**Codes:**

**CNN Model**

#CNN Model

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense

# Load dataset

file\_path = "/content/drive/MyDrive/-Early-Detection-of-Alzheimer-s-Disease-Based-on-Handwriting-Analysis-Using-CNN-Models-Visualiz-main/data (3).csv"

df = pd.read\_csv(file\_path)

# Drop ID column and separate features and target

df = df.drop(columns=['ID'])  # Remove identifier column

X = df.drop(columns=['class'])  # Features

y = df['class']  # Target

# Encode target labels

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y)

y\_encoded = tf.keras.utils.to\_categorical(y\_encoded)

# Normalize features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)  # Scale entire dataset

X\_scaled = np.expand\_dims(X\_scaled, axis=2)  # Reshape for CNN input

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y\_encoded, test\_size=0.2, random\_state=42)

# Build CNN Model

model = Sequential([

    Conv1D(32, kernel\_size=3, activation='relu', input\_shape=(X\_train.shape[1], 1)),

    MaxPooling1D(pool\_size=2),

    Flatten(),

    Dense(64, activation='relu'),

    Dense(y\_train.shape[1], activation='softmax')

])

# Compile Model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train Model

model.fit(X\_train, y\_train, epochs=20, batch\_size=10, validation\_data=(X\_test, y\_test))

# Predictions on full dataset

y\_full\_pred = model.predict(X\_scaled)

y\_full\_pred\_classes = np.argmax(y\_full\_pred, axis=1)

y\_true\_classes = np.argmax(y\_encoded, axis=1)

# Accuracy on full dataset

full\_accuracy = accuracy\_score(y\_true\_classes, y\_full\_pred\_classes)

print(f"Full Dataset Accuracy: {full\_accuracy:.4f}")

# Classification report

print("\nClassification Report:\n", classification\_report(y\_true\_classes, y\_full\_pred\_classes, target\_names=label\_encoder.classes\_))

# Confusion matrix for all 174 samples

conf\_matrix\_full = confusion\_matrix(y\_true\_classes, y\_full\_pred\_classes)

# Visualization of Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix\_full, annot=True, fmt='d', cmap='Blues',

            xticklabels=label\_encoder.classes\_, yticklabels=label\_encoder.classes\_)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.show()

# Bar Chart for Healthy vs. Unhealthy Predictions

labels = label\_encoder.classes\_

unique, counts = np.unique(y\_full\_pred\_classes, return\_counts=True)

plt.figure(figsize=(6, 4))

plt.bar(labels, counts, color=['green', 'red'])

plt.xlabel('Class')

plt.ylabel('Count')

plt.title('Prediction Distribution')

plt.show()

**2. AdaBoost**

# Train AdaBoost

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

# Load dataset

file\_path =  "/content/drive/MyDrive/-Early-Detection-of-Alzheimer-s-Disease-Based-on-Handwriting-Analysis-Using-CNN-Models-Visualiz-main/data (3).csv"

df = pd.read\_csv(file\_path)

# Drop ID column and separate features and target

df = df.drop(columns=['ID'])  # Remove identifier column

X = df.drop(columns=['class'])  # Features

y = df['class']  # Target

# Encode target labels

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y)

# Normalize features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)  # Scale entire dataset

# Train-Test Split (80% Training, 20% Testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y\_encoded, test\_size=0.2, random\_state=42)

# Train AdaBoost Classifier with Reduced Complexity

base\_model = DecisionTreeClassifier(max\_depth=1)  # Reduce depth for a weaker learner

adaboost\_model = AdaBoostClassifier(base\_model, n\_estimators=20, learning\_rate=0.3, random\_state=42)

adaboost\_model.fit(X\_train, y\_train)

# Predictions on Test Set (to check accuracy)

y\_test\_pred = adaboost\_model.predict(X\_test)

# Accuracy on Test Set

test\_accuracy = accuracy\_score(y\_test, y\_test\_pred)

print(f"Test Set Accuracy: {test\_accuracy:.4f}")  # Expected ~90%

# Predictions on Full Dataset (All 174 Samples)

y\_full\_pred = adaboost\_model.predict(X\_scaled)

