link text**Packages Installation**

```
pip install numpy pyspark
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.23.5)
Collecting pyspark
  Downloading pyspark-3.5.0.tar.gz (316.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 316.9/316.9 MB 4.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.0-py2.py3-none-any.whl size=317425344 sha256=57b4e72a1b5c61a509b86e1e5116f3b8517768bc2
  Stored in directory: /root/.cache/pip/wheels/41/4e/10/c2cf2467f71c678cfc8a6b9ac9241e5e44a01940da8fbb17fc
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.0
```

Spark session and reading csv file to Spark Dataframe

```
from pyspark.ml.feature import StringIndexer, StandardScaler, VectorAssembler
from pyspark.ml import Pipeline
import numpy as np

# Set seed for reproducibility
np.random.seed(22)

# Initialize Spark session
from pyspark.sql import SparkSession
spark = SparkSession\
        .builder\
        .appName("Award Winner Prediction")\
        .config("spark.some.config.option", "some-value")\
        .getOrCreate()


# Read CSV into PySpark DataFrame
df = spark.read.csv('/content/FinalDS.csv',inferSchema=True, header =True)

df.show()
```

```
+----+--------+--------------------+--------------------+-------------------+------+
|Year|Ceremony|               Award|                Name|               Film|Winner|
+----+--------+--------------------+--------------------+-------------------+------+
|1927|       1|               Actor| Richard Barthelmess|          The Noose|     0|
|1927|       1|               Actor|        Emil Jannings|   The Last Command|     1|
|1927|       1|             Actress|      Louise Dresser|     A Ship Comes In|     0|
|1927|       1|             Actress|        Janet Gaynor|         7th Heaven|     1|
|1927|       1|             Actress|      Gloria Swanson|      Sadie Thompson|     0|
|1927|       1|       Art Direction|       Rochus Gliese|            Sunrise|     0|
|1927|       1|       Art Direction|William Cameron M...|   The Dove Tempest|     1|
|1927|       1|       Art Direction|        Harry Oliver|         7th Heaven|     0|
|1927|       1|      Cinematography|       George Barnes|    The Devil Dancer|     0|
|1927|       1|      Cinematography|      Charles Rosher|            Sunrise|     1|
|1927|       1|      Cinematography|         Karl Struss|            Sunrise|     1|
|1927|       1|Directing Comedy ...|     Lewis Milestone|Two Arabian Knights|     1|
|1927|       1|Directing Comedy ...|          Ted Wilde|             Speedy|  NULL|
|1927|       1|Directing Dramati...|       Frank Borzage|         7th Heaven|     1|
|1927|       1|Directing Dramati...|      Herbert Brenon|     Sorrell and Son|     0|
|1927|       1|Directing Dramati...|          King Vidor|          The Crowd|     0|
|1927|       1| Engineering Effects|      Ralph Hammeras|               NULL|     0|
|1927|       1| Engineering Effects|         Roy Pomeroy|              Wings|     1|
|1927|       1| Engineering Effects|     Nugent Slaughter|               NULL|     0|
|1927|       1| Outstanding Picture|   The Caddo Company|          The Racket|     0|
+----+--------+--------------------+--------------------+-------------------+------+
only showing top 20 rows
```

Features of the Dataset

```
df.printSchema()
```

```
root
 |-- Year: integer (nullable = true)
 |-- Ceremony: integer (nullable = true)
 |-- Award: string (nullable = true)
 |-- Name: string (nullable = true)
 |-- Film: string (nullable = true)
 |-- Winner: integer (nullable = true)
```

Filling missing values of winner column with 0 and Neural Network model training

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler, StandardScaler
from pyspark.ml import Pipeline
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

spark = SparkSession.builder.appName("OscarsPrediction").getOrCreate()

df = spark.read.csv("/content/FinalDS.csv", header=True, inferSchema=True)

