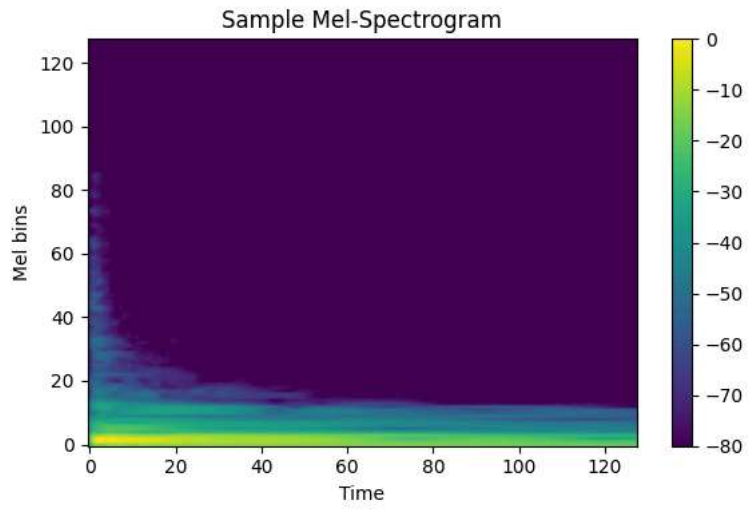
```
# =========================
# 1. Mount Google Drive
# =========================
from google.colab import drive
drive.mount('/content/drive')


# =========================
# 2. Import Required Libraries
# =========================
import numpy as np
import matplotlib.pyplot as plt
import json
import os


# =========================
# 3. Define File Paths
# =========================
BASE_PATH = '/content/drive/MyDrive/processed'

X_PATH = os.path.join(BASE_PATH, 'X.npy')
Y_PATH = os.path.join(BASE_PATH, 'y գմն.npy') #remove this

#Wait remove mistake

X_PATH = os.path.join(BASE_PATH, 'X.npy')
Y_PATH = os.path.join(BASE_PATH, 'y.npy')
LABEL_PATH = os.path.join(BASE_PATH, 'label_map.json')


# =========================
# 4. Verify Files Exist
# =========================
print("Files in processed folder:")
print(os.listdir(BASE_PATH))


# =========================
# 5. Load Data
# =========================
X = np.load(X_PATH)
y = np.load(Y_PATH)

print("\nData loaded successfully")
print("X shape:", X.shape)   # (5600, 128, 128)
print("y shape:", y.shape)   # (5600,)


# =========================
# 6. Load Label Map
# =========================
with open(LABEL_PATH, 'r') as f:
    label_map = json.load(f)

print("\nLabel Map:")
print(label_map)


# =========================
# 7. Visualize One Sample
# =========================
plt.figure(figsize=(6, 4))
plt.imshow(X[0], aspect='auto', origin='lower')
plt.colorbar()
plt.title("Sample Mel-Spectrogram")
plt.xlabel("Time")
plt.ylabel("Mel bins")
plt.tight_layout()
plt.show()
```

```
Mounted at /content/drive
Files in processed folder:
['y (1).npy', 'y.npy', 'label_map (1).json', 'X.npy', 'label_map.json']

Data loaded successfully
X shape: (5227, 128, 128)
y shape: (5227,)

Label Map:
{'bass': 0, 'brass': 1, 'flute': 2, 'guitar': 3, 'keyboard': 4, 'mallet': 5, 'organ': 6, 'reed': 7, 'string': 8}
```



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


# =========================
# 2. Import Libraries
# =========================
import numpy as np
import matplotlib.pyplot as plt
import json
import os


# =========================
# 3. Define Paths
# =========================
BASE_PATH = '/content/drive/MyDrive/processed'

X_PATH = os.path.join(BASE_PATH, 'X.npy')
Y_PATH = os.path.join(BASE_PATH, 'y.npy')
LABEL_PATH = os.path.join(BASE_PATH, 'label_map.json')


# =========================
# 4. Check Files
# =========================
print("Files in processed folder:")
print(os.listdir(BASE_PATH))


# =========================
# 5. Load Data
# =========================
X = np.load(X_PATH)
y = np.load(Y_PATH)

print("\nData loaded successfully")
print("X shape:", X.shape)   # (5600, 128, 128)
print("y shape:", y.shape)   # (5600,)


# =========================
# 6. Load Label Map
# 6. Load Label Map
# =========================
with open(LABEL_PATH, 'r') as f:
    label_map = json.load(f)

print("\nLabel Map:")
print(label_map)


# =========================
# 7. Visualize One Mel-Spectrogram
# =========================
plt.figure(figsize=(6, 4))
plt.imshow(X[0], aspect='auto', origin='lower')
plt.colorbar()
plt.title("Sample Mel-Spectrogram")
plt.xlabel("Time")
plt.ylabel("Mel bins")
plt.tight_layout()
plt.show()


