

## ✓ FINAL TEAM PROJECT

AAI - 511

ARIFA KOKAB (INDIVIDUAL)


### IMPORT LIBRARIES

```
import os
import numpy as np
import pretty_midi
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

### LOAD THE DATA

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')


# Define paths to the dataset
drive_path = '/content/drive/MyDrive/Composers/'
composers = ['bach', 'handel', 'chopin', 'mozart']
```

 Mounted at /content/drive

```
# Helper function to convert MIDI files to sequences of note pitches
def midi_to_sequence(midi_file):
    try:
        midi_data = pretty_midi.PrettyMIDI(midi_file)
        notes = []
        for instrument in midi_data.instruments:
            for note in instrument.notes:
                notes.append(note.pitch)
        return np.array(notes)
    except:
        return None
```

```
# Load and preprocess data
def load_data():
    sequences = []
    labels = []
    for composer in composers:
        composer_folder = os.path.join(drive_path, composer)
        for midi_file in os.listdir(composer_folder):
            file_path = os.path.join(composer_folder, midi_file)
            sequence = midi_to_sequence(file_path)
            if sequence is not None:
                sequences.append(sequence)
                labels.append(composer)
    return sequences, labels
```

```
sequences, labels = load_data()
```

 /usr/local/lib/python3.10/dist-packages/pretty\_midi/pretty\_midi.py:100: RuntimeWarning: Tempo, Key or Time signature change events found on non-zero tracks. This is not a warnings.warn()

### PRE-PROCESS THE DATA

```
# Encode labels
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(labels)

# Pad sequences to the same length
max_seq_length = max([len(seq) for seq in sequences])
padded_sequences = np.array([np.pad(seq, (0, max_seq_length - len(seq)), 'constant') for seq in sequences])
```

### SPLIT THE DATA INTO TRAINING AND TESTING SETS

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, encoded_labels, test_size=0.2, random_state=42)

# Reshape for hybrid model input (LSTM expects 2D, CNN expects 3D)
X_train = X_train.reshape(-1, max_seq_length, 1)
X_test = X_test.reshape(-1, max_seq_length, 1)
```

### BUILD THE HYBRID LSTM-CNN MODEL

```
def build_hybrid_model(input_shape):
    # Input layer
    input_layer = tf.keras.Input(shape=input_shape)

    # LSTM layers with reduced units
    lstm_layer = layers.LSTM(32, return_sequences=True)(input_layer)
    lstm_layer = layers.LSTM(16, return_sequences=True)(lstm_layer)

    # Reshape for CNN input (after LSTM processing)
    cnn_input = layers.Reshape((lstm_layer.shape[1], lstm_layer.shape[2], 1))(lstm_layer)

    # CNN layers with reduced filters
    conv_layer = layers.Conv2D(16, kernel_size=(3, 1), activation='relu')(cnn_input)
    conv_layer = layers.MaxPooling2D(pool_size=(2, 1))(conv_layer)
    conv_layer = layers.Conv2D(32, kernel_size=(3, 1), activation='relu')(conv_layer)
    conv_layer = layers.MaxPooling2D(pool_size=(2, 1))(conv_layer)

    # Flatten the output
    flat_layer = layers.Flatten()(conv_layer)

    # Dense layers with reduced units
    dense_layer = layers.Dense(16, activation='relu')(flat_layer)
    output_layer = layers.Dense(4, activation='softmax')(dense_layer) # 4 classes for 4 composers

    # Create the model
    model = models.Model(inputs=input_layer, outputs=output_layer)

    return model

# Build the hybrid model
hybrid_model = build_hybrid_model((max_seq_length, 1))

# Compile the model
hybrid_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Print the model summary
hybrid_model.summary()
```

Model: "functional\_1"

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 31879, 1)	0
lstm_5 (LSTM)	(None, 31879, 32)	4,352
lstm_6 (LSTM)	(None, 31879, 16)	3,136
reshape_2 (Reshape)	(None, 31879, 16, 1)	0
conv2d_4 (Conv2D)	(None, 31877, 16, 16)	64
max_pooling2d_4 (MaxPooling2D)	(None, 15938, 16, 16)	0
conv2d_5 (Conv2D)	(None, 15936, 16, 32)	1,568
max_pooling2d_5 (MaxPooling2D)	(None, 7968, 16, 32)	0
flatten_2 (Flatten)	(None, 4079616)	0
dense_3 (Dense)	(None, 16)	65,273,872
dense_4 (Dense)	(None, 4)	68

Total params: 65,283,060 (249.04 MB)  
Trainable params: 65,283,060 (249.04 MB)  
Non-trainable params: 0 (0.00 B)

TRAIN THE MODEL