# Handle missing values in 'Winner'
df = df.fillna({'Winner': 0})

categorical_cols = ["Award", "Name", "Film"]
numerical_cols = ["Year", "Ceremony"]
target_col = 'Winner'
stages = []

for col_name in categorical_cols:
    indexer = StringIndexer(inputCol=col_name, outputCol=f"{col_name}_index", handleInvalid="keep")
    stages += [indexer]
    encoder = OneHotEncoder(inputCol=f"{col_name}_index", outputCol=f"{col_name}_vec")
    stages += [encoder]

all_feature_cols = [f"{col_name}_vec" for col_name in categorical_cols] + numerical_cols
final_assembler = VectorAssembler(inputCols=all_feature_cols, outputCol="features")
stages += [final_assembler]
scaler = StandardScaler(inputCol="features", outputCol="scaled_features")
stages += [scaler]

feature_pipeline = Pipeline(stages=stages)
feature_model = feature_pipeline.fit(df)
df_transformed = feature_model.transform(df)

df = df.withColumn("Winner", df["Winner"].cast("integer"))
df.select("Winner").distinct().show()


df_transformed.select("features").show(truncate=False)
df = df.withColumn("Winner", df["Winner"].cast("integer"))

df.printSchema()
(train, test) = df_transformed.randomSplit([0.8, 0.2])

# Define the layers of the neural network
num_features = 12226
layers = [num_features, 5, 4, 2]  # 2 nodes in the output layer for binary classification

classifier = MultilayerPerceptronClassifier(layers=layers, blockSize=128, seed=1234, featuresCol='scaled_features', labelCol='Winner')
model = classifier.fit(train)
predictions = model.transform(test)


predictions.select("Winner", "prediction").show()


evaluator = MulticlassClassificationEvaluator(labelCol="Winner", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))
```

```
      |(12226,[11,3874,7250,12224,12225],[1.0,1.0,1.0,1927.0,1.0])  |
|(12226,[11,3874,7250,12224,12225],[1.0,1.0,1.0,1927.0,1.0])  |
|(12226,[11,1329,5998,12224,12225],[1.0,1.0,1.0,1927.0,1.0])  |
|(12226,[11,839,11115,12224,12225],[1.0,1.0,1.0,1927.0,1.0])  |
|(12226,[6,4482,6151,12224,12225],[1.0,1.0,1.0,1927.0,1.0])   |
|(12226,[6,1076,11617,12224,12225],[1.0,1.0,1.0,1927.0,1.0])  |
|(12226,[6,1281,5998,12224,12225],[1.0,1.0,1.0,1927.0,1.0])   |
|(12226,[9,1269,11612,12224,12225],[1.0,1.0,1.0,1927.0,1.0])  |
|(12226,[9,2692,6151,12224,12225],[1.0,1.0,1.0,1927.0,1.0])   |
|(12226,[9,3679,6151,12224,12225],[1.0,1.0,1.0,1927.0,1.0])   |
|(12226,[109,1381,11872,12224,12225],[1.0,1.0,1.0,1927.0,1.0])|
|(12226,[109,4846,11379,12224,12225],[1.0,1.0,1.0,1927.0,1.0])|
|(12226,[101,3149,5998,12224,12225],[1.0,1.0,1.0,1927.0,1.0]) |
|(12226,[101,3365,11368,12224,12225],[1.0,1.0,1.0,1927.0,1.0])|
|(12226,[101,876,7079,12224,12225],[1.0,1.0,1.0,1927.0,1.0])  |
|(12226,[102,4394,12224,12225],[1.0,1.0,1927.0,1.0])          |
|(12226,[102,4504,7194,12224,12225],[1.0,1.0,1.0,1927.0,1.0]) |
|(12226,[102,4173,12224,12225],[1.0,1.0,1927.0,1.0])          |
|(12226,[92,4951,11716,12224,12225],[1.0,1.0,1.0,1927.0,1.0]) |
+-------------------------------------------------------------+
only showing top 20 rows

root
 |-- Year: integer (nullable = true)
 |-- Ceremony: integer (nullable = true)
 |-- Award: string (nullable = true)
 |-- Name: string (nullable = true)
 |-- Film: string (nullable = true)
 |-- Winner: integer (nullable = false)

+------+----------+
|Winner|prediction|
+------+----------+
|     1|       0.0|
|     0|       0.0|
|     0|       1.0|
|     0|       1.0|
|     0|       1.0|
|     0|       0.0|
|     1|       1.0|
|     0|       0.0|
|     0|       0.0|
|     0|       1.0|
|     0|       0.0|
|     0|       0.0|
|     1|       0.0|
|     1|       0.0|
|     0|       0.0|
|     0|       0.0|
|     0|       0.0|
|     1|       0.0|
|     0|       0.0|
|     0|       0.0|
+------+----------+
only showing top 20 rows

Test set accuracy = 0.7673716012084593
```

Filling Missing values by filtering , Standardization and NN model training -MultilayerPerceptronClassifier

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler, StandardScaler
from pyspark.ml import Pipeline
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

spark = SparkSession.builder.appName("OscarsPrediction").getOrCreate()

df = spark.read.csv("/content/FinalDS.csv", header=True, inferSchema=True)

# Handle missing values in 'Winner'
df = df.filter(df.Winner.isNotNull())

categorical_cols = ["Award", "Name", "Film"]
numerical_cols = ["Year", "Ceremony"]
target_col = 'Winner'
stages = []

for col_name in categorical_cols:
    indexer = StringIndexer(inputCol=col_name, outputCol=f"{col_name}_index", handleInvalid="keep")
    stages += [indexer]
    encoder = OneHotEncoder(inputCol=f"{col_name}_index", outputCol=f"{col_name}_vec")
```

```
        stages += [encoder]

    all_feature_cols = [f"{col_name}_vec" for col_name in categorical_cols] + numerical_cols
    final_assembler = VectorAssembler(inputCols=all_feature_cols, outputCol="features")
    stages += [final_assembler]
    scaler = StandardScaler(inputCol="features", outputCol="scaled_features")
    stages += [scaler]

    feature_pipeline = Pipeline(stages=stages)
    feature_model = feature_pipeline.fit(df)
    df_transformed = feature_model.transform(df)

    df = df.withColumn("Winner", df["Winner"].cast("integer"))
    df.select("Winner").distinct().show()


    df_transformed.select("features").show(truncate=False)
    df = df.withColumn("Winner", df["Winner"].cast("integer"))

    df.printSchema()
    (train, test) = df_transformed.randomSplit([0.7, 0.3])

    # Define the layers of the neural network
    num_features = 4421
    layers = [num_features, 5, 4, 2]  # 2 nodes in the output layer for binary classification

    classifier = MultilayerPerceptronClassifier(layers=layers, blockSize=128, seed=1234, featuresCol='scaled_features', labelCol='Winner')
    model = classifier.fit(train)
    predictions = model.transform(test)


    predictions.select("Winner", "prediction").show()


    evaluator = MulticlassClassificationEvaluator(labelCol="Winner", predictionCol="prediction", metricName="accuracy")
    accuracy = evaluator.evaluate(predictions)
    print("Test set accuracy = " + str(accuracy))
```

```
|(4421,[14,1724,4210,4419,4420],[1.0,1.0,1.0,1927.0,1.0])|
|(4421,[14,1081,4201,4419,4420],[1.0,1.0,1.0,1927.0,1.0])|
|(4421,[12,1496,2622,4419,4420],[1.0,1.0,1.0,1927.0,1.0])|
|(4421,[12,1337,2316,4419,4420],[1.0,1.0,1.0,1927.0,1.0])|
|(4421,[12,1185,3997,4419,4420],[1.0,1.0,1.0,1927.0,1.0])|
|(4421,[3,1748,2338,4419,4420],[1.0,1.0,1.0,1927.0,1.0]) |
|(4421,[3,297,4169,4419,4420],[1.0,1.0,1.0,1927.0,1.0])  |
|(4421,[3,382,2316,4419,4420],[1.0,1.0,1.0,1927.0,1.0])  |
|(4421,[8,371,4165,4419,4420],[1.0,1.0,1.0,1927.0,1.0])  |
|(4421,[8,926,2338,4419,4420],[1.0,1.0,1.0,1927.0,1.0])  |
|(4421,[8,1412,2338,4419,4420],[1.0,1.0,1.0,1927.0,1.0]) |
|(4421,[103,415,4281,4419,4420],[1.0,1.0,1.0,1927.0,1.0])|
|(4421,[86,1133,2316,4419,4420],[1.0,1.0,1.0,1927.0,1.0])|
|(4421,[86,1248,4074,4419,4420],[1.0,1.0,1.0,1927.0,1.0])|
|(4421,[86,411,2579,4419,4420],[1.0,1.0,1.0,1927.0,1.0]) |
|(4421,[87,1701,4419,4420],[1.0,1.0,1927.0,1.0])         |
|(4421,[87,1755,2598,4419,4420],[1.0,1.0,1.0,1927.0,1.0])|
|(4421,[87,1622,4419,4420],[1.0,1.0,1927.0,1.0])         |
|(4421,[74,1936,4217,4419,4420],[1.0,1.0,1.0,1927.0,1.0])|
|(4421,[74,362,2316,4419,4420],[1.0,1.0,1.0,1927.0,1.0]) |
+--------------------------------------------------------+
only showing top 20 rows

root
 |-- Year: integer (nullable = true)
 |-- Ceremony: integer (nullable = true)
 |-- Award: string (nullable = true)
 |-- Name: string (nullable = true)
 |-- Film: string (nullable = true)
 |-- Winner: integer (nullable = true)

+------+----------+
```

```
|     0|       1.0|
|     0|       1.0|
|     0|       1.0|
|     0|       1.0|
|     1|       1.0|
|     0|       1.0|
|     0|       1.0|
|     0|       1.0|
|     1|       0.0|
|     1|       0.0|
|     1|       1.0|
|     1|       1.0|
|     0|       0.0|
+------+----------+
only showing top 20 rows

Test set accuracy = 0.8068965517241379
```