# =========================
# 8. Plot Class Distribution (YOUR REQUEST)
# =========================
unique, counts = np.unique(y, return_counts=True)

plt.figure(figsize=(8, 4))
plt.bar(unique, counts)
plt.xlabel("Class Index")
plt.ylabel("Number of Samples")
plt.title("Class Distribution")
plt.xticks(unique)
plt.tight_layout()
plt.show()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Files in processed folder:
['y (1).npy', 'y.npy', 'label_map (1).json', 'X.npy', 'label_map.json']

Data loaded successfully
X shape: (5227, 128, 128)
y shape: (5227,)

Label Map:
{'bass': 0, 'brass': 1, 'flute': 2, 'guitar': 3, 'keyboard': 4, 'mallet': 5, 'organ': 6, 'reed': 7, 'string': 8}
```

```
# ==========================================================
# STEP 0: VERIFY GPU IS ENABLED
# ==========================================================
import tensorflow as tf

print("TensorFlow version:", tf.__version__)
print("GPU Available:", tf.config.list_physical_devices('GPU'))

# If GPU is enabled, you should see something like:
# [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]


# ==========================================================
# STEP 1: MOUNT GOOGLE DRIVE
# ==========================================================
from google.colab import drive
drive.mount('/content/drive')


# ==========================================================
# STEP 2: IMPORT LIBRARIES
# ==========================================================
import numpy as np
import matplotlib.pyplot as plt
import os

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical


# ==========================================================
# STEP 3: LOAD CLEAN DATA
# ==========================================================
BASE_PATH = '/content/drive/MyDrive/processed'

X = np.load(os.path.join(BASE_PATH, 'X.npy'))
y = np.load(os.path.join(BASE_PATH, 'y.npy'))

print("X shape:", X.shape)
print("y shape:", y.shape)


# ==========================================================
# STEP 4: PREPARE DATA FOR CNN
# ==========================================================

# Add channel dimension (grayscale)
X = X[..., np.newaxis]
print("Reshaped X:", X.shape)

# Encode labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

num_classes = len(np.unique(y_encoded))
y_categorical = to_categorical(y_encoded, num_classes)

# Train / Validation split
X_train, X_val, y_train, y_val = train_test_split(
    X,
    y_categorical,
    test_size=0.2
    random_state=42,
    stratify=y_encoded
)

print("Training samples:", X_train.shape[0])
print("Validation samples:", X_val.shape[0])


# ==========================================================
# STEP 5: BUILD CNN MODEL
# ==========================================================
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=X_train.shape[1:]),
    MaxPooling2D((2,2)),

    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),

    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D((2,2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

model.summary()


# ==========================================================
# STEP 6: TRAIN THE MODEL (GPU ACCELERATED)
# ==========================================================
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    X_train,
    y_train,
    epochs=20,
    batch_size=32,
    validation_data=(X_val, y_val)
)


# ================================== =======================
# STEP 7: EVALUATION
# ==========================================================

# Accuracy & Loss Curves
plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train')
plt.plot(history.history['val_accuracy'], label='Validation')
plt.title('Accuracy vs Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train')
plt.plot(history.history['val_loss'], label='Validation')
plt.title('Loss vs Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()


# Confusion Matrix
y_pred = model.predict(X_val)
y_pred_classes = np.argmax(y_pred, axis=1)
```

```
y_true_classes = np.argmax(y_val, axis=1)

cm = confusion_matrix(y_true_classes, y_pred_classes)
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=label_encoder.classes_
)

disp.plot(cmap='Blues', xticks_rotation=45)
plt.title("Confusion Matrix")
plt.show()


# ==========================================================
# STEP 8: SAVE MODEL
# ==========================================================
MODEL_PATH = '/content/drive/MyDrive/instrunet_model_v1.h5'
model.save(MODEL_PATH)

print("✅ Milestone 2 Completed Successfully")
print("Model saved at:", MODEL_PATH)
```

```
# Get validation accuracy
loss, accuracy = model.evaluate(X_val, y_val, verbose=0)

print(f"Model Validation Accuracy: {accuracy * 100:.2f}%")
```

```
Model Validation Accuracy: 96.75%
```

```
import numpy as np
from collections import Counter

# Convert one-hot labels back to class indices (if needed)
```

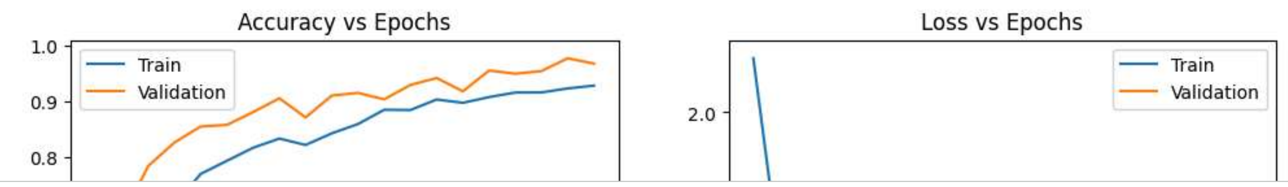
```
y_true = np.argmax(y_val, axis=1)

