Here’s the code for each of the four hybrid models:

**1. CNN Model:**

This model uses Convolutional layers to extract spatial features from MFCC input.

import numpy as np

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.utils import to\_categorical

# Reshaping data for CNN

timesteps = 1 # Treat each instance as a single time step

n\_features = X\_mfcc\_train.shape[1] # Number of MFCC features

X\_mfcc\_train\_cnn = X\_mfcc\_train.reshape((X\_mfcc\_train.shape[0], timesteps, n\_features))

X\_mfcc\_test\_cnn = X\_mfcc\_test.reshape((X\_mfcc\_test.shape[0], timesteps, n\_features))

# One-hot encode labels

num\_classes = len(np.unique(y\_mfcc\_train))

y\_train\_cnn = to\_categorical(y\_mfcc\_train, num\_classes)

y\_test\_cnn = to\_categorical(y\_mfcc\_test, num\_classes)

# CNN Model

cnn\_model = Sequential()

cnn\_model.add(Conv1D(64, kernel\_size=3, activation='relu', input\_shape=(timesteps, n\_features)))

cnn\_model.add(MaxPooling1D(pool\_size=2))

cnn\_model.add(Flatten())

cnn\_model.add(Dropout(0.5))

cnn\_model.add(Dense(num\_classes, activation='softmax'))

# Compile and Train the Model

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

cnn\_model.fit(X\_mfcc\_train\_cnn, y\_train\_cnn, epochs=20, batch\_size=32, validation\_split=0.2, verbose=2)

**2. CNN-LSTM Hybrid Model:**

This model first extracts features using a CNN and then processes them with an LSTM for temporal patterns.

from tensorflow.keras.layers import LSTM

# CNN-LSTM Model

cnn\_lstm\_model = Sequential()

cnn\_lstm\_model.add(Conv1D(64, kernel\_size=3, activation='relu', input\_shape=(timesteps, n\_features)))

cnn\_lstm\_model.add(MaxPooling1D(pool\_size=2))

cnn\_lstm\_model.add(LSTM(64, activation='relu'))

cnn\_lstm\_model.add(Dropout(0.5))

cnn\_lstm\_model.add(Dense(num\_classes, activation='softmax'))

# Compile and Train the Model

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

cnn\_lstm\_model.fit(X\_mfcc\_train\_cnn, y\_train\_cnn, epochs=20, batch\_size=32, validation\_split=0.2, verbose=2)

**3. CNN-BiLSTM Hybrid Model:**

This model combines CNN with Bidirectional LSTM for better capturing context in sequences.

from tensorflow.keras.layers import Bidirectional

# CNN-BiLSTM Model

cnn\_bilstm\_model = Sequential()

cnn\_bilstm\_model.add(Conv1D(64, kernel\_size=3, activation='relu', input\_shape=(timesteps, n\_features)))

cnn\_bilstm\_model.add(MaxPooling1D(pool\_size=2))

cnn\_bilstm\_model.add(Bidirectional(LSTM(64, activation='relu')))

cnn\_bilstm\_model.add(Dropout(0.5))

cnn\_bilstm\_model.add(Dense(num\_classes, activation='softmax'))

# Compile and Train the Model

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

cnn\_bilstm\_model.fit(X\_mfcc\_train\_cnn, y\_train\_cnn, epochs=20, batch\_size=32, validation\_split=0.2, verbose=2)

**4. CNN-RNN-SVM Hybrid Model:**

This model uses a CNN to extract features, an RNN to learn sequence patterns, and then uses an SVM for classification.

**Note**: You’ll first extract the CNN-RNN feature representation and then feed that into an SVM for classification. Here, we'll use an LSTM as the RNN part.

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, f1\_score, precision\_score, recall\_score

# CNN-RNN Model to extract features

cnn\_rnn\_model = Sequential()

cnn\_rnn\_model.add(Conv1D(64, kernel\_size=3, activation='relu', input\_shape=(timesteps, n\_features)))

cnn\_rnn\_model.add(MaxPooling1D(pool\_size=2))

cnn\_rnn\_model.add(LSTM(64, activation='relu', return\_sequences=False)) # Extract feature representation

cnn\_rnn\_model.add(Flatten()) # Flatten the output for SVM input

# Compile and Train the CNN-RNN model

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

cnn\_rnn\_model.fit(X\_mfcc\_train\_cnn, y\_train\_cnn, epochs=20, batch\_size=32, validation\_split=0.2, verbose=2)

# Extract CNN-RNN features for SVM classification

train\_features\_cnn\_rnn = cnn\_rnn\_model.predict(X\_mfcc\_train\_cnn)

test\_features\_cnn\_rnn = cnn\_rnn\_model.predict(X\_mfcc\_test\_cnn)

# Train SVM on extracted features

svm\_clf = SVC(kernel='linear')

svm\_clf.fit(train\_features\_cnn\_rnn, y\_mfcc\_train)

# Predict and evaluate the model using SVM

y\_pred\_svm\_cnn\_rnn = svm\_clf.predict(test\_features\_cnn\_rnn)