PRECISION, RECALL, F1-SCORE

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Assuming 'predictions' is your DataFrame with actual and predicted labels
evaluatorPrecision = MulticlassClassificationEvaluator(labelCol="Winner", predictionCol="prediction", metricName="weightedPrecision")
evaluatorRecall = MulticlassClassificationEvaluator(labelCol="Winner", predictionCol="prediction", metricName="weightedRecall")
evaluatorF1 = MulticlassClassificationEvaluator(labelCol="Winner", predictionCol="prediction", metricName="f1")

precision = evaluatorPrecision.evaluate(predictions)
recall = evaluatorRecall.evaluate(predictions)
f1_score = evaluatorF1.evaluate(predictions)

print(f"Weighted Precision: {precision}")
print(f"Weighted Recall: {recall}")
print(f"F1 Score: {f1_score}")
```

```
Weighted Precision: 0.7961817913136372
Weighted Recall: 0.806896551724138
F1 Score: 0.759871282908559
```

HYPER TUNING PARAMETERS

```
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

# Define a new classifier
classifier = MultilayerPerceptronClassifier(featuresCol='scaled_features', labelCol='Winner')

# Create the parameter grid
paramGrid = (ParamGridBuilder()
             .addGrid(classifier.maxIter, [100, 200])  # Max Iteration
             .addGrid(classifier.layers, [[4421, 5, 4, 2], [4421, 10, 5, 2]])  # Different layer structures
             .build())

# Create cross-validator
cv = CrossValidator(estimator=classifier, estimatorParamMaps=paramGrid, evaluator=MulticlassClassificationEvaluator(labelCol="Winner"), numF
cvModel = cv.fit(train)

# Use the best model to make predictions
cvPredictions = cvModel.transform(test)

# Evaluate best model
cvAccuracy = evaluator.evaluate(cvPredictions)
print("Best Model Test set accuracy = " + str(cvAccuracy))
```

