NAME: VIDYA SINHA
ROLL no: 92200133021

FEATURE EXTRACTION

1. Image Feature Extraction

Methods Used:

- **Color Histogram**: Extracts the distribution of colors in an image.
- **Edge Detection (Canny)**: Detects edges or boundaries in an image.
- **Histogram of Oriented Gradients (HOG)**: Captures object structure and outlines.
- **Texture Features (LBP)**: Extracts patterns based on texture.
- **Image Moments**: Captures shape-based features.

Libraries:

- **OpenCV**: Used for color histograms and edge detection. It's fast and widely used for image processing.
- **scikit-image**: Used for HOG and LBP. It provides tools for advanced image analysis.
- **NumPy**: Used for calculating basic statistical properties like moments.

How it Worked:

- OpenCV made it easy to preprocess the images (grayscale, resizing) and extract features like edges and color distributions.
- scikit-image provided built-in functions like hog() and local_binary_pattern() for texture and shape analysis, which worked seamlessly on various test images.
- The results were numerical arrays, which could be visualized or fed into machine learning models for classification.

Image Taken for example:

NAME: VIDYA SINHA
ROLL no: 92200133021

CODE:

```
import cv2

import numpy as np

import matplotlib.pyplot as plt


# Read image

image = cv2.imread('image.jpg')

image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)


# Calculate histogram for each channel (Red, Green, Blue)

r_hist = cv2.calcHist([image], [0], None, [256], [0, 256])

g_hist = cv2.calcHist([image], [1], None, [256], [0, 256])

b_hist = cv2.calcHist([image], [2], None, [256], [0, 256])


# Plot histograms

plt.plot(r_hist, color='red')

plt.plot(g_hist, color='green')

plt.plot(b_hist, color='blue')
```

NAME: VIDYA SINHA
ROLL no: 92200133021

```python
plt.title("Color Histogram")

plt.show()




# Read image

image = cv2.imread('image.jpg', 0)  # Grayscale image


# Apply Canny edge detection

edges = cv2.Canny(image, 100, 200)


# Display the result

plt.imshow(edges, cmap='gray')

plt.title('Edge Detection')

plt.show()




from skimage.feature import local_binary_pattern

from skimage import img_as_ubyte


# Read image and convert to grayscale

image = cv2.imread('image.jpg', 0)


# Apply Local Binary Pattern (LBP)

radius = 1

n_points = 8 * radius

lbp = local_binary_pattern(image, n_points, radius, method='uniform')
```

