EX.NO:

DATE:

A MULTILAYER PERCEPTRON WITH A HYPERPARAMETER TUNING

AIM:

To implement a Multilayer Perceptron (MLP) model and perform hyperparameter tuning to improve its performance on a given dataset.

ALGORITHM:

STEP1 : Import all necessary libraries including pandas, NumPy, scikit-learn modules, and TensorFlow Keras components for building and evaluating the neural network model.

STEP2 : Load the dataset using pandas and read it into a DataFrame from the specified file path.

STEP3 : Create a binary target column named 'pass' by checking if the GPA is greater than or equal to 2.0.

STEP4 : Engineer a new feature called study_attendance by dividing weekly study time to reflect effort adjusted for attendance.

STEP5 : Remove irrelevant or redundant columns such as StudentID, GPA, and GradeClass to avoid data leakage and unnecessary noise.

STEP6 : Apply one-hot encoding to convert categorical variables into numeric format using dummy variables while dropping the first category to avoid multicollinearity.

STEP7 : Separate the input features (X) and the target variable (y) for modeling.

STEP8 : Normalize the feature set using StandardScaler to ensure that all features contribute equally to the training process.

STEP9 : Split the data into training and testing sets using an 80-20 ratio and define class weights to address any class imbalance.

STEP10 : Build and compile a Sequential MLP model with multiple hidden layers and dropout, train it with early stopping, and evaluate the model's accuracy and classification report on the test set.

SOURCE CODE:

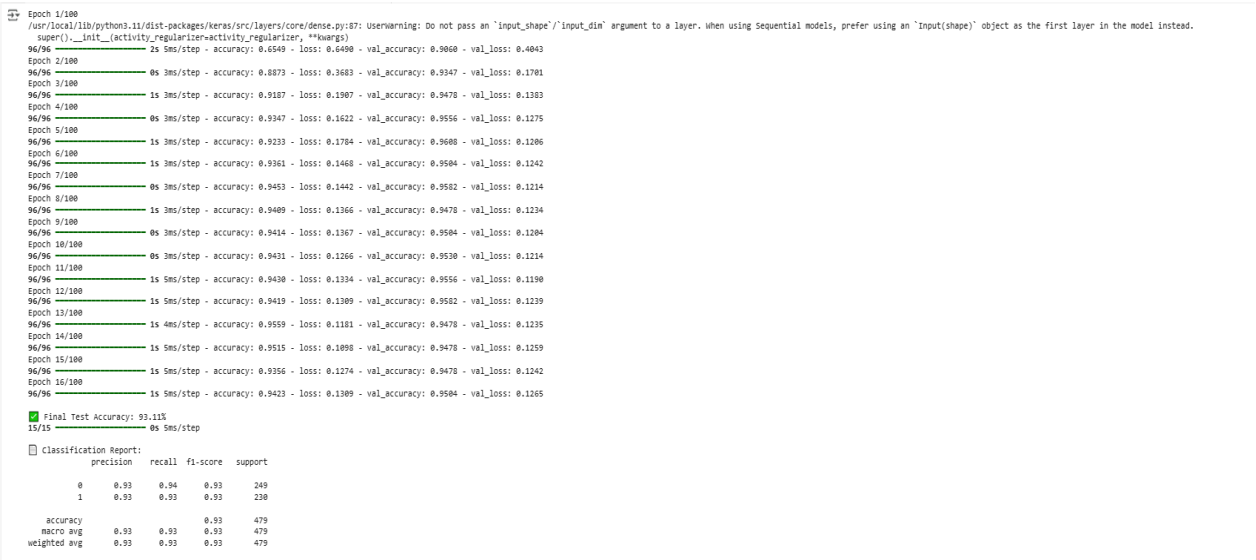
```
#  Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# 2. Load dataset
df = pd.read_csv('/content/drive/MyDrive/Student_performance_data _.csv')

#  3. Create binary target: pass (1 if GPA >= 2.0)
df['pass'] = (df['GPA'] >= 2.0).astype(int)

#  4. Feature engineering
df['study_attendance'] = df['StudyTimeWeekly'] / (1 + df['Absences'])

# 5. Drop unnecessary columns
df = df.drop(['StudentID', 'GPA', 'GradeClass'], axis=1)

#  6. One-hot encoding for categorical features
df = pd.get_dummies(df, drop_first=True)

#  7. Split features and target
X = df.drop('pass', axis=1)
y = df['pass']

# 8. Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 9. Train-test split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

#  1. Import Libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

#  2. Load Dataset
df = pd.read_csv('/content/drive/MyDrive/Student_performance_data _.csv')

#  3. Create Binary Target: Pass if GPA >= 2.0
df['pass'] = (df['GPA'] >= 2.0).astype(int)

#  4. Feature Engineering
df['study_attendance'] = df['StudyTimeWeekly'] / (1 + df['Absences'])

#  5. Drop Unnecessary Columns
df = df.drop(['StudentID', 'GPA', 'GradeClass'], axis=1)

#  6. One-Hot Encoding for Categorical Features
df = pd.get_dummies(df, drop_first=True)

# 7. Split Features and Target
X = df.drop('pass', axis=1)
y = df['pass']
```

```
# 8. Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 9. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

#  10. Build MLP Model with 3 Hidden Layers
model = Sequential([
    Dense(128, activation='relu', input_shape=(X.shape[1],)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),  # Additional Hidden Layer
    Dense(1, activation='sigmoid')  # Binary output
])

#  Compile Model
model.compile(optimizer=Adam(learning_rate=0.0005),
              loss='binary_crossentropy',
              metrics=['accuracy'])

#  11. EarlyStopping to Prevent Overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# 12. Train the Model
history = model.fit(X_train, y_train,
                    validation_split=0.2,
                    epochs=100,
                    batch_size=16,
```

```
        callbacks=[early_stop],
        verbose=1)

#  13. Evaluate Final Test Accuracy
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print(f"\n✅ Final Test Accuracy: {test_acc * 100:.2f}%")

# 14. Classification Report
y_pred = (model.predict(X_test) > 0.5).astype("int32")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

OUTPUT



