EX NO : 02

DATE :

A MULTILAYER PERCEPTRON WITH A HYPERPARAMETER TUNING

AIM:

To develop a Multilayer Perceptron (MLP) model with hyperparameter tuning using Keras Tuner for predicting high spending behavior based on demographic and lifestyle features from the given dataset.

ALGORITHM:

Step 1: Import necessary libraries.

Step 2: Load the dataset into Python.

Step 3: Create a binary target column based on Spending_Score.

Step 4: Remove the ID and Spending_Score columns.

Step 5: Handle missing values in the dataset.

Step 6: Encode all categorical columns.

Step 7: Normalize the input features.

Step 8: Split the dataset into training and testing sets.

Step 9: Write a function to build the MLP model.

Step 10: Use Keras Tuner to tune the model's hyperparameters.

Step 11: Get the best model from the tuner.

Step 12: Train and evaluate the best model on the test data.

CODING:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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

```python
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import classification_report

# Step 1: Load and preprocess your dataset
df = pd.read_csv("/content/drive/MyDrive/DL/Test.csv")

df['target'] = (df['Spending_Score'] == 'High').astype(int)
df.drop(columns=['ID', 'Spending_Score'], inplace=True)

# Fill missing values
for col in df.columns:
    if df[col].dtype == 'object':
        df[col].fillna(df[col].mode()[0], inplace=True)
    else:
        df[col].fillna(df[col].mean(), inplace=True)

# Encode categorical variables
cat_cols = df.select_dtypes(include='object').columns
for col in cat_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])

# Split features and target
X = df.drop(columns='target')
y = df['target']

# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data
```

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

# Step 2: Define different learning rates and epochs
learning_rates = [0.001, 0.0005, 0.0001]
epoch_list = [20, 30, 50]

results = []

# Step 3: Loop through different settings
for lr in learning_rates:
    for epochs in epoch_list:
        print(f"\n    Training with LR={lr}, Epochs={epochs}")

        # Build model
        model = Sequential([
            Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
            Dense(16, activation='relu'),
            Dense(1, activation='sigmoid')
        ])

        optimizer = Adam(learning_rate=lr)
        model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])

        # Train
        history = model.fit(
            X_train, y_train,
            validation_split=0.1,
            epochs=epochs,
            batch_size=8,
            verbose=0
        )
```

```
    # Evaluate on test data
    test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)

    # Save metrics
    results.append({
        'Learning Rate': lr,
        'Epochs': epochs,
        'Train Accuracy': history.history['accuracy'][-1] * 100,
        'Validation Accuracy': history.history['val_accuracy'][-1] * 100,
        'Test Accuracy': test_acc * 100,
        'Train Loss': history.history['loss'][-1],
        'Validation Loss': history.history['val_loss'][-1]
    })
```

```
# Step 4: Display table
df_results = pd.DataFrame(results)
print("\n    Comparison Table:")
display(df_results)
```

```
# Step 5: Plot Accuracy and Loss
plt.figure(figsize=(14, 6))
sns.set_style("whitegrid")
```

```
# Accuracy plot
plt.subplot(1, 2, 1)
for lr in learning_rates:
    subset = df_results[df_results['Learning Rate'] == lr]
    plt.plot(subset['Epochs'], subset['Train Accuracy'], marker='o', label=f'Train Acc (LR={lr})')
    plt.plot(subset['Epochs'], subset['Validation Accuracy'], marker='o', linestyle='--', label=f'Val Acc (LR={lr})')
    plt.plot(subset['Epochs'], subset['Test Accuracy'], marker='x', linestyle=':', label=f'Test Acc (LR={lr})')
```

```
plt.xlabel("Epochs")
plt.ylabel("Accuracy (%)")
plt.title("Accuracy Comparison")
plt.legend()
```