# Count samples per class
class_counts = Counter(y_true)

# Find the most frequent class
most_common_class, most_common_count = class_counts.most_common(1)[0]

# Calculate baseline accuracy
baseline_accuracy = most_common_count / len(y_true)

print(f"Baseline Accuracy: {baseline_accuracy * 100:.2f}%")
```

```
│ conv2d_3 (Conv2D)                │ (None, 126, 126, 32)      │           320 │
Baseline Accuracy: 13.38%
│ max_pooling2d_3 (MaxPooling2D)   │ (None, 63, 63, 32)        │             0 │
```

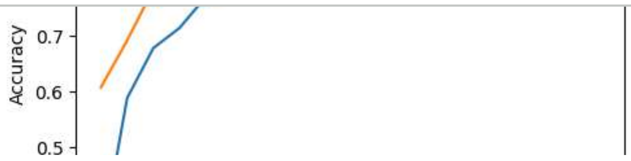
```
from sklearn.metrics import classification_report, accuracy_score

# Predict
y_pred = model.predict(X_val)

# Convert one-hot to labels
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_val, axis=1)

# Accuracy
acc = accuracy_score(y_true_classes, y_pred_classes)
print(f"Overall Accuracy: {acc * 100:.2f}%\n")

# Generate class names safely
num_classes = len(np.unique(y_true_classes))
class_names = [f"Class_{i}" for i in range(num_classes)]

# Classification report (NO ERROR)
report = classification_report(
    y_true_classes,
    y_pred_classes,
    target_names=class_names,
    zero_division=0
)

print("Classification Report:")
print(report)
```

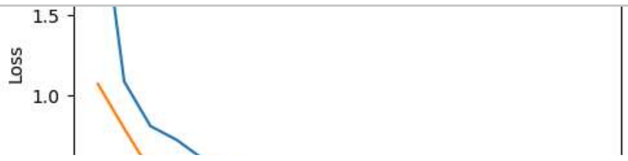
```
33/33 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step
Overall Accuracy: 11.85%

Classification Report:
              precision    recall  f1-score   support

     Class_0       0.00      0.00      0.00        35
     Class_1       1.00      0.04      0.08       140
     Class_2       0.00      0.00      0.00       140
     Class_3       0.00      0.00      0.00       140
     Class_4       0.00      0.00      0.00       140
     Class_5       0.00      0.00      0.00       140
     Class_6       0.03      0.10      0.04        31
     Class_7       0.12      0.82      0.22       140
     Class_8       0.00      0.00      0.00       140

    accuracy                           0.12      1046
   macro avg       0.13      0.11      0.04      1046
weighted avg       0.15      0.12      0.04      1046
```

```
131/131 ━━━━━━━━━━━━━━━━━━━━ 148s 1s/step - accuracy: 0.8051 - loss: 0.4893 - val_accuracy: 0.8805 - val_loss: 0.3311
Epoch 8/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 132s 1s/step - accuracy: 0.8406 - loss: 0.4032 - val_accuracy: 0.9054 - val_loss: 0.2585
Epoch 9/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.8357 - loss: 0.4524 - val_accuracy: 0.8709 - val_loss: 0.3009
Epoch 10/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 131s 1000ms/step - accuracy: 0.8379 - loss: 0.4237 - val_accuracy: 0.9101 - val_loss: 0.2256
Epoch 11/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 148s 1s/step - accuracy: 0.8535 - loss: 0.3631 - val_accuracy: 0.9149 - val_loss: 0.2167
Epoch 12/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 133s 1s/step - accuracy: 0.8959 - loss: 0.2676 - val_accuracy: 0.9034 - val_loss: 0.2507
Epoch 13/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 134s 1s/step - accuracy: 0.8710 - loss: 0.3272 - val_accuracy: 0.9293 - val_loss: 0.1594
Epoch 14/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 143s 1s/step - accuracy: 0.9077 - loss: 0.2579 - val_accuracy: 0.9417 - val_loss: 0.1551
Epoch 15/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 133s 981ms/step - accuracy: 0.8948 - loss: 0.2730 - val_accuracy: 0.9178 - val_loss: 0.2013
Epoch 16/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 130s 995ms/step - accuracy: 0.9120 - loss: 0.2290 - val_accuracy: 0.9551 - val_loss: 0.1183
Epoch 17/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 129s 990ms/step - accuracy: 0.9159 - loss: 0.1989 - val_accuracy: 0.9493 - val_loss: 0.1350
Epoch 18/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 142s 991ms/step - accuracy: 0.9250 - loss: 0.1909 - val_accuracy: 0.9541 - val_loss: 0.1225
Epoch 19/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 139s 970ms/step - accuracy: 0.9223 - loss: 0.2079 - val_accuracy: 0.9771 - val_loss: 0.0706
Epoch 20/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 146s 997ms/step - accuracy: 0.9308 - loss: 0.1817 - val_accuracy: 0.9675 - val_loss: 0.0760
```



]


Accuracy
0.7
0.6
0.5


Loss
1.5
1.0