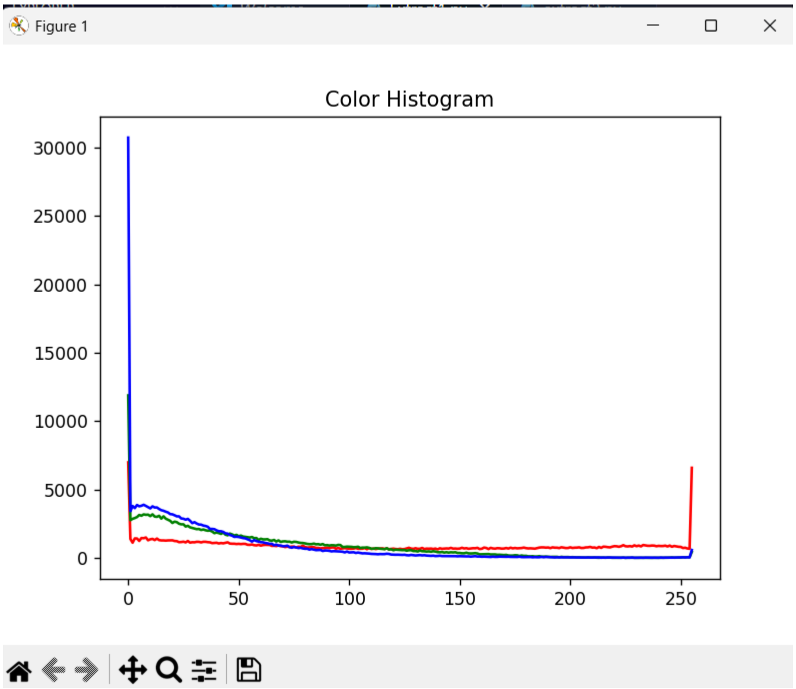
NAME: VIDYA SINHA
ROLL no: 92200133021

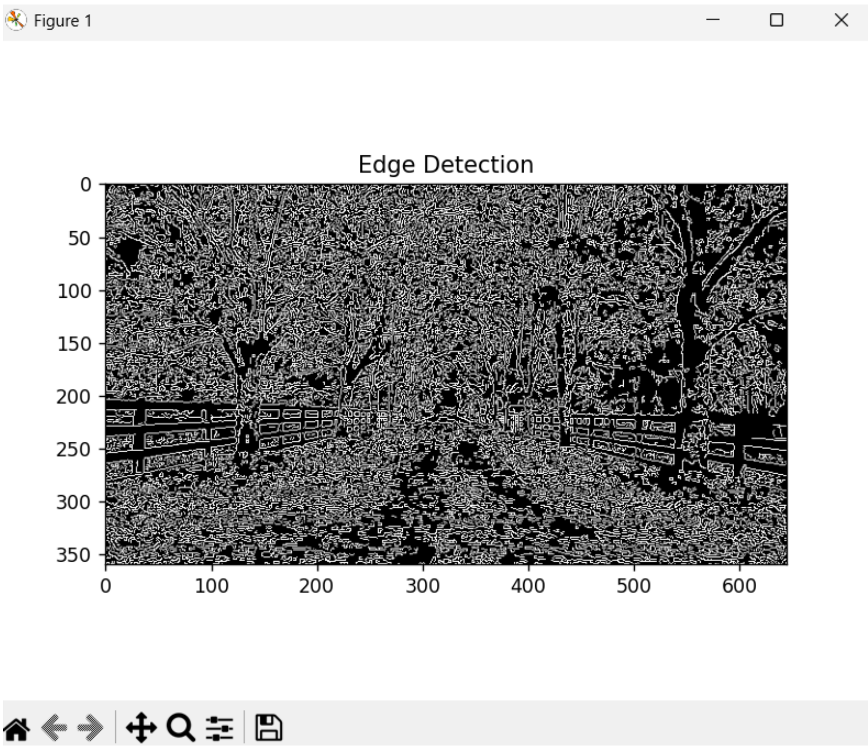
```
# Display LBP
plt.imshow(lbp, cmap='gray')
plt.title('Local Binary Pattern (Texture)')
plt.show()
```

```
from skimage.feature import hog
from skimage import exposure

# Read image and convert to grayscale
image = cv2.imread('image.jpg', 0)

# Compute HOG features
features, hog_image = hog(image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=True)

# Enhance the HOG image for better visualization
hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))

# Display the HOG image
plt.imshow(hog_image_rescaled, cmap='gray')
plt.title('Histogram of Oriented Gradients')
plt.show()
```

