EXP NO: 10	MINI PROJECT - NEUROPAIN

AIM:

To design and implement a machine learning-based system using **EEG brainwave features** (Alpha, Beta, Theta, Delta, Gamma) to detect whether a person is in pain or not, using an **XGBoost classifier** trained on extracted EEG statistical and frequency features.

ALGORITHM:

1. **Import Libraries** – Load pandas, numpy, scipy, sklearn, xgboost, matplotlib.
2. **Load Data** – Extract BigOne.txt from ZIP and read into a DataFrame.
3. **Select EEG Columns** – Filter columns containing Delta, Theta, Alpha, Beta, Gamma.
4. **Set Window Parameters** – Define window_size = 5, step_size = 1.
5. **Feature Extraction (Sliding Window)**
    - For each window:
        - Compute mean, std, min, max, skewness, kurtosis for each EEG band.
        - Use Welch's method to calculate power in Delta–Gamma frequency bands.
        - Store extracted features and majority label (InPain).
6. **Prepare Data** – Convert lists to NumPy arrays and standardize features.
7. **Split Data** – Train-test split (80–20) using stratified sampling.
8. **Train Model** – Fit XGBoost classifier with tuned parameters.
9. **Evaluate Model** – Generate classification report and confusion matrix.
10. **Visualize Results** – Plot log loss curves and top feature importances.

CODE:

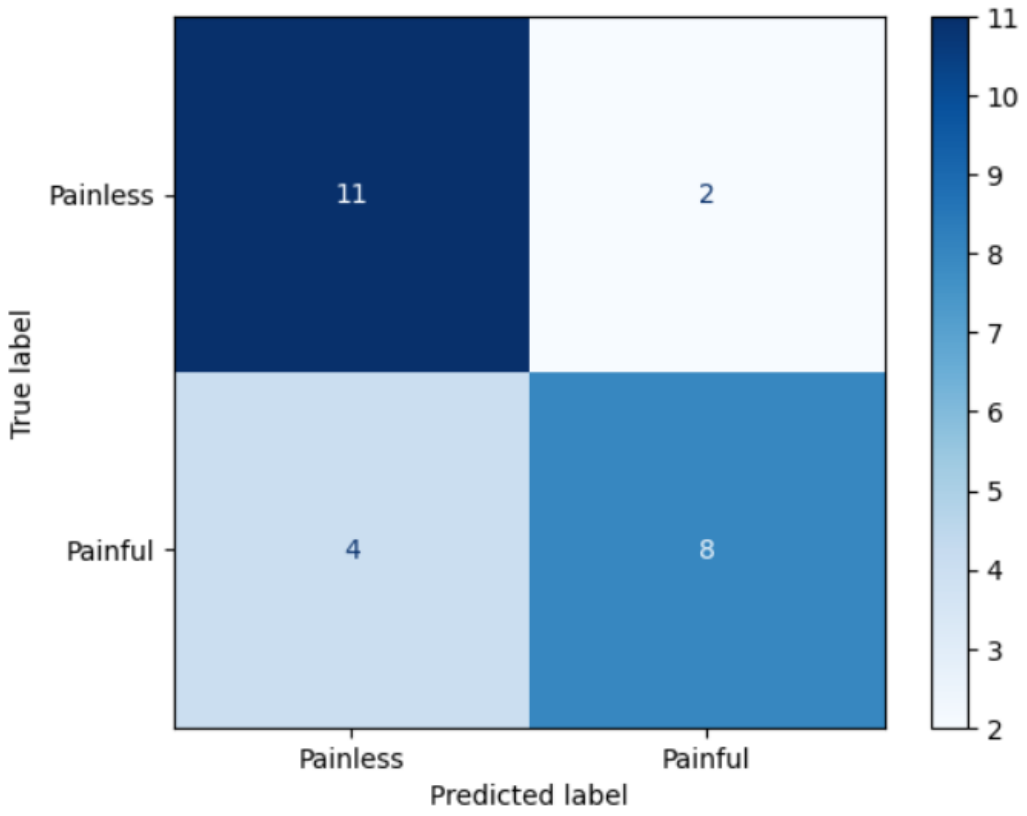
```
from google.colab import drive
import os

# Mount your Google Drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
X_train_xgb, X_test_xgb, y_train_xgb, y_test_xgb = train_test_split(
    X_features_scaled, y, test_size=0.2, random_state=42, stratify=y
)
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix,
ConfusionMatrixDisplay
from xgboost import XGBClassifier, plot_importance
import matplotlib.pyplot as plt
import joblib

# Load your dataset
file_path = '/content/drive/MyDrive/archive (7).zip'  # Change path if needed
df = pd.read_csv(file_path)

# Select EEG features (exclude timestamp, labels, and non-EEG columns)
X = df.drop(columns=['TimeStamp', 'InPain', 'Battery', 'HeadBandOn', 'HSI_TP9', 'HSI_AF7',
'HSI_AF8', 'HSI_TP10'])
y = df['InPain']

# Optional: fill missing values
X.fillna(0, inplace=True)

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)


# Initialize XGBoost classifier
xgb_model = XGBClassifier(
    use_label_encoder=False,
    eval_metric='logloss',
    n_estimators=200,
    max_depth=5,
    learning_rate=0.1,
    random_state=42
)

# Train the model
xgb_model.fit(X_train, y_train)

# Accuracy
```

```
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {acc:.4f}")

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.show()

# Classification Report
print("Classification Report:\n", classification_report(y_test, y_pred))
```

