STEP 0 — Install & import libraries

```
!pip install kagglehub librosa scikit-learn matplotlib
```

```
Requirement already satisfied: kagglehub in /usr/local/lib/python3.12/dist-packages (0.3.13)
Requirement already satisfied: librosa in /usr/local/lib/python3.12/dist-packages (0.11.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from kagglehub) (25.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.12/dist-packages (from kagglehub) (6.0.3)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from kagglehub) (2.32.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from kagglehub) (4.67.1)
Requirement already satisfied: audioread>=2.1.9 in /usr/local/lib/python3.12/dist-packages (from librosa) (3.1.0)
Requirement already satisfied: numba>=0.51.0 in /usr/local/lib/python3.12/dist-packages (from librosa) (0.60.0)
Requirement already satisfied: numpy>=1.22.3 in /usr/local/lib/python3.12/dist-packages (from librosa) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from librosa) (1.16.3)
Requirement already satisfied: joblib>=1.0 in /usr/local/lib/python3.12/dist-packages (from librosa) (1.5.3)
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.12/dist-packages (from librosa) (4.4.2)
Requirement already satisfied: soundfile>=0.12.1 in /usr/local/lib/python3.12/dist-packages (from librosa) (0.13.1)
Requirement already satisfied: pooch>=1.1 in /usr/local/lib/python3.12/dist-packages (from librosa) (1.8.2)
Requirement already satisfied: soxr>=0.3.2 in /usr/local/lib/python3.12/dist-packages (from librosa) (1.0.0)
Requirement already satisfied: typing_extensions>=4.1.1 in /usr/local/lib/python3.12/dist-packages (from librosa) (4.15.0)
Requirement already satisfied: lazy_loader>=0.1 in /usr/local/lib/python3.12/dist-packages (from librosa) (0.4)
Requirement already satisfied: msgpack>=1.0 in /usr/local/lib/python3.12/dist-packages (from librosa) (1.1.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.61.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.3.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.12/dist-packages (from numba>=0.51.0->librosa) (0.43.0)
Requirement already satisfied: platformdirs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (from pooch>=1.1->librosa) (4.5.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests->kagglehub) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->kagglehub) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->kagglehub) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->kagglehub) (2026.1.4)
Requirement already satisfied: cffi>=1.0 in /usr/local/lib/python3.12/dist-packages (from soundfile>=0.12.1->librosa) (2.0.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.12/dist-packages (from cffi>=1.0->soundfile>=0.12.1->librosa) (2.23)
```

```
import os
import numpy as np
import librosa
import matplotlib.pyplot as plt

import kagglehub

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA

from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Input, Conv1D, AveragePooling1D,
```

```
    Dropout, BatchNormalization,
    GlobalAveragePooling1D, Dense
)
from tensorflow.keras.utils import to_categorical
```

STEP 1 — Download GTZAN using kagglehub

```
import kagglehub

path = kagglehub.dataset_download(
    "andradaolteanu/gtzan-dataset-music-genre-classification"
)

print("Dataset downloaded to:", path)
```

```
Using Colab cache for faster access to the 'gtzan-dataset-music-genre-classification' dataset.
Dataset downloaded to: /kaggle/input/gtzan-dataset-music-genre-classification
```

STEP 2 — Define dataset path & genres (paper-faithful)

```
DATASET_PATH = os.path.join(path, "Data", "genres_original")

GENRES = ['classical', 'blues', 'pop', 'hiphop', 'metal']
N_MFCC = 20
MAX_LEN = 1290
```

```
GENRES = ['classical', 'blues', 'pop', 'hiphop', 'metal']
```

STEP 3 — MFCC extraction

```
def extract_mfcc(file_path):
    y, sr = librosa.load(file_path, duration=30)
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=N_MFCC)

    if mfcc.shape[1] < MAX_LEN:
        mfcc = np.pad(mfcc, ((0,0),(0, MAX_LEN - mfcc.shape[1])))
    else:
        mfcc = mfcc[:, :MAX_LEN]

    return mfcc.T  # (time, mfcc)
```