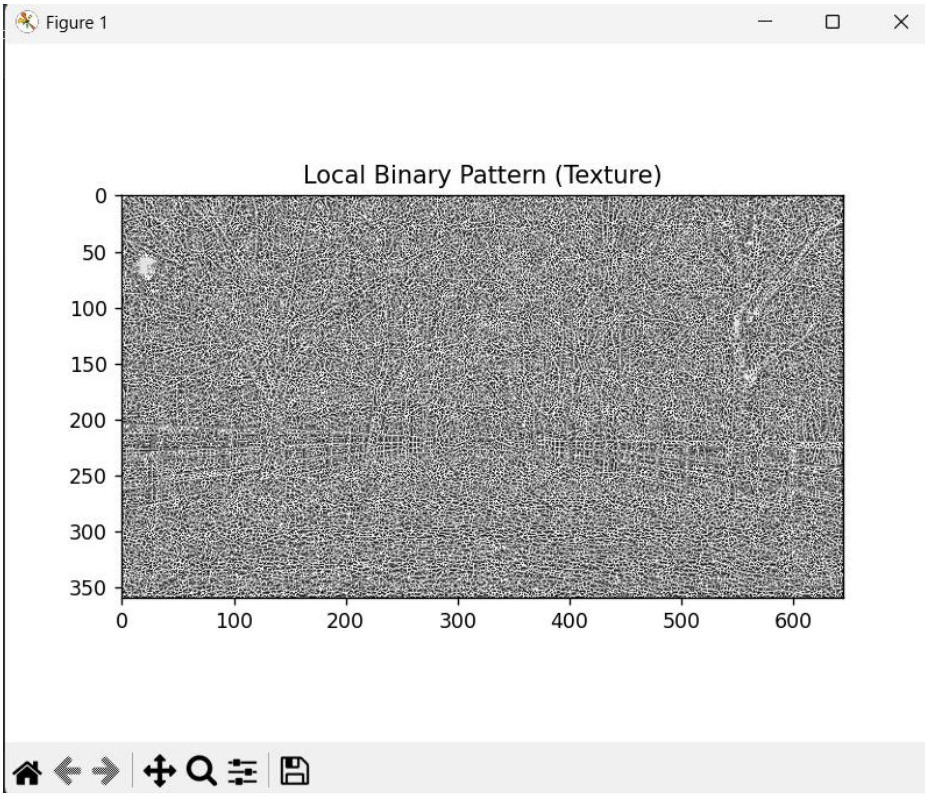
NAME: VIDYA SINHA
ROLL no: 92200133021

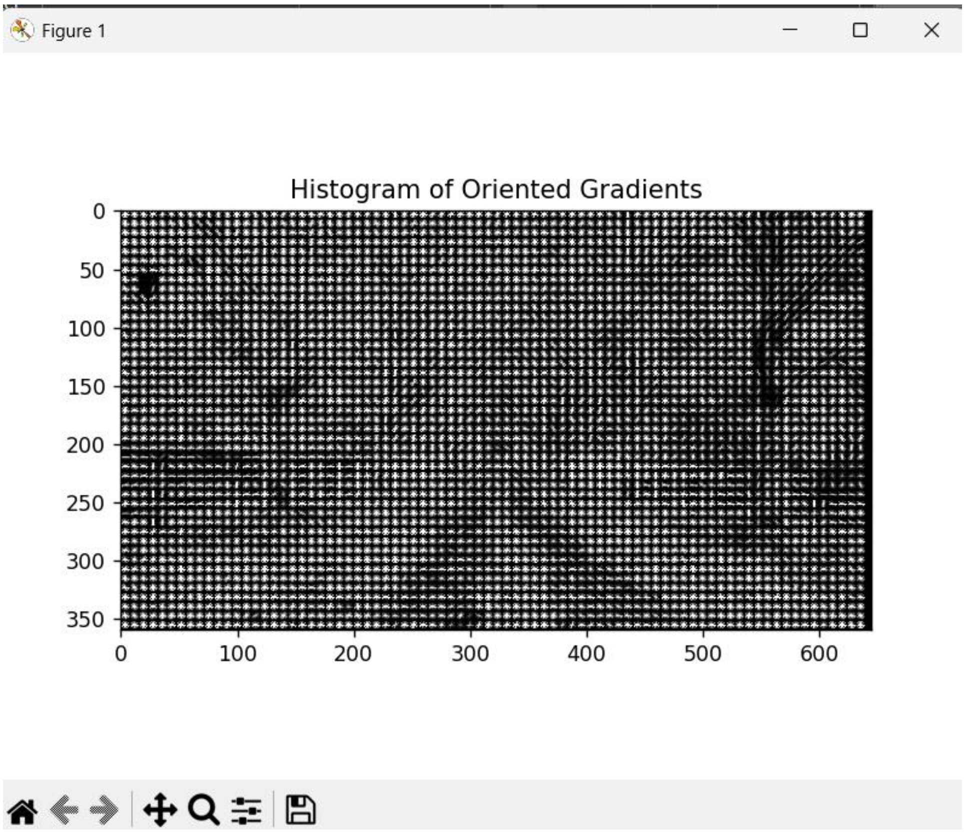
OUTPUT:





NAME: VIDYA SINHA
ROLL no: 92200133021





NAME: VIDYA SINHA
ROLL no: 92200133021

3. Audio Feature Extraction

Methods Used:

- **MFCCs (Mel-Frequency Cepstral Coefficients)**: Extracts frequency domain features for speech/music analysis.
- **Spectrogram**: Visual representation of frequencies over time.
- **Zero Crossing Rate**: Measures how often the signal changes sign, useful for rhythmic analysis.
- **Chroma Features**: Represents pitches in music.

Libraries:

- **Librosa**: The go-to library for audio processing, supporting MFCCs, spectrograms, chroma features, and more.
- **Matplotlib**: Used to visualize spectrograms and feature distributions.

How it Worked:

- Librosa was straightforward for computing advanced audio features like MFCCs and generating spectrograms.
- Visualization tools helped understand the features extracted from test audio files, like music or speech recordings.
- Audio with clear tonal or rhythmic components gave the most meaningful results.