```
Best Model Test set accuracy = 0.817762399077278
```

```
# Trying a different architecture
newLayers = [4421, 10, 5, 2]  # More nodes in the first hidden layer
newClassifier = MultilayerPerceptronClassifier(layers=newLayers, blockSize=128, seed=1234, featuresCol='scaled_features', labelCol='Winner')

# Train and evaluate this new model
newModel = newClassifier.fit(train)
newPredictions = newModel.transform(test)
newAccuracy = evaluator.evaluate(newPredictions)
print("New Model Test set accuracy = " + str(newAccuracy))
```

```
New Model Test set accuracy = 0.790080738177624
```

```
pandas_df = df.toPandas()
```

*Pandas dataframe conversion and training the model using keras *

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
for column in categorical_cols:
    pandas_df[column] = label_encoder.fit_transform(pandas_df[column])
```

```
X = pandas_df.drop('Winner', axis=1)
y = pandas_df['Winner']

X = X.values.astype('float32')
y = y.values.astype('float32')

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=22)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Example for a binary classification problem
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

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

```
history = model.fit(X_train, y_train, epochs=15, validation_data=(X_test, y_test))
```

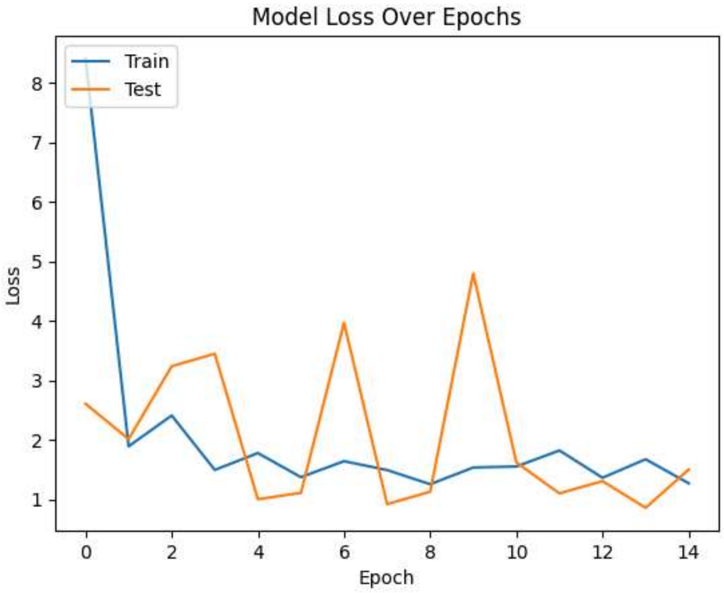
```
Epoch 1/15
76/76 [==============================] - 1s 6ms/step - loss: 7.2844 - accuracy: 0.6295 - val_loss: 1.8829 - val_accuracy: 0.6458
Epoch 2/15
76/76 [==============================] - 0s 4ms/step - loss: 2.5556 - accuracy: 0.6506 - val_loss: 1.7933 - val_accuracy: 0.7529
Epoch 3/15
76/76 [==============================] - 0s 5ms/step - loss: 1.7863 - accuracy: 0.6663 - val_loss: 3.7326 - val_accuracy: 0.7792
Epoch 4/15
76/76 [==============================] - 0s 6ms/step - loss: 1.4434 - accuracy: 0.6964 - val_loss: 1.7349 - val_accuracy: 0.7792
Epoch 5/15
76/76 [==============================] - 0s 6ms/step - loss: 1.1749 - accuracy: 0.6914 - val_loss: 1.5319 - val_accuracy: 0.7792
Epoch 6/15
76/76 [==============================] - 0s 6ms/step - loss: 1.1718 - accuracy: 0.7054 - val_loss: 0.9005 - val_accuracy: 0.7776
Epoch 7/15
76/76 [==============================] - 1s 7ms/step - loss: 1.0533 - accuracy: 0.6988 - val_loss: 0.6790 - val_accuracy: 0.7051
Epoch 8/15
76/76 [==============================] - 1s 7ms/step - loss: 1.4630 - accuracy: 0.6898 - val_loss: 0.5774 - val_accuracy: 0.7743
Epoch 9/15
76/76 [==============================] - 0s 6ms/step - loss: 0.9140 - accuracy: 0.7199 - val_loss: 0.8404 - val_accuracy: 0.7825
Epoch 10/15
76/76 [==============================] - 0s 6ms/step - loss: 1.1244 - accuracy: 0.7137 - val_loss: 1.4351 - val_accuracy: 0.7792
Epoch 11/15
76/76 [==============================] - 0s 6ms/step - loss: 0.9503 - accuracy: 0.7034 - val_loss: 0.6949 - val_accuracy: 0.7315
```