STEP 4 — Build dataset (500 samples)

```
X, y = [], []

for label, genre in enumerate(GENRES):
    genre_path = os.path.join(DATASET_PATH, genre)
```

```
        for file in os.listdir(genre_path):
            if file.endswith(".wav"):
                X.append(extract_mfcc(os.path.join(genre_path, file)))
                y.append(label)

X = np.array(X)   # (500, 1290, 20)
y = np.array(y)

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

```
X shape: (500, 1290, 20)
```

STEP 5 — Train / Test split (80–20)

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

y_train_cat = to_categorical(y_train, 5)
y_test_cat  = to_categorical(y_test, 5)
```

STEP 6 — Build Dilated CNN (feature extractor design)

This CNN is designed to output exactly 48 features (like the paper)

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Input, Conv1D, AveragePooling1D,
    Dropout, BatchNormalization,
    Flatten, Dense
)

input_layer = Input(shape=(1290, 20))

# -------- Block 1 --------
x = Conv1D(16, 8, dilation_rate=4, activation='relu')(input_layer)
x = AveragePooling1D(32)(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)

# -------- Block 2 (feature layer) --------
x = Conv1D(16, 4, dilation_rate=2, activation='relu', name="conv2")(x)

# IMPORTANT: pooling chosen so time dimension → 3
x = AveragePooling1D(pool_size=11)(x)   # ≈ 33 → 3
x = Dropout(0.5)(x)
x = BatchNormalization()(x)

# -------- Flatten → 48 features --------
x = Flatten()(x)   # 3 × 16 = 48

output = Dense(5, activation='softmax')(x)

cnn_model = Model(input_layer, output)
```

```
cnn_model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

cnn_model.summary()
```

Model: "functional_14"

Layer (type)	Output Shape	Param #
input_layer_5 (InputLayer)	(None, 1290, 20)	0
conv1d_5 (Conv1D)	(None, 1262, 16)	2,576
average_pooling1d_8 (AveragePooling1D)	(None, 39, 16)	0
dropout_8 (Dropout)	(None, 39, 16)	0
batch_normalization_8 (BatchNormalization)	(None, 39, 16)	64
conv2 (Conv1D)	(None, 33, 16)	1,040
average_pooling1d_9 (AveragePooling1D)	(None, 3, 16)	0
dropout_9 (Dropout)	(None, 3, 16)	0
batch_normalization_9 (BatchNormalization)	(None, 3, 16)	64
flatten_1 (Flatten)	(None, 48)	0
dense_3 (Dense)	(None, 5)	245

Total params: 3,989 (15.58 KB)

STEP 7 — Train the CNN

```
cnn_model.fit(
    X_train, y_train_cat,
    epochs=40,
    batch_size=16,
    validation_data=(X_test, y_test_cat),
    verbose=1
)
```

```
Epoch 16/40
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.7779 - loss: 0.5385 - val_accuracy: 0.8100 - val_loss: 0.4925
Epoch 17/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.7861 - loss: 0.5912 - val_accuracy: 0.8500 - val_loss: 0.4798
Epoch 18/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.8070 - loss: 0.5788 - val_accuracy: 0.7900 - val_loss: 0.5565
Epoch 19/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.8170 - loss: 0.5832 - val_accuracy: 0.8600 - val_loss: 0.4190
Epoch 20/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8069 - loss: 0.5541 - val_accuracy: 0.8600 - val_loss: 0.3974
Epoch 21/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.8246 - loss: 0.5412 - val_accuracy: 0.8800 - val_loss: 0.4058
Epoch 22/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.8427 - loss: 0.5053 - val_accuracy: 0.8200 - val_loss: 0.4779
Epoch 23/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.7903 - loss: 0.4985 - val_accuracy: 0.8400 - val_loss: 0.4418
Epoch 24/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.7572 - loss: 0.6160 - val_accuracy: 0.8700 - val_loss: 0.4564
Epoch 25/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.8109 - loss: 0.5368 - val_accuracy: 0.8900 - val_loss: 0.4249
Epoch 26/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.8561 - loss: 0.4040 - val_accuracy: 0.8700 - val_loss: 0.3897
Epoch 27/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.8279 - loss: 0.4873 - val_accuracy: 0.8400 - val_loss: 0.4383
Epoch 28/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.8224 - loss: 0.5195 - val_accuracy: 0.8500 - val_loss: 0.4091
Epoch 29/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8638 - loss: 0.4092 - val_accuracy: 0.8800 - val_loss: 0.3959
Epoch 30/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8477 - loss: 0.4459 - val_accuracy: 0.8800 - val_loss: 0.3739
Epoch 31/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8797 - loss: 0.3669 - val_accuracy: 0.8500 - val_loss: 0.3917
Epoch 32/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8722 - loss: 0.4158 - val_accuracy: 0.8800 - val_loss: 0.3668
Epoch 33/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8894 - loss: 0.3231 - val_accuracy: 0.8600 - val_loss: 0.4074
Epoch 34/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8126 - loss: 0.5079 - val_accuracy: 0.8700 - val_loss: 0.3719
Epoch 35/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8725 - loss: 0.4083 - val_accuracy: 0.8800 - val_loss: 0.3742
Epoch 36/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8610 - loss: 0.4326 - val_accuracy: 0.8900 - val_loss: 0.3880
Epoch 37/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.8450 - loss: 0.4724 - val_accuracy: 0.8500 - val_loss: 0.3918
Epoch 38/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8468 - loss: 0.4248 - val_accuracy: 0.7400 - val_loss: 0.6948
Epoch 39/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8493 - loss: 0.4786 - val_accuracy: 0.8700 - val_loss: 0.3921
Epoch 40/40
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.8533 - loss: 0.3884 - val_accuracy: 0.8800 - val_loss: 0.3658
```

STEP 8 — Classical baselines (Raw MFCC)