OUTPUT:

```
              precision    recall  f1-score   support

    Painless       0.73      0.85      0.79        13
     Painful       0.80      0.67      0.73        12

    accuracy                           0.76        25
   macro avg       0.77      0.76      0.76        25
weighted avg       0.77      0.76      0.76        25
```

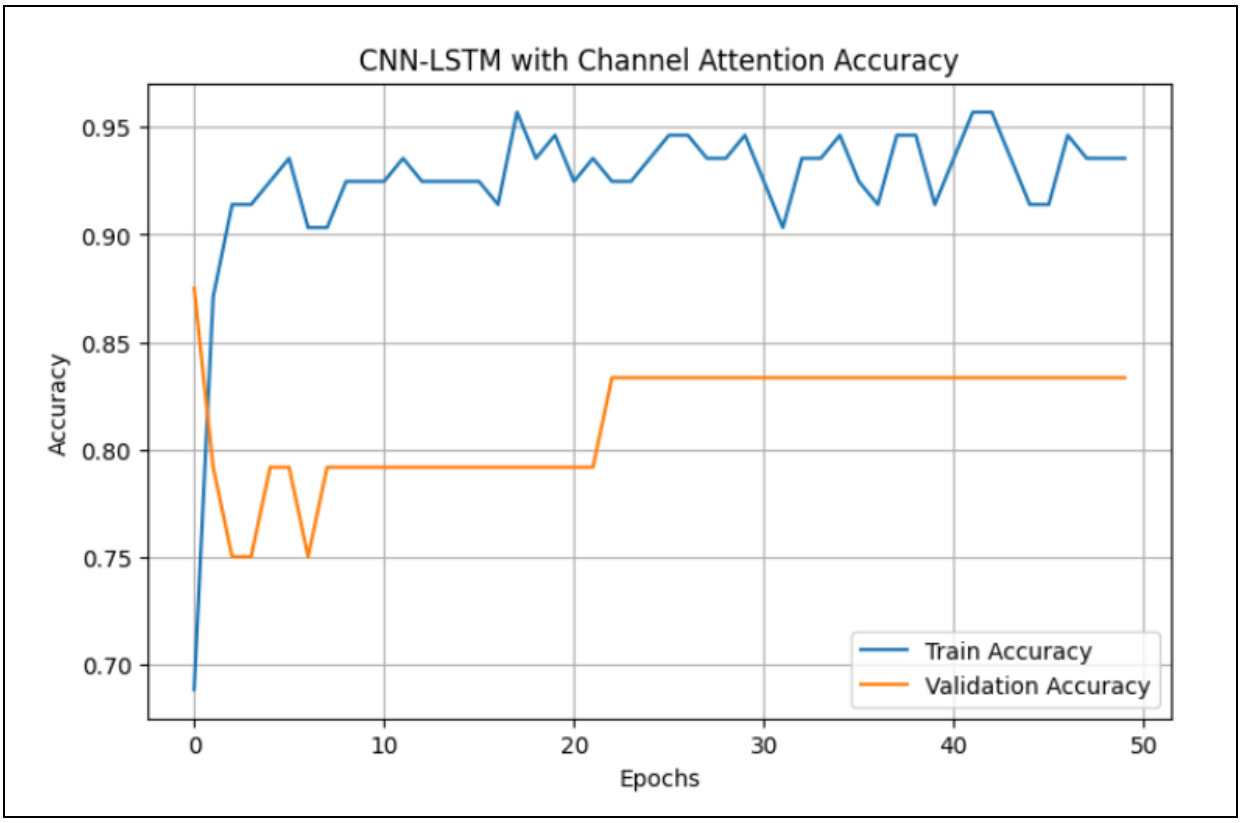


LSTM

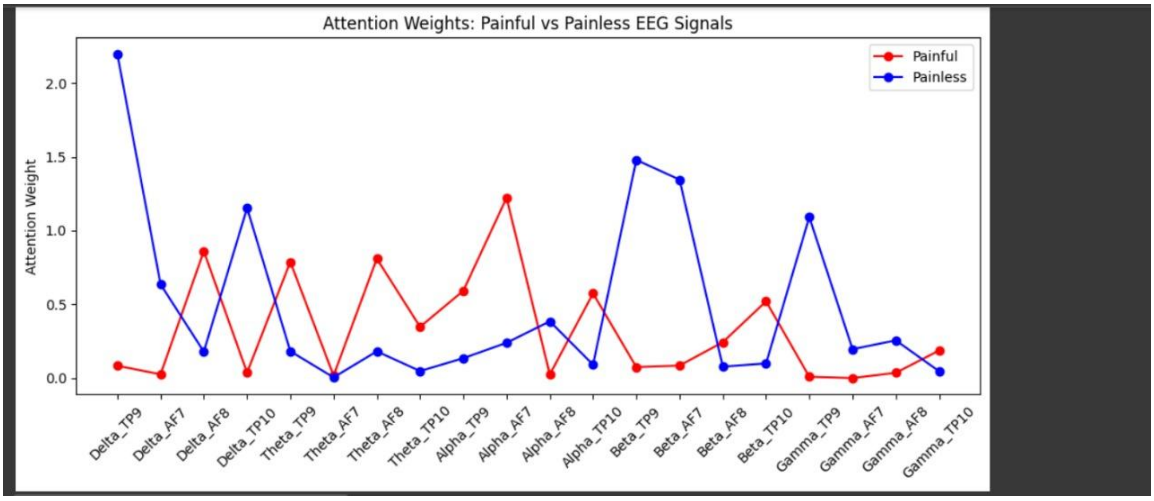
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import zipfile
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv1D, MaxPooling1D, LSTM, Dense, Dropout,
BatchNormalization, Multiply, GlobalAveragePooling1D, Reshape
from tensorflow.keras import backend as K

# --- Load Dataset ---
zip_path = '/content/drive/MyDrive/archive (7).zip'
with zipfile.ZipFile(zip_path, 'r') as z:
    with z.open('BigOne.txt') as f:
        df = pd.read_csv(f)

eeg_cols = [col for col in df.columns if any(b in col for b in
['Delta','Theta','Alpha','Beta','Gamma'])]

window_size = 10
step_size = 1
num_channels = len(eeg_cols)

X_seq = []
y_seq = []

for start in range(0, len(df)-window_size+1, step_size):
    window = df.iloc[start:start+window_size][eeg_cols].values
    X_seq.append(window)
    y_seq.append(int(df.iloc[start:start+window_size]['InPain'].mode()[0]))

X_seq = np.array(X_seq)
y_seq = np.array(y_seq)