```
# Loss plot
plt.subplot(1, 2, 2)
for lr in learning_rates:
    subset = df_results[df_results['Learning Rate'] == lr]
    plt.plot(subset['Epochs'], subset['Train Loss'], marker='o', label=f'Train Loss (LR={lr})')
    plt.plot(subset['Epochs'], subset['Validation Loss'], marker='o', linestyle='--', label=f'Val Loss (LR={lr})')
```

```
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss Comparison")
plt.legend()
```

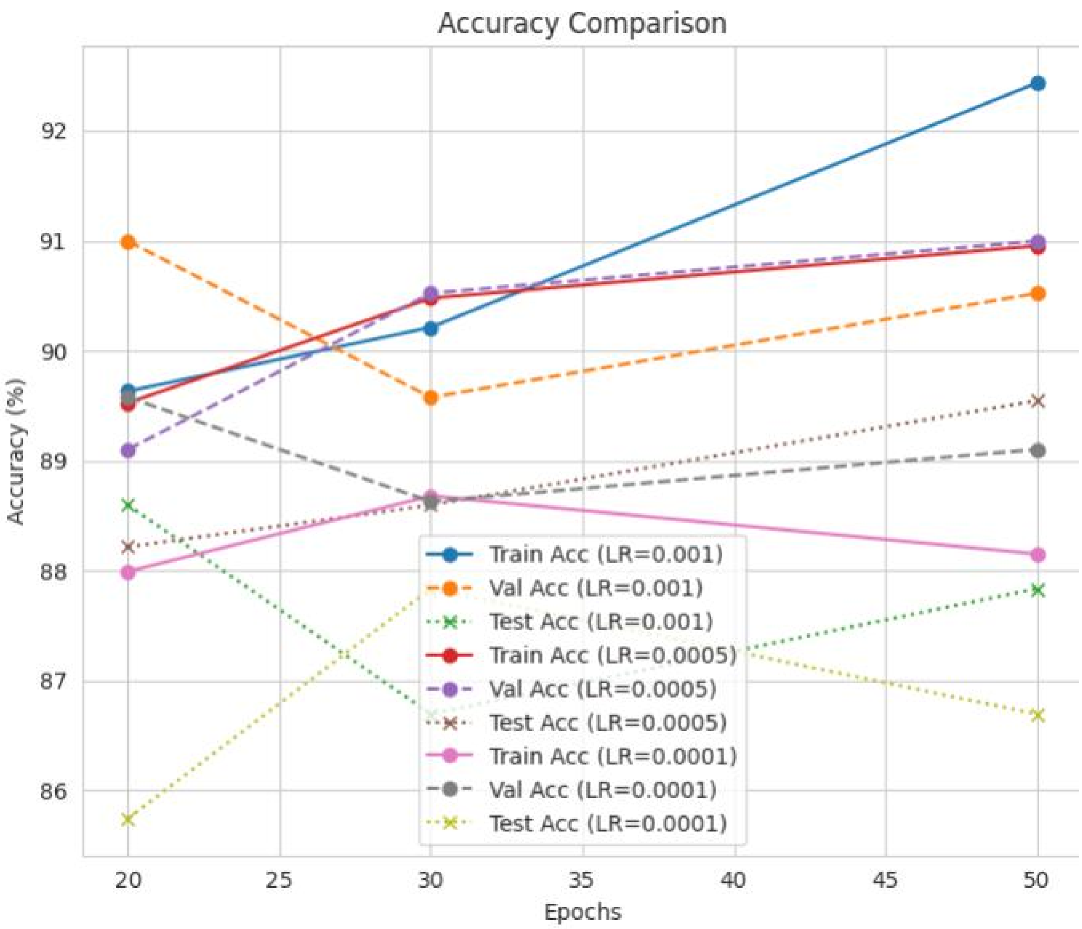
```
plt.tight_layout()
plt.show()
```

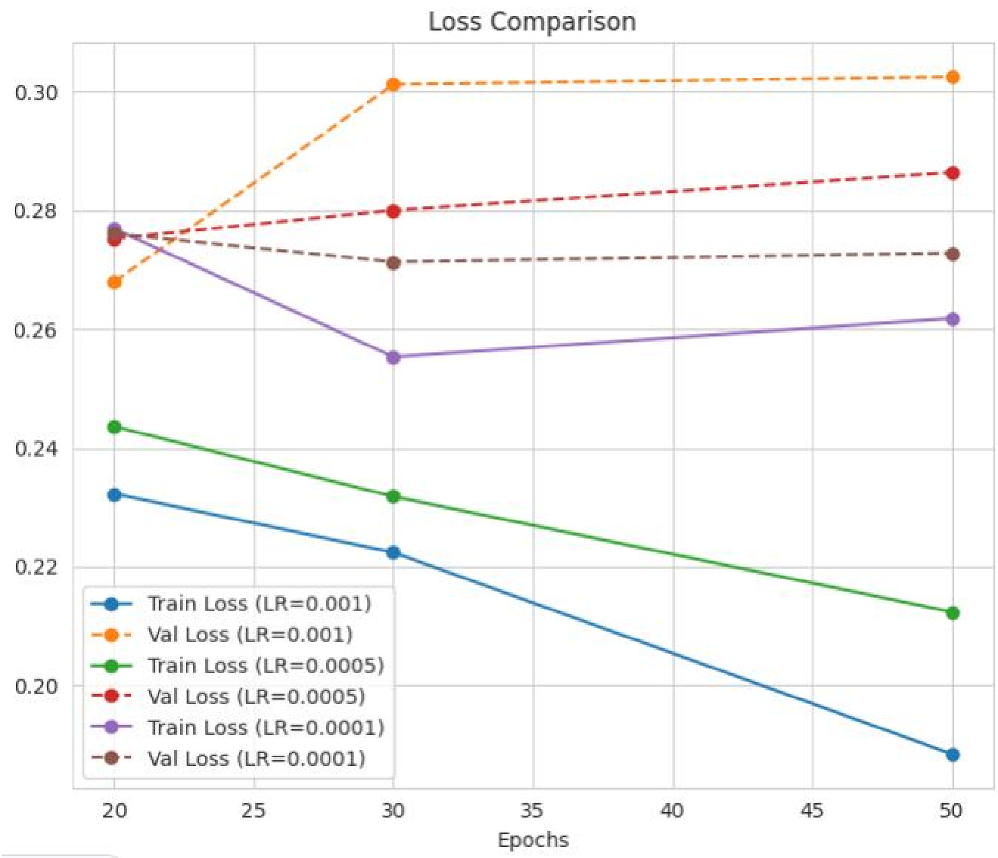
OUTPUT:

```
Training with LR=0.001, Epochs=20
Training with LR=0.001, Epochs=30
Training with LR=0.001, Epochs=50
Training with LR=0.0005, Epochs=20
Training with LR=0.0005, Epochs=30
Training with LR=0.0005, Epochs=50
Training with LR=0.0001, Epochs=20
Training with LR=0.0001, Epochs=30
Training with LR=0.0001, Epochs=50
```

📊 Comparison Table:

	Learning Rate	Epochs	Train Accuracy	Validation Accuracy	Test Accuracy	Train Loss	Validation Loss
0	0.0010	20	89.629632	90.995258	88.593155	0.232290	0.267986
1	0.0010	30	90.211642	89.573461	86.692017	0.222391	0.301207
2	0.0010	50	92.433864	90.521330	87.832701	0.188469	0.302496
3	0.0005	20	89.523810	89.099526	88.212925	0.243595	0.275269
4	0.0005	30	90.476191	90.521330	88.593155	0.231816	0.280010
5	0.0005	50	90.952379	90.995258	89.543724	0.212384	0.286377
6	0.0001	20	87.989420	89.573461	85.741442	0.276916	0.276082
7	0.0001	30	88.677251	88.625592	87.832701	0.255324	0.271320
8	0.0001	50	88.148147	89.099526	86.692017	0.261793	0.272812





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

RESULT:

The Multilayer Perceptron (MLP) model was successfully implemented with hyperparameter tuning using Keras Tuner.