```
X_raw = X.reshape(X.shape[0], -1)

Xr_train, Xr_test, yr_train, yr_test = train_test_split(
    X_raw, y, test_size=0.2, stratify=y, random_state=42
)

lr_raw = LogisticRegression(max_iter=1000)
svm_raw = SVC(kernel='rbf')
rf_raw = RandomForestClassifier(n_estimators=200, max_depth=7)
```

```
lr_raw.fit(Xr_train, yr_train)
svm_raw.fit(Xr_train, yr_train)
rf_raw.fit(Xr_train, yr_train)

print("RAW MFCC")
print("LR :", lr_raw.score(Xr_test, yr_test))
print("SVM:", svm_raw.score(Xr_test, yr_test))
print("RF :", rf_raw.score(Xr_test, yr_test))
```

```
RAW MFCC
LR : 0.77
SVM: 0.77
RF : 0.78
```

STEP 9 — PCA baseline (paper comparison)

```
pca = PCA(n_components=50)
X_pca = pca.fit_transform(X_raw)

Xp_train, Xp_test, yp_train, yp_test = train_test_split(
    X_pca, y, test_size=0.2, stratify=y, random_state=42
)

lr_pca = LogisticRegression(max_iter=1000)
svm_pca = SVC(kernel='rbf')
rf_pca = RandomForestClassifier(n_estimators=200, max_depth=7)

lr_pca.fit(Xp_train, yp_train)
svm_pca.fit(Xp_train, yp_train)
rf_pca.fit(Xp_train, yp_train)

print("PCA MFCC")
print("LR :", lr_pca.score(Xp_test, yp_test))
print("SVM:", svm_pca.score(Xp_test, yp_test))
print("RF :", rf_pca.score(Xp_test, yp_test))
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
PCA MFCC
LR : 0.71
SVM: 0.82
RF : 0.77
```

STEP 10 — CNN feature extraction (CORE THEORY)

```
# STEP 10 — CNN feature extraction (paper-faithful)

feature_model = Model(
```

```
    inputs=cnn_model.input,
    outputs=cnn_model.get_layer("average_pooling1d_9").output
)

X_train_feat = feature_model.predict(X_train)
X_test_feat  = feature_model.predict(X_test)

# Flatten pooled CNN features
X_train_feat = X_train_feat.reshape(X_train_feat.shape[0], -1)
X_test_feat  = X_test_feat.reshape(X_test_feat.shape[0], -1)

print(X_train_feat.shape)
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
(400, 48)
```

```
X_train_feat = X_train_feat.reshape(X_train_feat.shape[0], -1)
X_test_feat  = X_test_feat.reshape(X_test_feat.shape[0], -1)
```

STEP 11 — Classical ML on CNN features (FINAL RESULT)