```
Epoch 1/100
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
96/96 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.6549 - loss: 0.6490 - val_accuracy: 0.9060 - val_loss: 0.4043
Epoch 2/100
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8873 - loss: 0.3683 - val_accuracy: 0.9347 - val_loss: 0.1701
Epoch 3/100
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9187 - loss: 0.1907 - val_accuracy: 0.9478 - val_loss: 0.1383
Epoch 4/100
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9347 - loss: 0.1622 - val_accuracy: 0.9556 - val_loss: 0.1275
Epoch 5/100
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9233 - loss: 0.1784 - val_accuracy: 0.9608 - val_loss: 0.1206
Epoch 6/100
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9361 - loss: 0.1468 - val_accuracy: 0.9504 - val_loss: 0.1242
Epoch 7/100
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9453 - loss: 0.1442 - val_accuracy: 0.9582 - val_loss: 0.1214
Epoch 8/100
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9409 - loss: 0.1366 - val_accuracy: 0.9478 - val_loss: 0.1234
Epoch 9/100
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9414 - loss: 0.1367 - val_accuracy: 0.9504 - val_loss: 0.1204
Epoch 10/100
96/96 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9431 - loss: 0.1266 - val_accuracy: 0.9530 - val_loss: 0.1214
Epoch 11/100
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9430 - loss: 0.1334 - val_accuracy: 0.9556 - val_loss: 0.1190
Epoch 12/100
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9419 - loss: 0.1309 - val_accuracy: 0.9582 - val_loss: 0.1239
Epoch 13/100
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9559 - loss: 0.1181 - val_accuracy: 0.9478 - val_loss: 0.1235
Epoch 14/100
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9515 - loss: 0.1098 - val_accuracy: 0.9478 - val_loss: 0.1259
Epoch 15/100
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9356 - loss: 0.1274 - val_accuracy: 0.9478 - val_loss: 0.1242
Epoch 16/100
96/96 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9423 - loss: 0.1309 - val_accuracy: 0.9504 - val_loss: 0.1265

✅ Final Test Accuracy: 93.11%
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step

📄 Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       249
           1       0.93      0.93      0.93       230

    accuracy                           0.93       479
   macro avg       0.93      0.93      0.93       479
weighted avg       0.93      0.93      0.93       479
```

```
# 1. Setup
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# 2. Load Data
df = pd.read_csv('/content/drive/MyDrive/Student_performance_data _.csv')

# 3. Create binary target: pass if GPA >= 2.0
df['pass'] = (df['GPA'] >= 2.0).astype(int)

# 4. Feature Engineering
df['study_attendance'] = df['StudyTimeWeekly'] / (1 + df['Absences'])

# 5. Drop Unused Columns
df.drop(['StudentID', 'GPA', 'GradeClass'], axis=1, inplace=True)

# 6. One-hot Encoding
df = pd.get_dummies(df, drop_first=True)

# 7. Define Features & Target
X = df.drop('pass', axis=1)
y = df['pass']

# 8. Normalize Features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 9. Train/Test Split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# 10. Handle Imbalance (adjust if needed based on your class distribution)
class_weights = {0: 1.2, 1: 0.8}

# 11. Build Model
model = Sequential([
    Dense(256, activation='relu', input_shape=(X.shape[1],)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

# 12. Compile Model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
              loss='binary_crossentropy',
              metrics=['accuracy'])