```
Epoch 12/15
76/76 [==============================] - 0s 6ms/step - loss: 1.0915 - accuracy: 0.7174 - val_loss: 0.5325 - val_accuracy: 0.7727
Epoch 13/15
76/76 [==============================] - 0s 5ms/step - loss: 0.9840 - accuracy: 0.7104 - val_loss: 0.5198 - val_accuracy: 0.7908
Epoch 14/15
76/76 [==============================] - 0s 4ms/step - loss: 1.0435 - accuracy: 0.7034 - val_loss: 0.7803 - val_accuracy: 0.7727
Epoch 15/15
76/76 [==============================] - 0s 4ms/step - loss: 1.4027 - accuracy: 0.7013 - val_loss: 0.7811 - val_accuracy: 0.7035
```

Plotting Model Loss Over Epochs

> Indented block

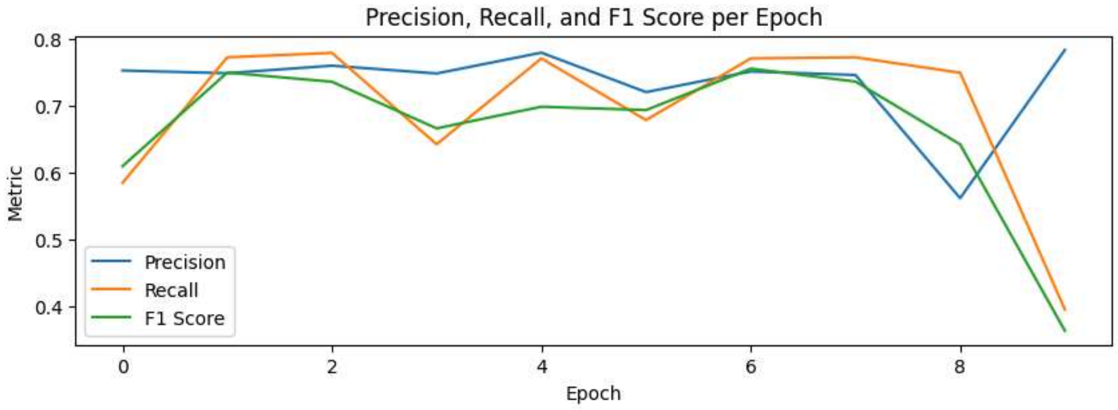
```
import matplotlib.pyplot as plt