CODE:

```
import librosa

import librosa.display

import matplotlib.pyplot as plt


# Load audio file

audio_file = 'audio.wav'

y, sr = librosa.load(audio_file)


# Compute MFCC

mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
```

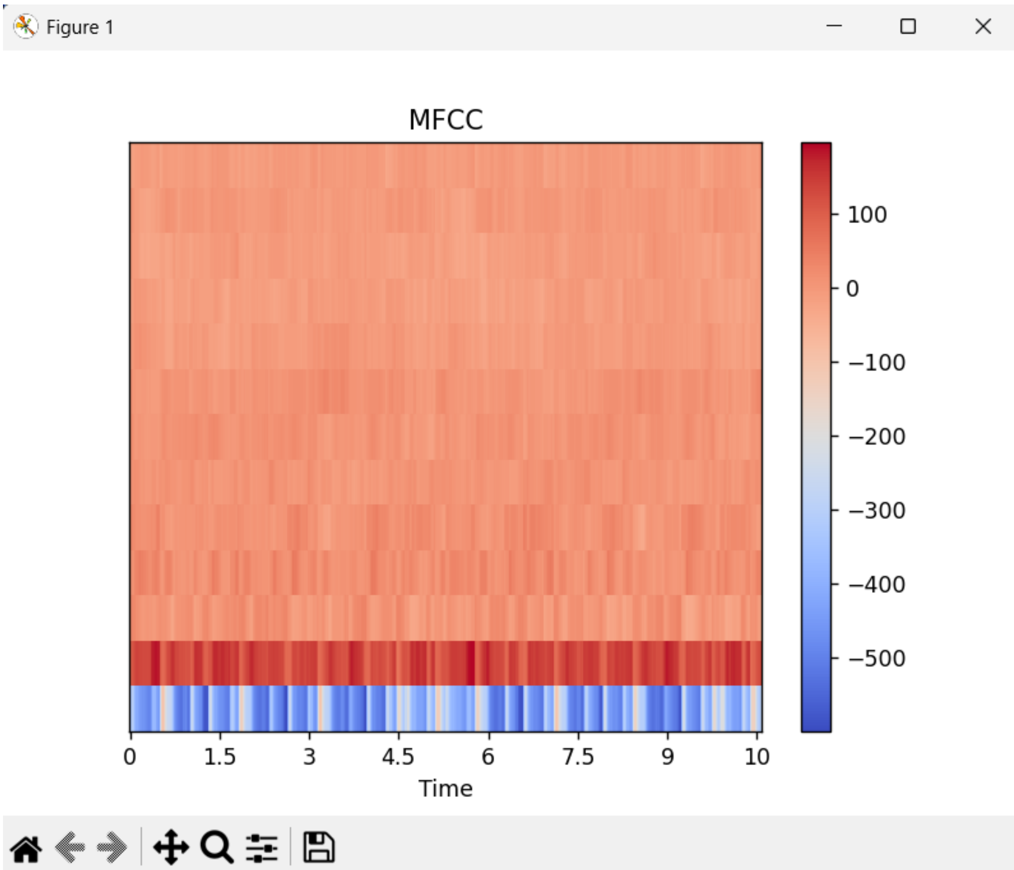
NAME: VIDYA SINHA
ROLL no: 92200133021

```
# Display MFCC
librosa.display.specshow(mfcc, x_axis='time')
plt.colorbar()
plt.title('MFCC')
plt.show()



import numpy as np
import librosa.display


# Load audio file
y, sr = librosa.load('audio.wav')
# Compute Zero Crossing Rate
zcr = librosa.feature.zero_crossing_rate(y)


# Plot Zero Crossing Rate
plt.plot(zcr.T)
plt.xlabel('Frames')
plt.ylabel('Zero Crossing Rate')
plt.title('Zero Crossing Rate')
plt.show()
# Compute the spectrogram
D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)


# Display spectrogram
librosa.display.specshow(D, x_axis='time', y_axis='log')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogram')
plt.show()
```