```
# STEP 11 — Classical ML on CNN features (paper-faithful)

feature_model = Model(
    inputs=cnn_model.input,
    outputs=cnn_model.get_layer("average_pooling1d_9").output
)

X_train_feat = feature_model.predict(X_train)
X_test_feat  = feature_model.predict(X_test)

# Flatten pooled CNN features → 48-D
X_train_feat = X_train_feat.reshape(X_train_feat.shape[0], -1)
X_test_feat  = X_test_feat.reshape(X_test_feat.shape[0], -1)

print("CNN feature shape:", X_train_feat.shape)
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
CNN feature shape: (400, 48)
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

# ----- Logistic Regression with scaling -----
lr = Pipeline([
```

```
    ("scaler", StandardScaler()),
    ("lr", LogisticRegression(max_iter=3000))
])

# ----- SVM with scaling -----
svm = Pipeline([
    ("scaler", StandardScaler()),
    ("svm", SVC(kernel='rbf'))
])

# ----- Random Forest (NO scaling) -----
rf = RandomForestClassifier(
    n_estimators=300,
    max_depth=None,
    random_state=42
)

# ----- Train models -----
lr.fit(X_train_feat, y_train)
svm.fit(X_train_feat, y_train)
rf.fit(X_train_feat, y_train)

# ----- Evaluate -----
print("CNN + LR :", lr.score(X_test_feat, y_test))
print("CNN + SVM:", svm.score(X_test_feat, y_test))
print("CNN + RF :", rf.score(X_test_feat, y_test))
```

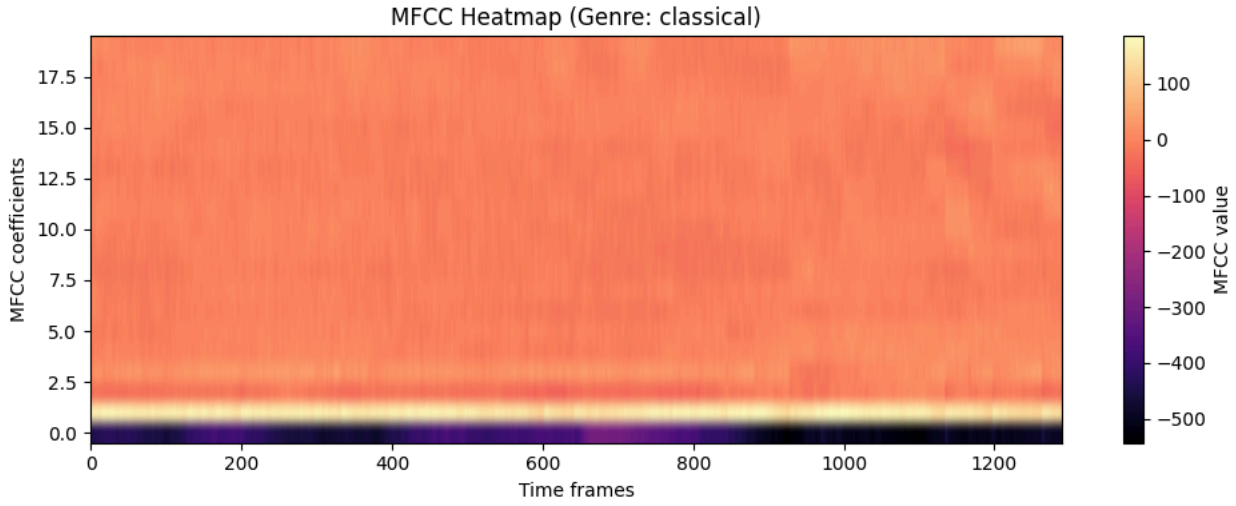
```
CNN + LR : 0.87
CNN + SVM: 0.87
CNN + RF : 0.88
```

Implement MFCC heatmap

```
import matplotlib.pyplot as plt

sample_idx = 0   # try different indices
mfcc_sample = X[sample_idx]   # shape: (1290, 20)
```

Plot MFCC heatmap