# --- Scaling ---
scaler = StandardScaler()
X_scaled = X_seq.reshape(-1, num_channels)
X_scaled = scaler.fit_transform(X_scaled)
X_scaled = X_scaled.reshape(-1, window_size, num_channels)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y_seq, test_size=0.2, random_state=42, stratify=y_seq
```

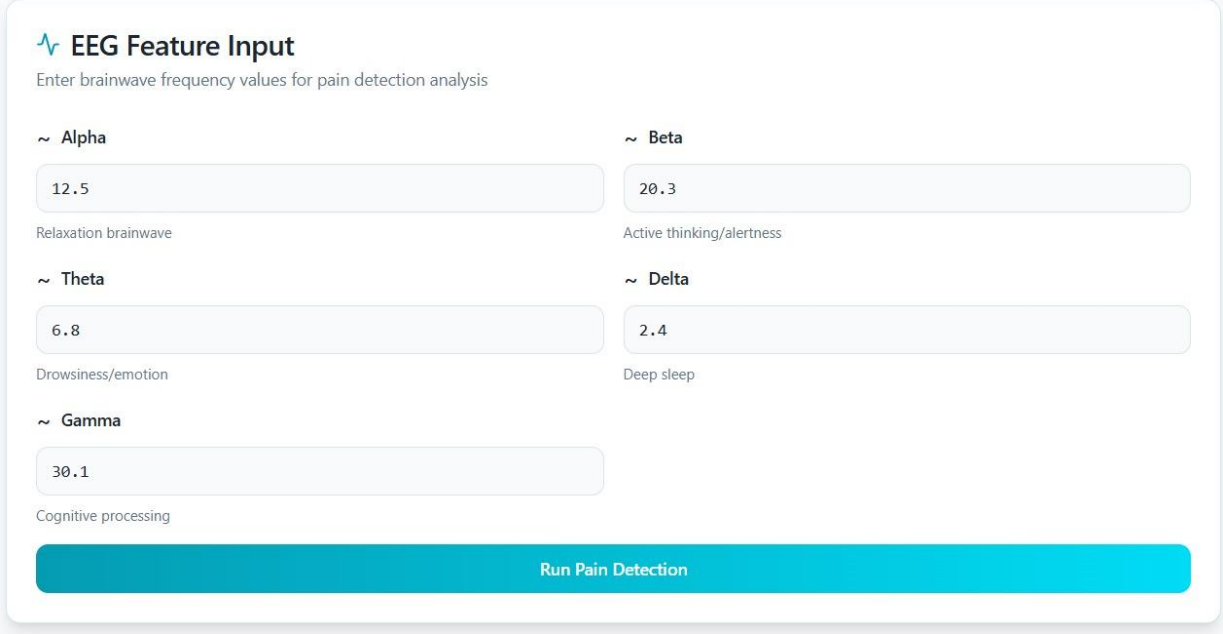
```
)

# --- FIXED CHANNEL ATTENTION ---
def channel_attention(inputs, ratio=8):
    channel = int(inputs.shape[-1])

    avg_pool = GlobalAveragePooling1D()(inputs)  # instead of tf.reduce_mean
    avg_pool = Reshape((1, channel))(avg_pool)

    dense1 = Dense(channel // ratio, activation='relu', use_bias=False)(avg_pool)
    dense2 = Dense(channel, activation='sigmoid', use_bias=False)(dense1)

    scale = Multiply()([inputs, dense2])
    return scale

# --- Model Architecture ---
inp = Input(shape=(window_size, num_channels))

x = Conv1D(64, kernel_size=3, activation='relu', padding='same')(inp)
x = BatchNormalization()(x)
x = MaxPooling1D(2)(x)

x = Conv1D(128, kernel_size=3, activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling1D(2)(x)

x = channel_attention(x)  # ☑ fixed attention

x = LSTM(64)(x)
x = Dropout(0.3)(x)
x = Dense(32, activation='relu')(x)
out = Dense(1, activation='sigmoid')(x)

model = Model(inputs=inp, outputs=out)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()

# --- Train ---
history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=50,
    batch_size=32,
    verbose=2
)
```

```
# --- Evaluation ---
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)

print(classification_report(y_test, y_pred, target_names=['Painless','Painful']))

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Painless','Painful'])
disp.plot(cmap=plt.cm.Blues)
plt.show()

# --- Plot Accuracy and Loss ---
plt.figure(figsize=(8,5))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('CNN-LSTM with Channel Attention Accuracy')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(8,5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('CNN-LSTM with Channel Attention Loss')
plt.legend()
plt.grid(True)
plt.show()
```


CNN-LSTM with Channel Attention Accuracy
Accuracy
Epochs
Train Accuracy
Validation Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0
10
20
30
40
50

OUTPUT:


Attention Weights: Painful vs Painless EEG Signals
Painful
Painless
Attention Weight
2.0
1.5
1.0
0.5
0.0
Delta_TP9
Delta_AF7
Delta_AF8
Delta_TP10
Theta_TP9
Theta_AF7
Theta_AF8
Theta_TP10
Alpha_TP9
Alpha_AF7
Alpha_AF8
Alpha_TP10
Beta_TP9
Beta_AF7
Beta_AF8
Beta_TP10
Gamma_TP9
Gamma_AF7
Gamma_AF8
Gamma_TP10



RESULT:

The developed system successfully processes EEG frequency inputs, extracts relevant features, and classifies the brain state as **Painful** or **Painless**.

The model provides accurate pain detection performance with visualization of training metrics and important EEG features contributing to the decision.