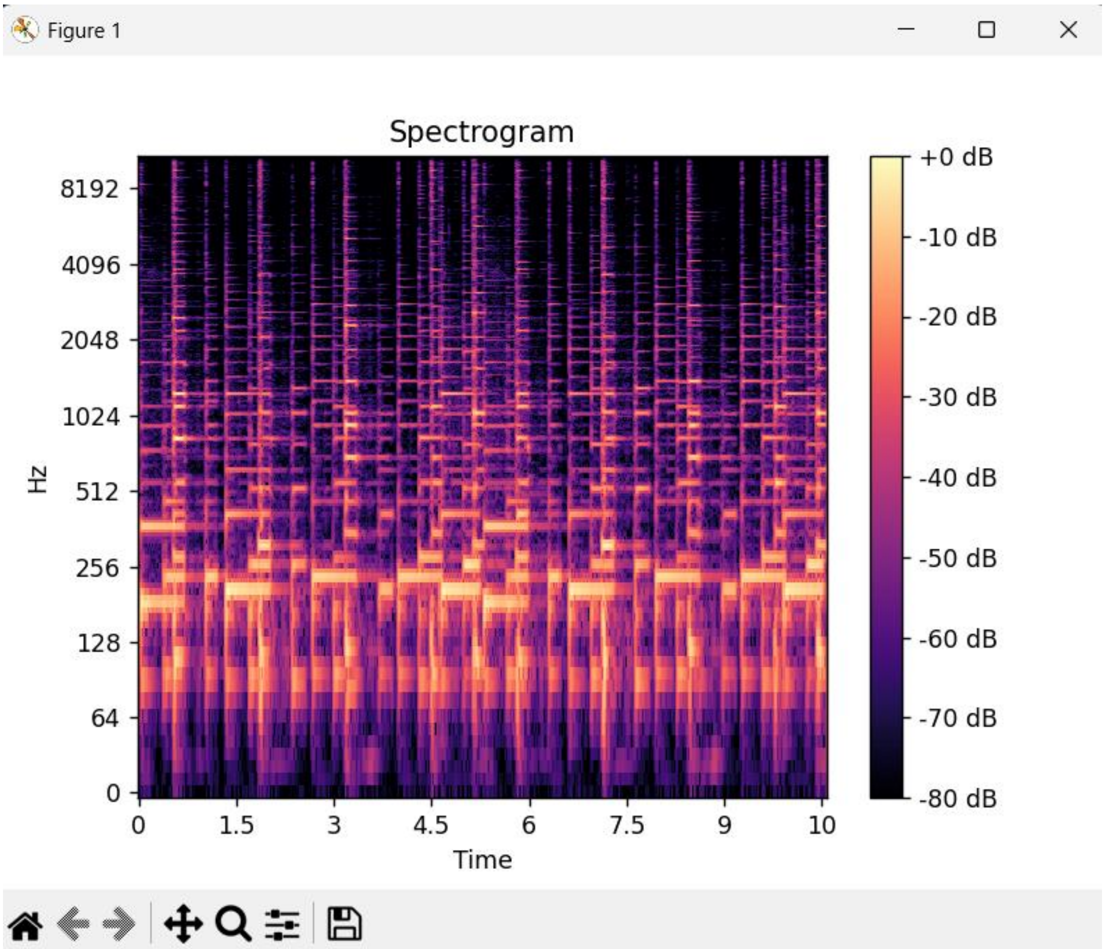
NAME: VIDYA SINHA
ROLL no: 92200133021

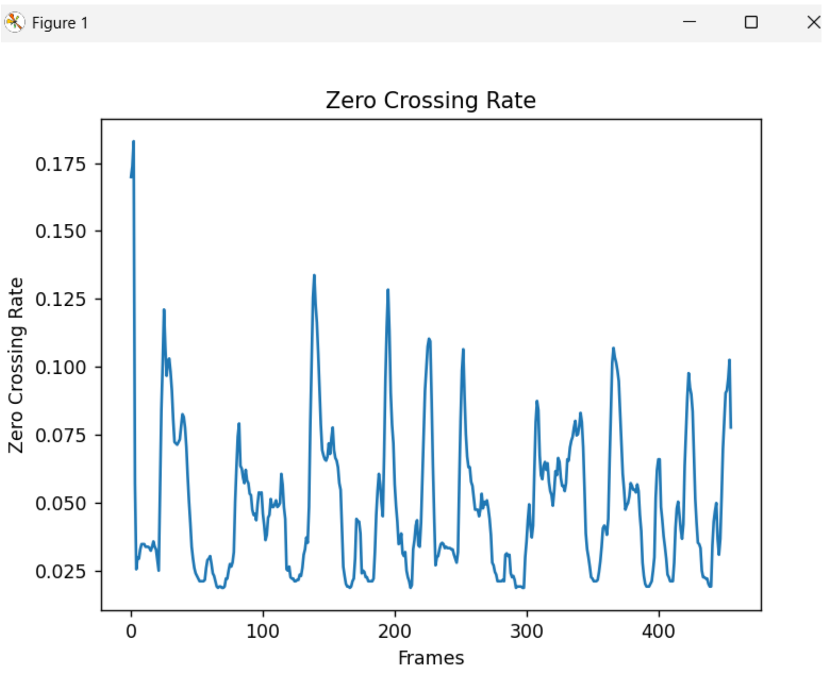
```
# Compute chroma feature

chroma = librosa.feature.chroma_stft(y=y, sr=sr)


# Display Chroma Feature

librosa.display.specshow(chroma, y_axis='chroma', x_axis='time')

plt.colorbar()

plt.title('Chroma Feature')

plt.show()
```

