = x.columns.astype(str)

    oversampler = RandomOverSampler(random_state=4)
    x, y = oversampler.fit_resample(x, y)

    train_x , test_x, train_y, test_y = train_test_split(x, y, test_size=0.27, stratify=y, random_state=4)

    return train_x , test_x, train_y, test_y
```

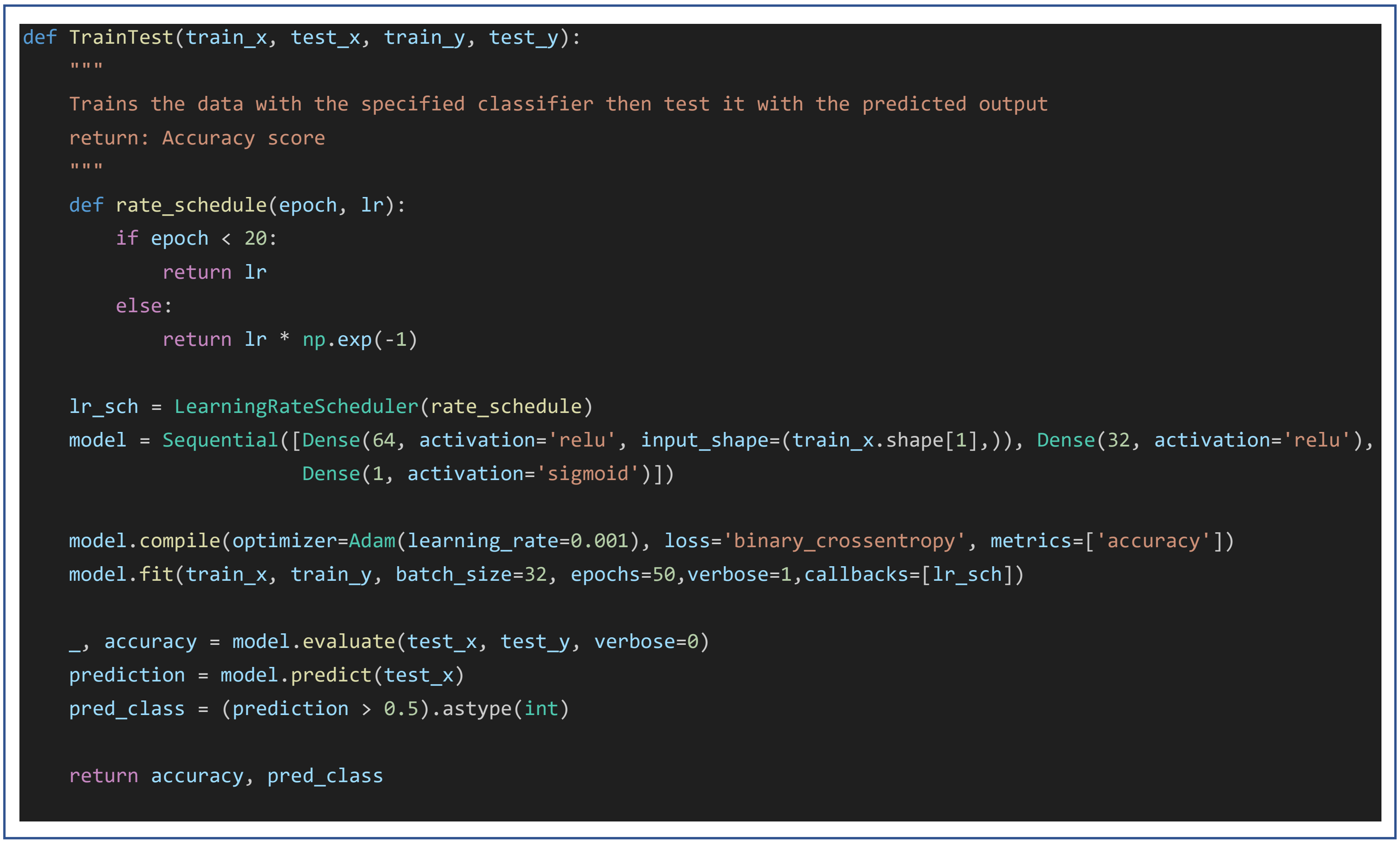
Figure 2.1 – Data Preprocessing Function in Python

3) Neural Network Architecture Selection

Sequential Neural Network architecture with two hidden layers is selected. Both of the layers use Rectified Linear Unit activation. Output layer has one neuron and it uses