```
# Train the hybrid model with a smaller batch size
hybrid_model.fit(X_train, y_train, epochs=20, batch_size=8, validation_split=0.2)
```

```
Epoch 1/20
14/14 ----- 34s 2s/step - accuracy: 0.4943 - loss: 2.0532 - val_accuracy: 0.5185 - val_loss: 1.8769
Epoch 2/20
14/14 ----- 28s 2s/step - accuracy: 0.6684 - loss: 0.9736 - val_accuracy: 0.4444 - val_loss: 1.5567
Epoch 3/20
14/14 ----- 28s 2s/step - accuracy: 0.6403 - loss: 0.9234 - val_accuracy: 0.4815 - val_loss: 1.7429
Epoch 4/20
14/14 ----- 28s 2s/step - accuracy: 0.6592 - loss: 0.7681 - val_accuracy: 0.5556 - val_loss: 1.3489
Epoch 5/20
14/14 ----- 28s 2s/step - accuracy: 0.7792 - loss: 0.5761 - val_accuracy: 0.4815 - val_loss: 1.6744
Epoch 6/20
14/14 ----- 28s 2s/step - accuracy: 0.8405 - loss: 0.4320 - val_accuracy: 0.4815 - val_loss: 1.8729
Epoch 7/20
14/14 ----- 28s 2s/step - accuracy: 0.9461 - loss: 0.2349 - val_accuracy: 0.4815 - val_loss: 2.2709
Epoch 8/20
14/14 ----- 28s 2s/step - accuracy: 0.9932 - loss: 0.0778 - val_accuracy: 0.4444 - val_loss: 3.0913
Epoch 9/20
14/14 ----- 28s 2s/step - accuracy: 0.8899 - loss: 0.1903 - val_accuracy: 0.3704 - val_loss: 2.6039
Epoch 10/20
14/14 ----- 28s 2s/step - accuracy: 1.0000 - loss: 0.0652 - val_accuracy: 0.4074 - val_loss: 2.5672
Epoch 11/20
14/14 ----- 28s 2s/step - accuracy: 0.9875 - loss: 0.0348 - val_accuracy: 0.4444 - val_loss: 2.6092
Epoch 12/20
14/14 ----- 28s 2s/step - accuracy: 1.0000 - loss: 0.0122 - val_accuracy: 0.4444 - val_loss: 2.7032
Epoch 13/20
14/14 ----- 28s 2s/step - accuracy: 0.9806 - loss: 0.0240 - val_accuracy: 0.4815 - val_loss: 2.7974
Epoch 14/20
```

```

14/14 ————— 28s 2s/step - accuracy: 1.0000 - loss: 0.0037 - val_accuracy: 0.4815 - val_loss: 3.0092
Epoch 15/20
14/14 ————— 28s 2s/step - accuracy: 1.0000 - loss: 0.0015 - val_accuracy: 0.4444 - val_loss: 3.0774
Epoch 16/20
14/14 ————— 28s 2s/step - accuracy: 1.0000 - loss: 0.0011 - val_accuracy: 0.4444 - val_loss: 3.1779
Epoch 17/20
14/14 ————— 28s 2s/step - accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 0.4444 - val_loss: 3.3137
Epoch 18/20
14/14 ————— 28s 2s/step - accuracy: 1.0000 - loss: 9.4709e-04 - val_accuracy: 0.4815 - val_loss: 3.4241
Epoch 19/20
14/14 ————— 28s 2s/step - accuracy: 1.0000 - loss: 7.4010e-04 - val_accuracy: 0.4815 - val_loss: 3.4546
Epoch 20/20
14/14 ————— 28s 2s/step - accuracy: 1.0000 - loss: 3.8275e-04 - val_accuracy: 0.4444 - val_loss: 3.5237
keras.src.callbacks.history.History at 0x7874e5892bf0>

```

## EVALUATE THE MODEL

```

# Evaluate the model on the test set
predictions = hybrid_model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

```

2/2 ————— 3s 865ms/step

```

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, predicted_classes)
precision = precision_score(y_test, predicted_classes, average='weighted')
recall = recall_score(y_test, predicted_classes, average='weighted')

print(f"Hybrid Model Accuracy: {accuracy}")
print(f"Hybrid Model Precision: {precision}")
print(f"Hybrid Model Recall: {recall}")

```

Hybrid Model Accuracy: 0.5454545454545454  
Hybrid Model Precision: 0.7083333333333334  
Hybrid Model Recall: 0.5454545454545454