# Calculate metrics

accuracy = accuracy\_score(y\_mfcc\_test, y\_pred\_svm\_cnn\_rnn)

f1 = f1\_score(y\_mfcc\_test, y\_pred\_svm\_cnn\_rnn, average='weighted')

precision = precision\_score(y\_mfcc\_test, y\_pred\_svm\_cnn\_rnn, average='weighted')

recall = recall\_score(y\_mfcc\_test, y\_pred\_svm\_cnn\_rnn, average='weighted')

print(f'Accuracy: {accuracy \* 100:.2f}%')

print(f'F1 Score: {f1:.2f}')

print(f'Precision: {precision:.2f}')

print(f'Recall: {recall:.2f}')

**Summary:**

* **CNN Model** extracts spatial features directly from MFCCs.
* **CNN-LSTM Hybrid Model** leverages the combination of CNN for feature extraction and LSTM for capturing temporal dependencies.
* **CNN-BiLSTM Hybrid Model** enhances sequence modeling using Bidirectional LSTMs for capturing context in both forward and backward directions.
* **CNN-RNN-SVM Hybrid Model** uses CNN and LSTM to extract features, followed by an SVM classifier for final classification.

These hybrid architectures can capture various spatial and temporal patterns in your MFCC audio data. Feel free to try and fine-tune them according to your dataset's specific requirements

5. The **CNN-XGBoost Hybrid Model** approach combines the feature extraction capability of a Convolutional Neural Network (CNN) with the powerful classification capabilities of the XGBoost algorithm. The idea is to use CNN layers to process and extract high-level features from the MFCC data and then feed these features into an XGBoost classifier.

**Step-by-Step Approach:**

1. **Use a CNN to extract high-level features** from the MFCC data.
2. **Flatten the CNN output** to create a feature vector.
3. **Train an XGBoost model** using the feature vector generated by the CNN

import numpy as np

from tensorflow.keras.models import Sequential

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

from xgboost import XGBClassifier

from sklearn.metrics import accuracy\_score, f1\_score, precision\_score, recall\_score

# Reshaping data for CNN input

timesteps = 1 # Treat each instance as a single time step

n\_features = X\_mfcc\_train.shape[1] # Number of MFCC features

X\_mfcc\_train\_cnn = X\_mfcc\_train.reshape((X\_mfcc\_train.shape[0], timesteps, n\_features))

X\_mfcc\_test\_cnn = X\_mfcc\_test.reshape((X\_mfcc\_test.shape[0], timesteps, n\_features))

# CNN Model for feature extraction

cnn\_xgb\_model = Sequential()

cnn\_xgb\_model.add(Conv1D(64, kernel\_size=3, activation='relu', input\_shape=(timesteps, n\_features)))

cnn\_xgb\_model.add(MaxPooling1D(pool\_size=2))

cnn\_xgb\_model.add(Flatten())

# Compile and Train the CNN Model

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

cnn\_xgb\_model.fit(X\_mfcc\_train\_cnn, np.array(y\_mfcc\_train), epochs=20, batch\_size=32, validation\_split=0.2, verbose=2)

# Extract features from CNN

train\_features\_cnn = cnn\_xgb\_model.predict(X\_mfcc\_train\_cnn)

test\_features\_cnn = cnn\_xgb\_model.predict(X\_mfcc\_test\_cnn)

# Train XGBoost model on extracted features

xgb\_clf = XGBClassifier(n\_estimators=100, max\_depth=5, random\_state=42, use\_label\_encoder=False, eval\_metric='mlogloss')

xgb\_clf.fit(train\_features\_cnn, y\_mfcc\_train)

# Make predictions using XGBoost

y\_pred\_xgb\_cnn = xgb\_clf.predict(test\_features\_cnn)

# Calculate metrics

accuracy = accuracy\_score(y\_mfcc\_test, y\_pred\_xgb\_cnn)

f1 = f1\_score(y\_mfcc\_test, y\_pred\_xgb\_cnn, average='weighted')

precision = precision\_score(y\_mfcc\_test, y\_pred\_xgb\_cnn, average='weighted')

recall = recall\_score(y\_mfcc\_test, y\_pred\_xgb\_cnn, average='weighted')

print(f'Accuracy: {accuracy \* 100:.2f}%')

print(f'F1 Score: {f1:.2f}')

print(f'Precision: {precision:.2f}')

print(f'Recall: {recall:.2f}')

**Explanation:**

1. **CNN for Feature Extraction**:
   * The CNN model consists of a convolutional layer and a pooling layer to extract spatial features from MFCCs.
   * The output of the CNN is flattened to create a 1D feature vector.
2. **Feature Feeding to XGBoost**:
   * The feature vector generated by the CNN is used as the input to the XGBoost classifier.
   * The XGBoost classifier is trained on the extracted CNN features, which are expected to capture key information about the audio data.
3. **Prediction and Evaluation**:
   * The trained XGBoost model is used to predict the test data based on the extracted CNN features.
   * Performance metrics (accuracy, F1 score, precision, and recall) are calculated to evaluate the model.

**Benefits of the CNN-XGBoost Hybrid Model:**

* CNNs excel at extracting spatial patterns, making them suitable for feature extraction from MFCC representations.
* XGBoost is known for its strong predictive performance, especially when decision boundaries are non-linear or complex.