```
plt.figure(figsize=(10, 4))
plt.imshow(mfcc_sample.T, aspect='auto', origin='lower', cmap='magma')
plt.colorbar(label='MFCC value')
plt.xlabel("Time frames")
plt.ylabel("MFCC coefficients")
plt.title(f"MFCC Heatmap (Genre: {GENRES[y[sample_idx]]})")
plt.tight_layout()
plt.show()
```



Heatmap type 2: CNN feature-map heatmap (Conv layer activations)

Step 1: Build a model that outputs conv2

```
from tensorflow.keras.models import Model

activation_model = Model(
    inputs=cnn_model.input,
    outputs=cnn_model.get_layer("conv2").output
)
```

Step 2: Get activations for one sample

```
sample_idx = 0
activation = activation_model.predict(X[sample_idx:sample_idx+1])

print(activation.shape)
```

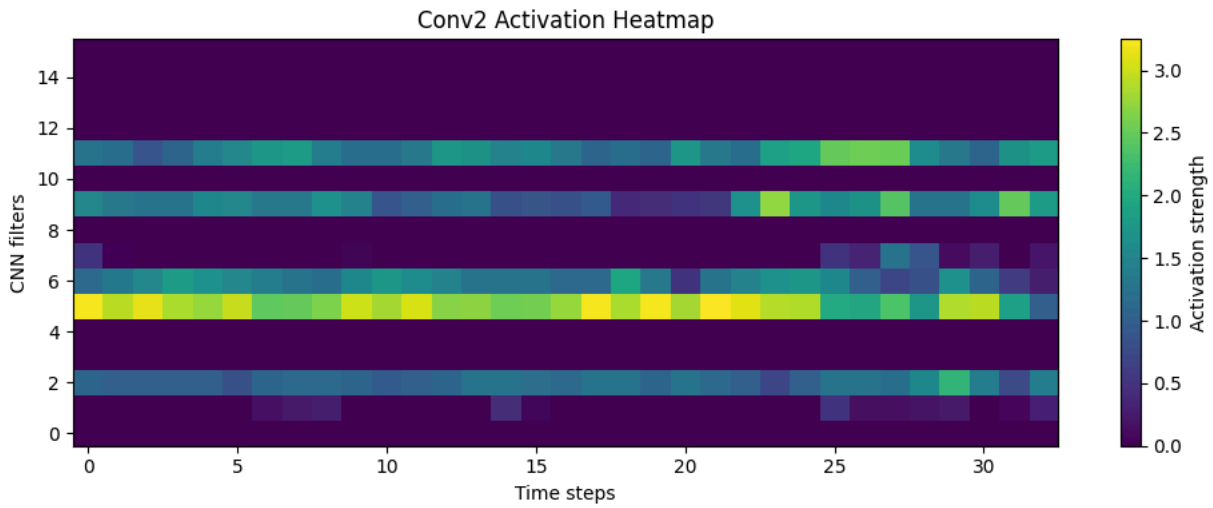
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 123ms/step
(1, 33, 16)
```

```
activation = activation[0]  # shape: (T, C)
```

