In [1]:
```
#imports
%matplotlib inline
import random
import tensorflow as tf
import numpy as np
import pandas as pd
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Flatten, Activation, Conv1D, MaxPooling1D, Dropout, Lambda, LeakyReLU
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
from skopt import BayesSearchCV
from sklearn.metrics import classification_report
```

```
2023-07-14 18:49:54.831770: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-07-14 18:49:55.742408: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
```

In [2]:
```
data1 = pd.read_csv('/home/mahinur/Desktop/CSV_1.csv')
data2 = pd.read_csv('/home/mahinur/Desktop/CSV_2.csv')
```

In [3]:
```
merged_data = pd.merge(data1, data2, on='sid')
numeric_columns = merged_data.select_dtypes(include=[float, int]).columns
merged_data = merged_data[numeric_columns]

# Normalize the merged data using Min-Max scaling
scaler = MinMaxScaler()
normalized_data = pd.DataFrame(scaler.fit_transform(merged_data), columns=merged_data.columns)

# Save the normalized data to a new CSV file
normalized_data.to_csv('/home/mahinur/Desktop/normalized_data.csv', index=False)
```

In [4]:
```
# Load the normalized data from the CSV file
normalized_data = pd.read_csv('/home/mahinur/Desktop/normalized_data.csv')

# Extract the features (X) and target (y) columns
X = normalized_data.drop('output1', axis=1).values
y = normalized_data['output1'].values

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the parameter search space with adjusted bounds
param_space = {
    'learning_rate': (0.01, 1.0, 'log-uniform'),
    'max_depth': (3, 11),
    'n_estimators': (50, 201),
    'gamma': (0.01, 1.0, 'log-uniform'),
    'min_child_weight': (1, 11),
}

# Create the XGBoost classifier
model = XGBClassifier()

# Perform Bayesian optimization for hyperparameter search
opt = BayesSearchCV(model, param_space, n_iter=50, cv=5, scoring='accuracy', random_state=42)
opt.fit(X_train, y_train)

# Get the best model and its hyperparameters
best_model = opt.best_estimator_
best_params = opt.best_params_
print("Best Hyperparameters:", best_params)

# Predict the target values using the best model
y_pred = best_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.4f}")
# Generate the classification report
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
Best Hyperparameters: OrderedDict([('gamma', 0.10325309897613151), ('learning_rate', 0.08905744151836509), ('max_depth', 3), ('min_child_weight', 1), ('n_estimators', 188)])
Test Accuracy: 0.9171
Classification Report:
              precision    recall  f1-score   support

         0.0       0.92      1.00      0.96      1372
         1.0       0.43      0.02      0.05       123

    accuracy                           0.92      1495
   macro avg       0.67      0.51      0.50      1495
weighted avg       0.88      0.92      0.88      1495
```

In [ ]:

In [ ]: