Deep Learning Lab Assignment 5: Music Generation Using the MAESTRO Dataset

Q) Develop a deep learning model to generate piano music using the MAESTRO dataset. Load and process MIDI files into piano roll or token-based sequences, split the data, apply normalization, and optionally use data augmentation. Design and train an LSTM, GRU, or Transformer-based model, evaluate its performance, and analyze loss, accuracy, and generated music quality. Finally, discuss challenges, improvements, and overall model effectiveness

Input / Code :

Part 1:

```
#Extract Notes and Piano_rolls from MIDI Files

import os
import numpy as np
import pandas as pd
import pretty_midi
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

# Define the dataset path
dataset_path = "/content/drive/MyDrive/Music_MAESTRO/maestro-v3.0.0/"

# Load metadata CSV file
metadata_file = os.path.join(dataset_path, "maestro-v3.0.0.csv")
df = pd.read_csv(metadata_file)

selected_years = [2017]
df = df[df['year'].isin(selected_years)]

# Get list of MIDI files
midi_files = [os.path.join(dataset_path, f) for f in
df['midi_filename'].tolist()]

# Function to extract note sequences from MIDI
def midi_to_notes(midi_file):
    pm = pretty_midi.PrettyMIDI(midi_file)
    notes = []
```

```python
    for instrument in pm.instruments:
        if instrument.is_drum:
            continue  # Skip drum tracks

        for note in instrument.notes:
            notes.append([note.start, note.end, note.pitch, note.velocity])

    return np.array(notes)

# Process all MIDI files
all_notes = [midi_to_notes(f) for f in midi_files]
all_notes = np.concatenate(all_notes, axis=0)

# Exploratory Data Analysis (EDA) on MIDI Notes
plt.figure(figsize=(10, 4))
sns.histplot(all_notes[:, 2], bins=50, kde=True, color="blue")  # MIDI
pitch distribution
plt.xlabel("MIDI Pitch")
plt.ylabel("Frequency")
plt.title("Pitch Distribution in MAESTRO Dataset")
plt.show()

# Function to convert MIDI to piano roll
def midi_to_piano_roll(midi_file, fs=100):
    pm = pretty_midi.PrettyMIDI(midi_file)
    piano_roll = pm.get_piano_roll(fs=fs)  # fs = time steps per second
    return piano_roll.T  # Transpose to have (time_steps, pitch_classes)

# Convert all MIDI files to piano rolls
piano_rolls = [midi_to_piano_roll(f) for f in midi_files]

# Define a fixed maximum sequence length
max_length = 1000

# Pad or truncate sequences to the same length
piano_rolls = [x[:max_length] if x.shape[0] > max_length else np.pad(x,
((0, max_length - x.shape[0]), (0, 0))) for x in piano_rolls]
```

```
# Convert to NumPy array
X = np.array(piano_rolls)
max = np.max(X)
print("Maximum Midi Velocity: ")
print(max)

# Train-Test-Validation Split
X_train, X_temp = train_test_split(X, test_size=0.2, random_state=42)
X_val, X_test = train_test_split(X_temp, test_size=0.5, random_state=42)

# Normalize input data
X_train = X_train / max  # Normalize MIDI velocities
X_val = X_val / max
X_test = X_test / max

# Print data shapes
print(f"Training Data Shape: {X_train.shape}")
print(f"Validation Data Shape: {X_val.shape}")
print(f"Test Data Shape: {X_test.shape}")
```

Part 2 :

```
#Build CNN+LSTM Model for Training

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, LSTM, Dense,
Dropout, TimeDistributed, BatchNormalization, UpSampling1D

# Define Model
def build_cnn_lstm(input_shape=(1000, 128)):
    model = Sequential()

    # CNN Feature Extraction
```

```
    model.add(Conv1D(64, kernel_size=3, activation='relu', padding='same',
input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=2))

    model.add(Conv1D(128, kernel_size=3, activation='relu',
padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=2))

    # LSTM for Temporal Learning
    model.add(LSTM(128, return_sequences=True))
    model.add(Dropout(0.3))

    model.add(LSTM(64, return_sequences=True))
    model.add(Dropout(0.3))

    # Upsampling to match the original temporal dimension
    model.add(UpSampling1D(size=2))  # Upsample by a factor of 2
    model.add(UpSampling1D(size=2))  # Upsample by a factor of 2

    # Fully Connected Output Layer (Predict Note Activations)
    model.add(TimeDistributed(Dense(128, activation='sigmoid')))

    # Compile Model
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

    return model


# Define input shape (time_steps, pitch_classes)
input_shape = (1000, 128)
model = build_cnn_lstm(input_shape)
model.summary()

# Train Model
history = model.fit(
    X_train, X_train,  # Autoencoder-style training
    epochs=100,
```

```
    batch_size=8,
    validation_data=(X_val, X_val)
)

# Evaluate Model
loss, accuracy = model.evaluate(X_test, X_test)
print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")

# Plot Training History
plt.figure(figsize=(10, 4))

# Loss Curve
plt.subplot(1, 2, 1)
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('Training & Validation Loss')
plt.legend()

# Accuracy Curve
plt.subplot(1, 2, 2)
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('Training & Validation Accuracy')
plt.legend()

plt.show()
```

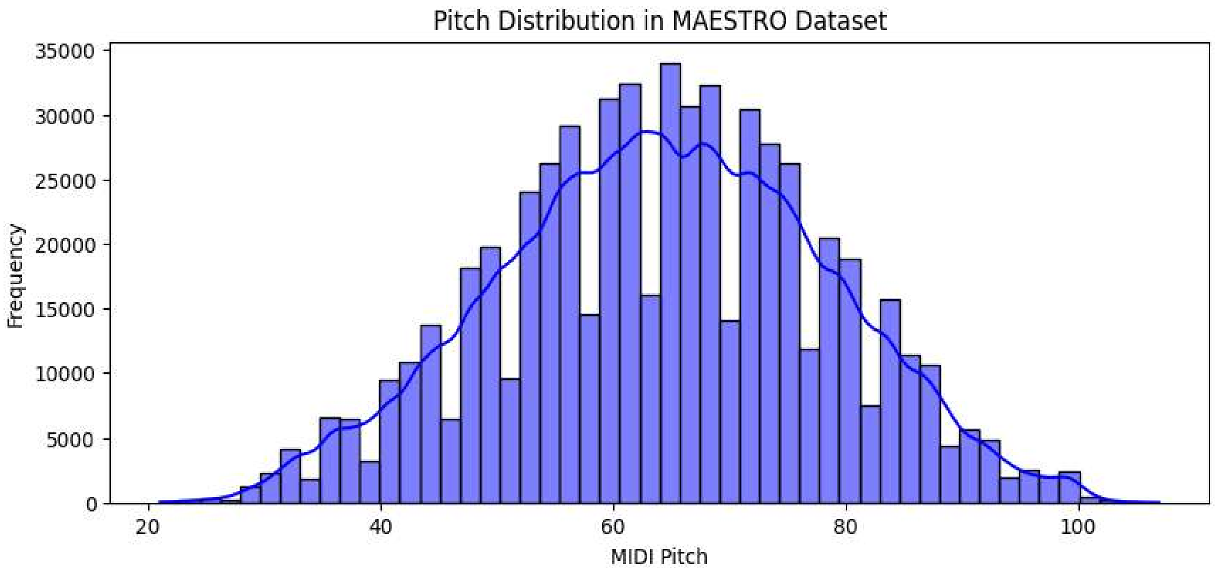
Part 3 :

```
# Generate Music Sequence
generated_sequence = model.predict(X_test[:1])   # Generate for one test sample

print(generated_sequence)
```

```python
# Convert Generated Sequence to MIDI
def piano_roll_to_midi(piano_roll, fs=100):
    pm = pretty_midi.PrettyMIDI()
    instrument = pretty_midi.Instrument(program=0)  # Acoustic Grand Piano

    for time, pitch_vector in enumerate(piano_roll):
        for pitch, velocity in enumerate(pitch_vector):
            if velocity > 0:
                note = pretty_midi.Note(
                    velocity=int(velocity * 127),
                    pitch=pitch,
                    start=time / fs,
                    end=(time + 1) / fs
                )
                instrument.notes.append(note)

    pm.instruments.append(instrument)
    return pm

# Convert and save generated MIDI file
generated_midi = piano_roll_to_midi(generated_sequence[0])
generated_midi.write("generated_music.mid")
print("Generated MIDI saved as 'generated_music.mid'")
```

Output :