# 13. EarlyStopping
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# 14. Train Model
history = model.fit(X_train, y_train,
                    epochs=100,
                    batch_size=32,
                    validation_split=0.2,
                    class_weight=class_weights,
                    callbacks=[early_stop],
                    verbose=1)
```

15. Evaluate Accuracy

loss, acc = model.evaluate(X_test, y_test, verbose=0)

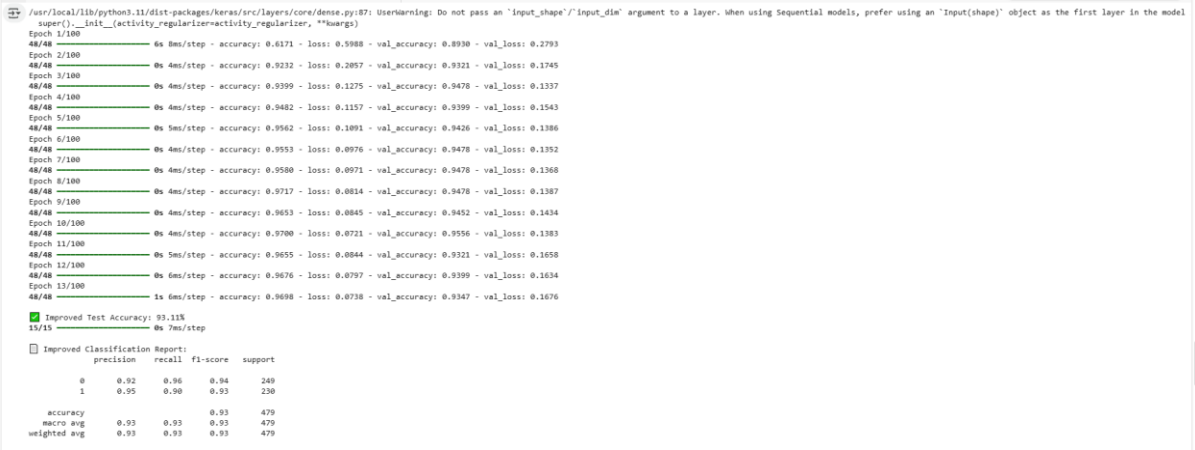
print(f"\n✅ Improved Test Accuracy: {acc * 100:.2f}%")

16. Classification Report

y_pred = (model.predict(X_test) > 0.5).astype(int)

print("\nImproved Classification Report:")

print(classification_report(y_test, y_pred))



from sklearn.metrics import confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

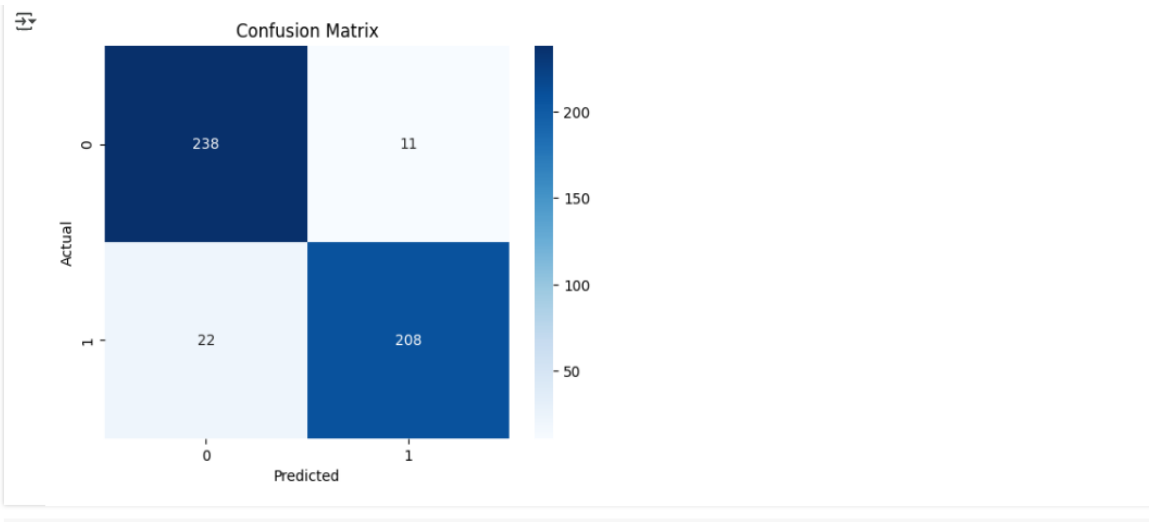
plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

OUTPUT



COE(20)	
RECORD(20)	
VIVA(10)	
TOTAL(50)	

RESULT:

The Multilayer Perceptron model achieved high accuracy in classifying student performance based on the provided features.