# Classification Report

print("\nClassification Report:\n", classification\_report(y\_encoded, y\_full\_pred, target\_names=label\_encoder.classes\_))

# Confusion Matrix for All 174 Samples

conf\_matrix\_full = confusion\_matrix(y\_encoded, y\_full\_pred)

# Visualization of Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix\_full, annot=True, fmt='d', cmap='Blues',

            xticklabels=label\_encoder.classes\_, yticklabels=label\_encoder.classes\_)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.show()

# Bar Chart for Healthy vs. Unhealthy Predictions

labels = label\_encoder.classes\_

unique, counts = np.unique(y\_full\_pred\_classes, return\_counts=True)

plt.figure(figsize=(6, 4))

plt.bar(labels, counts, color=['green', 'red'])

plt.xlabel('Class')

plt.ylabel('Count')

plt.title('Prediction Distribution')

plt.show()

**3. Gradient Boosting**

# Train Gradient Boosting

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

# Load dataset

file\_path = "/content/drive/MyDrive/-Early-Detection-of-Alzheimer-s-Disease-Based-on-Handwriting-Analysis-Using-CNN-Models-Visualiz-main/data (3).csv"

df = pd.read\_csv(file\_path)

# Drop ID column and separate features and target

df = df.drop(columns=['ID'])  # Remove identifier column

X = df.drop(columns=['class'])  # Features

y = df['class']  # Target

# Encode target labels

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y)

# Normalize features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)  # Scale entire dataset

# Train-Test Split (80% Training, 20% Testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y\_encoded, test\_size=0.2, random\_state=42)

# Train Gradient Boosting Classifier with Reduced Complexity

gb\_model = GradientBoostingClassifier(n\_estimators=50, learning\_rate=0.1, max\_depth=3, random\_state=42)

gb\_model.fit(X\_train, y\_train)

# Predictions on Test Set (to check accuracy)

y\_test\_pred = gb\_model.predict(X\_test)

# Accuracy on Test Set

test\_accuracy = accuracy\_score(y\_test, y\_test\_pred)

print(f"Test Set Accuracy: {test\_accuracy:.4f}")  # Expected ~90%

# Predictions on Full Dataset (All 174 Samples)

y\_full\_pred = gb\_model.predict(X\_scaled)

# Classification Report

print("\nClassification Report:\n", classification\_report(y\_encoded, y\_full\_pred, target\_names=label\_encoder.classes\_))

# Confusion Matrix for All 174 Samples

conf\_matrix\_full = confusion\_matrix(y\_encoded, y\_full\_pred)

# Visualization of Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix\_full, annot=True, fmt='d', cmap='Blues',

            xticklabels=label\_encoder.classes\_, yticklabels=label\_encoder.classes\_)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.show()

# Bar Chart for Healthy vs. Unhealthy Predictions

labels = label\_encoder.classes\_

unique, counts = np.unique(y\_full\_pred\_classes, return\_counts=True)

plt.figure(figsize=(6, 4))

plt.bar(labels, counts, color=['green', 'red'])

plt.xlabel('Class')

plt.ylabel('Count')

plt.title('Prediction Distribution')

plt.show()

# Tracking Training Loss and Accuracy for Each Iteration

train\_loss = []

train\_accuracy = []

for y\_pred\_train in gb\_model.staged\_predict(X\_train):

    train\_loss.append(np.mean(y\_pred\_train != y\_train))  # Misclassification rate (1 - Accuracy)

    train\_accuracy.append(accuracy\_score(y\_train, y\_pred\_train))

# Plotting Training Loss and Accuracy

iterations = np.arange(1, len(train\_loss) + 1)

plt.figure(figsize=(10, 6))

# Training Loss Plot

plt.plot(iterations, train\_loss, label='Training Loss', color='red', marker='o')

# Training Accuracy Plot

plt.plot(iterations, train\_accuracy, label='Training Accuracy', color='blue', marker='o')

# Plot Customization

plt.xlabel('Iterations (n\_estimators)')

plt.ylabel('Metrics')

plt.title('Training Loss and Accuracy Plot')

plt.legend()

plt.grid()

plt.show()