sigmoid activation. Sigmoid activation is selected for binary classification because its output values are between 0 and 1 showing the probability of one class.
Model uses Adam optimizer and "binary_crossentropy" loss function which is effective for binary classification. It also uses "LearningRateScheduler" function to modify the learning rate in the training process for better results. Model is shown in the Figure 4.1.

4) Evaluation

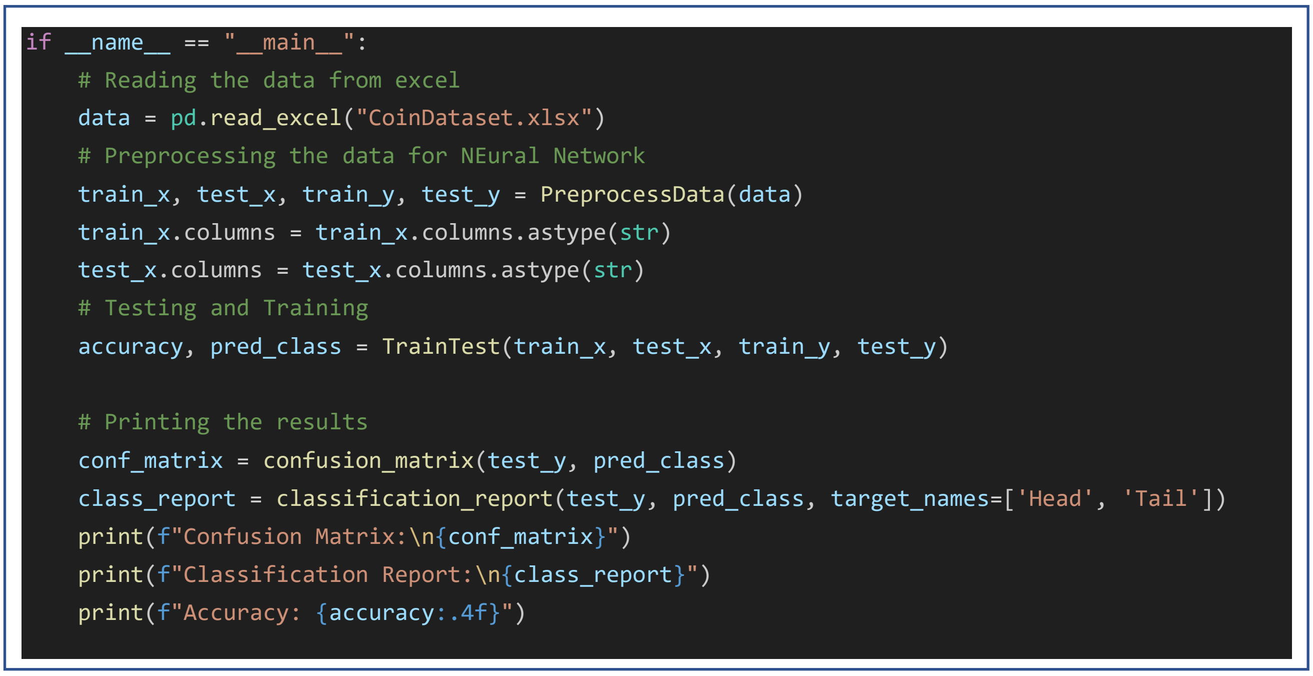
Train and test are performed as in the figure below. Accuracy and prediction values are returned.



```python
def TrainTest(train_x, test_x, train_y, test_y):
    """
    Trains the data with the specified classifier then test it with the predicted output
    return: Accuracy score
    """
    def rate_schedule(epoch, lr):
        if epoch < 20:
            return lr
        else:
            return lr * np.exp(-1)

    lr_sch = LearningRateScheduler(rate_schedule)
    model = Sequential([Dense(64, activation='relu', input_shape=(train_x.shape[1],)), Dense(32, activation='relu'),
                        Dense(1, activation='sigmoid')])

    model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])
    model.fit(train_x, train_y, batch_size=32, epochs=50,verbose=1,callbacks=[lr_sch])

    _, accuracy = model.evaluate(test_x, test_y, verbose=0)
    prediction = model.predict(test_x)
    pred_class = (prediction > 0.5).astype(int)

    return accuracy, pred_class
```

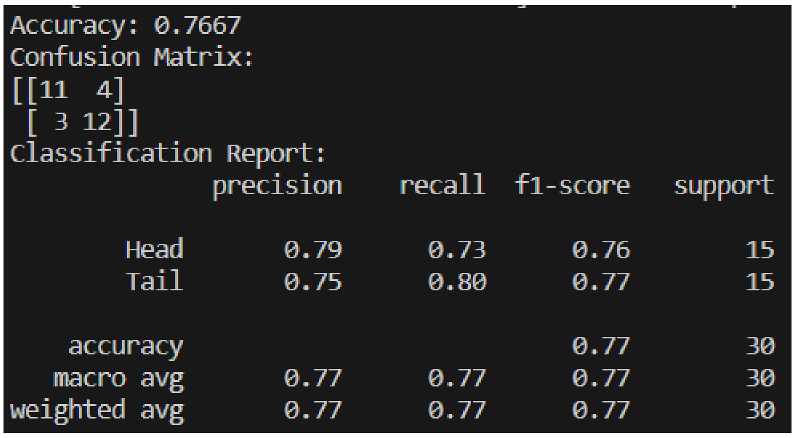
Figure 4.1 – Train and Test of the Obtained Data Set

5) Results



```
if __name__ == "__main__":
    # Reading the data from excel
    data = pd.read_excel("CoinDataset.xlsx")
    # Preprocessing the data for NEural Network
    train_x, test_x, train_y, test_y = PreprocessData(data)
    train_x.columns = train_x.columns.astype(str)
    test_x.columns = test_x.columns.astype(str)
    # Testing and Training
    accuracy, pred_class = TrainTest(train_x, test_x, train_y, test_y)

    # Printing the results
    conf_matrix = confusion_matrix(test_y, pred_class)
    class_report = classification_report(test_y, pred_class, target_names=['Head', 'Tail'])
    print(f"Confusion Matrix:\n{conf_matrix}")
    print(f"Classification Report:\n{class_report}")
    print(f"Accuracy: {accuracy:.4f}")
```

Figure 5.1 – Main Code



```
Accuracy: 0.7667
Confusion Matrix:
[[11  4]
 [ 3 12]]
Classification Report:
              precision    recall  f1-score   support

        Head       0.79      0.73      0.76        15
        Tail       0.75      0.80      0.77        15

    accuracy                           0.77        30
   macro avg       0.77      0.77      0.77        30
weighted avg       0.77      0.77      0.77        30
```

Figure 5.2 - Results

76.67% accuracy is achieved in the prediction. Precision for head is 79%, tail is 75% so the it is predicted correctly 79% of head and 75% of tails. This model is less precise at predicting tails. Support shows that 15 number of test is provided both for heads and tails. Macro average and weighted average are same with 0.77 because we balanced the class distribution with oversampling.

6) Discussion About Challenges:

- Unbalanced class distribution: Unbalanced class distribution causes separated results in the recall. It is solved by applying oversampling to the class.
- Noise reduction: Unnecessary variables in the dataset such as drop number and drop height are removed in the data preprocessing section because they can effect the efficiency of the model.
- Size of the collected coin dataset is not sufficient for the Neural Network's performance. More data should be collected for better performance. Data could be collected with the design of automated and reliable robots for flipping coin. There are larger datasets in the internet provided by University of California which have 40000 coin flip data but I prefer to use my dataset because larger datasets have only one "Initial Face" label.
- More labels should be added to the coin dataset for better accuracy. These labels could be vertical and horizontal angles of the coin, spin number of the coin in the air, and force acted on the coin.

Github link:
https://github.com/dundaremreozbirecikli/ANN-Coin-Drop-Prediction