Feel free to fine-tune the hyperparameters of the CNN and XGBoost components to achieve better results! Let me know if you need further customizations or guidance.

Ensemble learning combines multiple models to improve the overall performance, usually by averaging or voting on predictions. For audio classification tasks, combining a CNN (which excels at feature extraction) with ensemble techniques like Bagging, Boosting, or Stacking can yield stronger results.

Here are three popular ensemble techniques:

Bagging with Random Forest: Utilize the Random Forest classifier on CNN features.

Boosting with XGBoost: Use CNN features to train an XGBoost classifier, which we've partially done in the CNN-XGBoost hybrid model.

Stacking: Combine predictions from CNN, XGBoost, and other classifiers like SVM or KNN using a meta-classifier.

Let’s implement Stacking as an ensemble learning approach:

1. Stacking Approach with CNN Features:

We’ll use the CNN model to extract features and then apply stacking with multiple base classifiers, combining predictions using a meta-classifier.

Step-by-Step:

Extract features using a CNN.

Train multiple base classifiers (e.g., RandomForest, XGBoost, SVM).

Combine their predictions using a meta-classifier (e.g., Logistic Regression).

import numpy as np

from tensorflow.keras.models import Sequential

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.ensemble import StackingClassifier

from sklearn.metrics import accuracy\_score, f1\_score, precision\_score, recall\_score

# Step 1: Reshape data for CNN

timesteps = 1

n\_features = X\_mfcc\_train.shape[1]

X\_mfcc\_train\_cnn = X\_mfcc\_train.reshape((X\_mfcc\_train.shape[0], timesteps, n\_features))

X\_mfcc\_test\_cnn = X\_mfcc\_test.reshape((X\_mfcc\_test.shape[0], timesteps, n\_features))

# Step 2: Define a CNN model for feature extraction

cnn\_model = Sequential()

cnn\_model.add(Conv1D(64, kernel\_size=3, activation='relu', input\_shape=(timesteps, n\_features)))

cnn\_model.add(MaxPooling1D(pool\_size=2))

cnn\_model.add(Flatten())

# Compile and train the CNN model

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

cnn\_model.fit(X\_mfcc\_train\_cnn, np.array(y\_mfcc\_train), epochs=20, batch\_size=32, validation\_split=0.2, verbose=2)

# Step 3: Extract CNN features

train\_features = cnn\_model.predict(X\_mfcc\_train\_cnn)

test\_features = cnn\_model.predict(X\_mfcc\_test\_cnn)

# Step 4: Define base classifiers and meta-classifier for stacking

base\_learners = [

('rf', RandomForestClassifier(n\_estimators=100, random\_state=42)),

('svm', SVC(kernel='linear', probability=True)),

('xgb', XGBClassifier(n\_estimators=100, max\_depth=5, random\_state=42, use\_label\_encoder=False, eval\_metric='mlogloss'))

]

# Meta-classifier (Logistic Regression in this case)

meta\_learner = LogisticRegression(random\_state=42)

# Step 5: Create a stacking classifier

stacking\_clf = StackingClassifier(estimators=base\_learners, final\_estimator=meta\_learner, cv=5)

# Train the stacking classifier

stacking\_clf.fit(train\_features, y\_mfcc\_train)

# Step 6: Make predictions with the stacking model

y\_pred\_stacking = stacking\_clf.predict(test\_features)

# Step 7: Evaluate the model

accuracy = accuracy\_score(y\_mfcc\_test, y\_pred\_stacking)

f1 = f1\_score(y\_mfcc\_test, y\_pred\_stacking, average='weighted')

precision = precision\_score(y\_mfcc\_test, y\_pred\_stacking, average='weighted')

recall = recall\_score(y\_mfcc\_test, y\_pred\_stacking, average='weighted')

print(f'Ensemble Accuracy: {accuracy \* 100:.2f}%')

print(f'Ensemble F1 Score: {f1:.2f}')

print(f'Ensemble Precision: {precision:.2f}')

print(f'Ensemble Recall: {recall:.2f}')

**Explanation:**

1. **CNN Feature Extraction**:
   * A CNN extracts spatial patterns from MFCC features.
   * Features are then flattened and used as input to the ensemble classifiers.
2. **Base Classifiers**:
   * We use three base classifiers: Random Forest, SVM, and XGBoost. These models are diverse, capturing different patterns in the data.
3. **Stacking**:
   * Stacking combines the predictions from the base classifiers and trains a meta-classifier (in this case, **Logistic Regression**) on these predictions to make the final decision.
   * You can customize the base classifiers or use other meta-classifiers based on the problem.

**Benefits of Stacking:**

* Stacking leverages the strengths of multiple algorithms to provide a more robust and generalized model.
* It combines different types of classifiers (e.g., SVM, Random Forest, XGBoost) to capture complementary patterns in the data.

This **CNN + Ensemble learning approach** can lead to better performance in audio classification tasks due to the complementary nature of the models. Feel free to experiment with other ensemble techniques or base classifiers as per your needs!