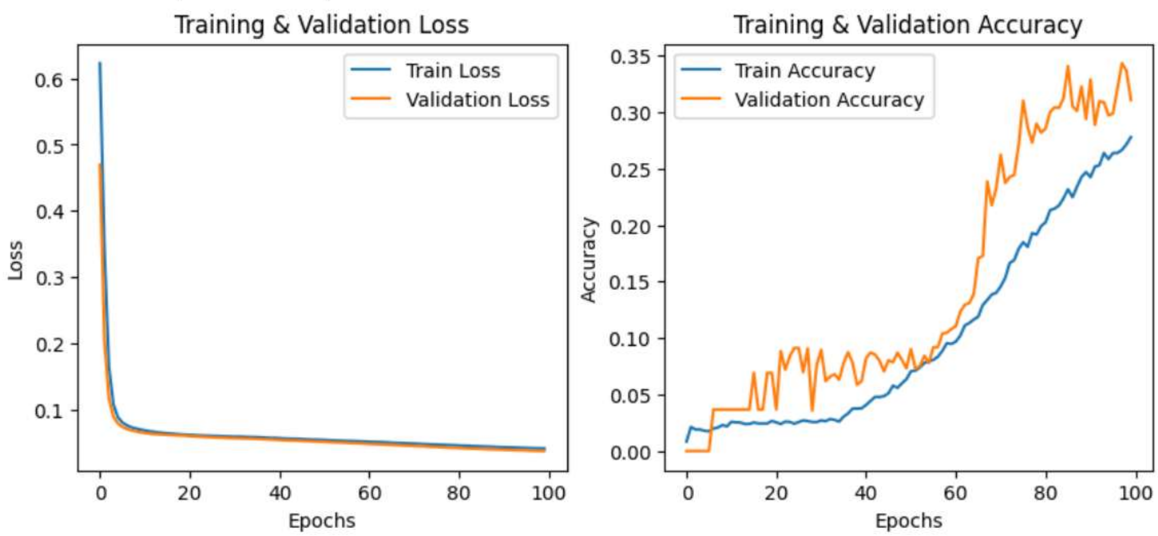
Part 1 (Exploratory Data Analysis) :




```
Maximum Midi Velocity :
114.0
Training Data Shape: (112, 1000, 128)
Validation Data Shape: (14, 1000, 128)
Test Data Shape: (14, 1000, 128)
```

<u>Part 2 (Training and Evaluation) :</u>

```
Epoch 95/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 26s 2s/step - accuracy: 0.2650 - loss: 0.0408 - val_accuracy: 0.2973 - val_loss: 0.0391
Epoch 96/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 43s 2s/step - accuracy: 0.2681 - loss: 0.0422 - val_accuracy: 0.2987 - val_loss: 0.0389
Epoch 97/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 42s 2s/step - accuracy: 0.2691 - loss: 0.0413 - val_accuracy: 0.3207 - val_loss: 0.0385
Epoch 98/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 28s 2s/step - accuracy: 0.2853 - loss: 0.0416 - val_accuracy: 0.3433 - val_loss: 0.0383
Epoch 99/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 28s 2s/step - accuracy: 0.2574 - loss: 0.0443 - val_accuracy: 0.3369 - val_loss: 0.0382
Epoch 100/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 41s 2s/step - accuracy: 0.2535 - loss: 0.0456 - val_accuracy: 0.3108 - val_loss: 0.0381
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 262ms/step - accuracy: 0.3254 - loss: 0.0496
Test Loss: 0.0496, Test Accuracy: 0.3254
```



<u>Part 3 (Generating Music Sequence and Convert to Midi) :</u>

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 9s 9s/step
[[[6.5662316e-04 1.3100143e-03 1.5065094e-03 ... 8.5185369e-04
   6.2444917e-04 2.1217684e-03]
  [6.5662316e-04 1.3100143e-03 1.5065094e-03 ... 8.5185369e-04
   6.2444917e-04 2.1217684e-03]
  [6.5662316e-04 1.3100143e-03 1.5065094e-03 ... 8.5185369e-04
   6.2444917e-04 2.1217684e-03]
  ...
  [2.5919311e-05 7.0540787e-05 8.7663771e-05 ... 3.7851834e-05
   2.6177184e-05 1.4176048e-04]
  [2.5919311e-05 7.0540787e-05 8.7663771e-05 ... 3.7851834e-05
   2.6177184e-05 1.4176048e-04]
  [2.5919311e-05 7.0540787e-05 8.7663771e-05 ... 3.7851834e-05
   2.6177184e-05 1.4176048e-04]]]
Generated MIDI saved as 'generated_music.mid'
```