Step 3: Plot CNN activation heatmap

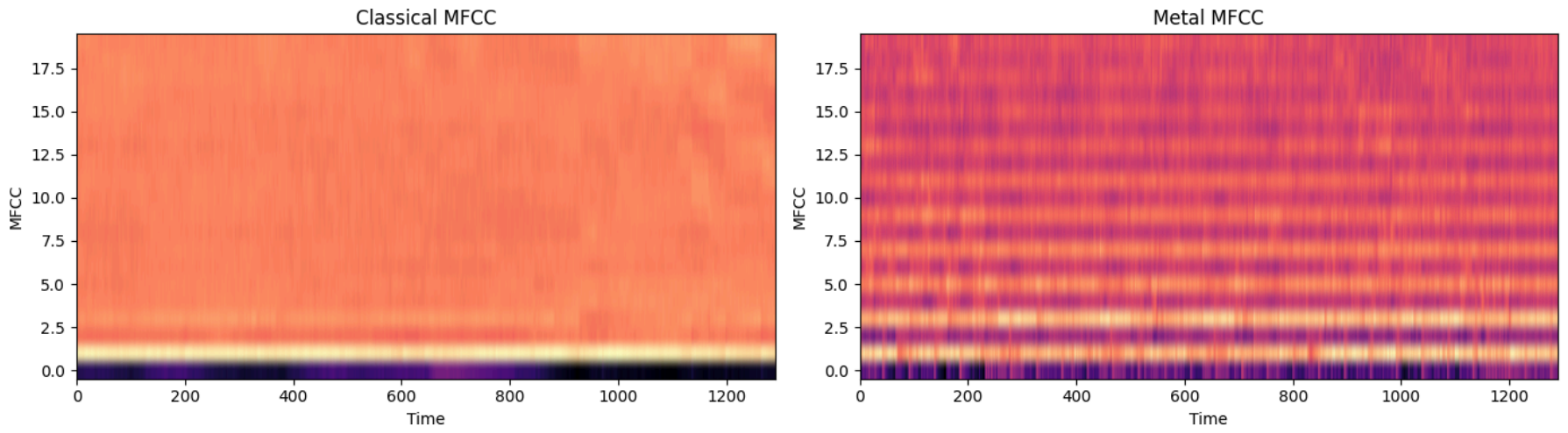
```
plt.figure(figsize=(10, 4))
plt.imshow(activation.T, aspect='auto', origin='lower', cmap='viridis')
plt.colorbar(label='Activation strength')
plt.xlabel("Time steps")
```

```
plt.ylabel("CNN filters")
plt.title("Conv2 Activation Heatmap")
plt.tight_layout()
plt.show()
```



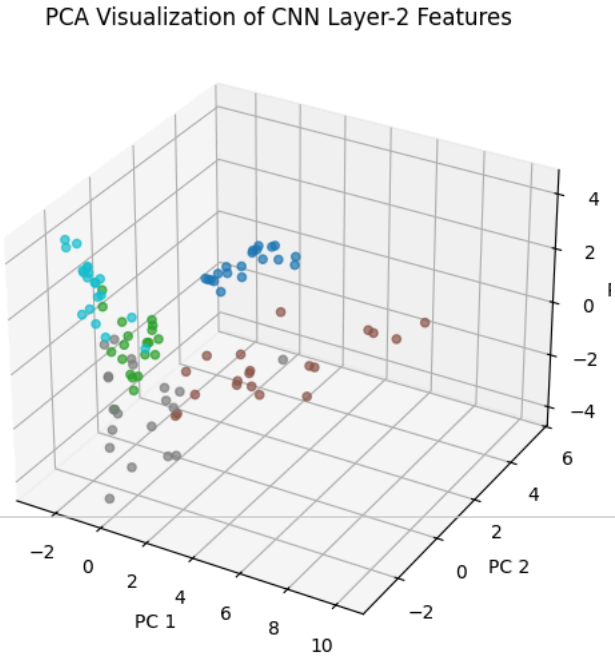
Side-by-side genre comparison

```
fig, axes = plt.subplots(1, 2, figsize=(14, 4))

# Classical
idx_c = y.tolist().index(GENRES.index("classical"))
axes[0].imshow(X[idx_c].T, aspect='auto', origin='lower', cmap='magma')
axes[0].set_title("Classical MFCC")

# Metal
idx_m = y.tolist().index(GENRES.index("metal"))
axes[1].imshow(X[idx_m].T, aspect='auto', origin='lower', cmap='magma')
axes[1].set_title("Metal MFCC")

for ax in axes:
    ax.set_xlabel("Time")
    ax.set_ylabel("MFCC")

plt.tight_layout()
plt.show()
```



PCA Visualization

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

pca = PCA(n_components=3)
X_3d = pca.fit_transform(X_test_feat)

fig = plt.figure(figsize=(7, 6))
ax = fig.add_subplot(111, projection='3d')

scatter = ax.scatter(
    X_3d[:, 0],
    X_3d[:, 1],
    X_3d[:, 2],
    c=y_test,
    cmap='tab10',
    alpha=0.7
)

ax.set_title("PCA Visualization of CNN Layer-2 Features")
ax.set_xlabel("PC 1")
ax.set_ylabel("PC 2")
ax.set_zlabel("PC 3")

plt.show()
```


PCA Visualization of CNN Layer-2 Features
PC 1
PC 2
−2
0
2
4
6
8
10
−2
0
2
4
6
4
2
0
−2
−4