OUTPUT:

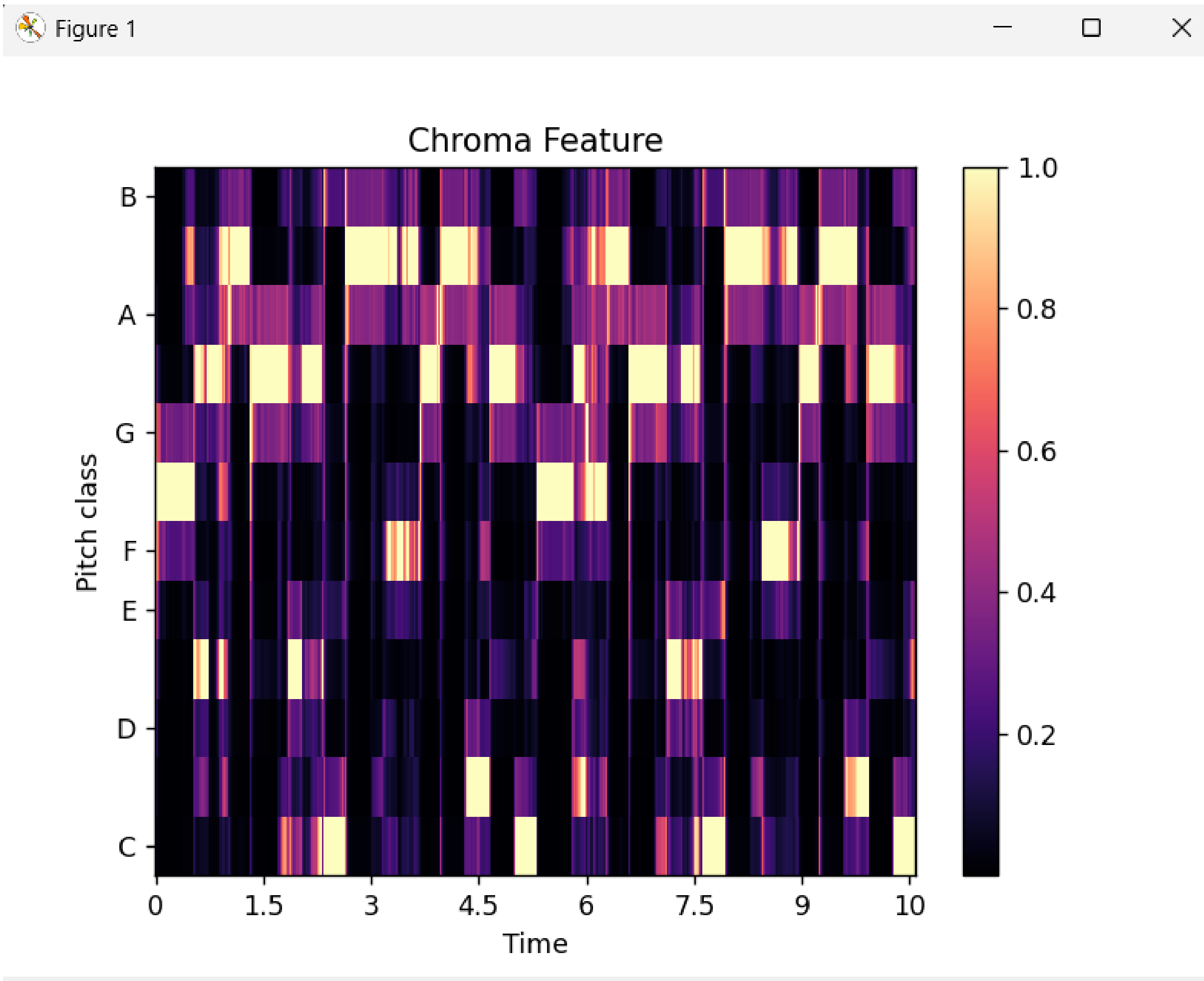


NAME: VIDYA SINHA
ROLL no: 92200133021





NAME: VIDYA SINHA
ROLL no: 92200133021



3 Video Selection

When dealing with **video feature extraction** (such as optical flow, motion analysis, and color histograms), it's important to consider the following:

- **Optical Flow and Motion Detection**:
  - **Best Video Types**: Videos with significant movement, like moving vehicles, walking people, or sports activities (e.g., football, basketball).
  - **Why**: Optical flow and motion features are most useful when there is noticeable movement. These methods help track motion patterns or object displacement in consecutive frames.