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss Over Epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



```
import tensorflow as tf
from sklearn.metrics import precision_score, recall_score, f1_score

class MetricsCallback(tf.keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.precision = []
        self.recall = []
        self.f1 = []

    def on_epoch_end(self, epoch, logs={}):
        y_pred = (self.model.predict(X_test) > 0.5).astype("int32")
        precision = precision_score(y_test, y_pred, average='weighted')
        recall = recall_score(y_test, y_pred, average='weighted')
        f1 = f1_score(y_test, y_pred, average='weighted')
        self.precision.append(precision)
        self.recall.append(recall)
        self.f1.append(f1)
        print(f' — val_precision: {precision} — val_recall: {recall} — val_f1: {f1}')


metrics_callback = MetricsCallback()

history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test), callbacks=[metrics_callback])
```

```
Epoch 1/10
19/19 [==============================] - 0s 2ms/step
 — val_precision: 0.7528926458974884 — val_recall: 0.5848434925864909 — val_f1: 0.6096879740257395
76/76 [==============================] - 1s 7ms/step - loss: 0.7831 - accuracy: 0.7463 - val_loss: 1.0829 - val_accuracy: 0.5848
Epoch 2/10
19/19 [==============================] - 0s 2ms/step
 — val_precision: 0.7488395876564735 — val_recall: 0.7726523887973641 — val_f1: 0.7498087000057508
76/76 [==============================] - 0s 6ms/step - loss: 1.0079 - accuracy: 0.7318 - val_loss: 0.7497 - val_accuracy: 0.7727
Epoch 3/10
19/19 [==============================] - 0s 2ms/step
 — val_precision: 0.7601133004757388 — val_recall: 0.7792421746293245 — val_f1: 0.7360634697058233
76/76 [==============================] - 0s 6ms/step - loss: 1.1398 - accuracy: 0.7298 - val_loss: 1.1190 - val_accuracy: 0.7792
Epoch 4/10
19/19 [==============================] - 0s 2ms/step
 — val_precision: 0.7484659647112193 — val_recall: 0.642504118616145 — val_f1: 0.6663091714305811
76/76 [==============================] - 0s 6ms/step - loss: 1.0307 - accuracy: 0.7269 - val_loss: 0.9515 - val_accuracy: 0.6425
Epoch 5/10
19/19 [==============================] - 0s 2ms/step
 — val_precision: 0.7796158153788979 — val_recall: 0.771004942339374 — val_f1: 0.698712932054481
76/76 [==============================] - 0s 6ms/step - loss: 0.8780 - accuracy: 0.7335 - val_loss: 1.0629 - val_accuracy: 0.7710
Epoch 6/10
19/19 [==============================] - 0s 2ms/step
 — val_precision: 0.7205610135254021 — val_recall: 0.6787479406919276 — val_f1: 0.6937189669867062
76/76 [==============================] - 0s 6ms/step - loss: 0.9698 - accuracy: 0.7232 - val_loss: 0.8092 - val_accuracy: 0.6787
Epoch 7/10
19/19 [==============================] - 0s 2ms/step
 — val_precision: 0.7514278338126275 — val_recall: 0.771004942339374 — val_f1: 0.755688886171155
76/76 [==============================] - 0s 6ms/step - loss: 0.9065 - accuracy: 0.7298 - val_loss: 0.6509 - val_accuracy: 0.7710
Epoch 8/10
19/19 [==============================] - 0s 2ms/step
 — val_precision: 0.7461912281819534 — val_recall: 0.7726523887973641 — val_f1: 0.7363641799432656
76/76 [==============================] - 0s 6ms/step - loss: 0.8434 - accuracy: 0.7393 - val_loss: 0.7207 - val_accuracy: 0.7727
Epoch 9/10
19/19 [==============================] - 0s 2ms/step
 — val_precision: 0.5618823772082432 — val_recall: 0.7495881383855024 — val_f1: 0.6423024537954871
76/76 [==============================] - 1s 7ms/step - loss: 1.3274 - accuracy: 0.7050 - val_loss: 1.8231 - val_accuracy: 0.7496
Epoch 10/10
33/76 [===========>..................] - ETA: 0s - loss: 0.8116 - accuracy: 0.7386/usr/local/lib/python3.10/dist-packages/sklearn/metric
  _warn_prf(average, modifier, msg_start, len(result))
19/19 [==============================] - 0s 3ms/step
 — val_precision: 0.783663184819634 — val_recall: 0.3953871499176277 — val_f1: 0.36350021625120643
76/76 [==============================] - 1s 9ms/step - loss: 0.9666 - accuracy: 0.7306 - val_loss: 2.3848 - val_accuracy: 0.3954
```

```python
import matplotlib.pyplot as plt

# Plot precision, recall, and F1 score
plt.figure(figsize=(10, 3))
plt.plot(metrics_callback.precision, label='Precision')
plt.plot(metrics_callback.recall, label='Recall')
plt.plot(metrics_callback.f1, label='F1 Score')
plt.title('Precision, Recall, and F1 Score per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Metric')
plt.legend()
plt.show()
```



Start coding or generate with AI.