- **Color Histograms**:
  - **Best Video Types**: Videos with clear color changes or scenes with different lighting (e.g., a video of a cityscape transitioning between day and night, or a film with various costumes).
  - **Why**: Color histograms are useful for tracking and analyzing color distribution over time, such as in video color correction or scene change detection.

NAME: VIDYA SINHA
ROLL no: 92200133021

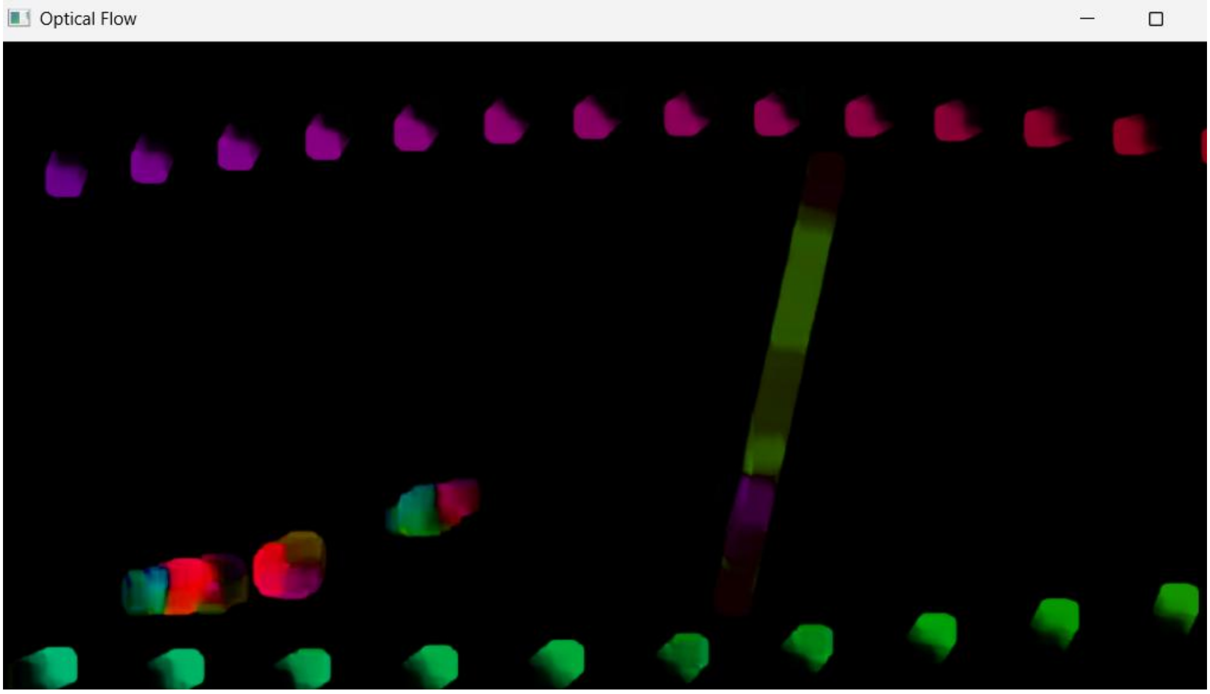
- **Video with Distinct Objects**:
    - **Best Video Types**: Videos that feature moving objects with different shapes and colors (e.g., animals in motion, vehicles driving).
    - **Why**: Distinct objects in video frames help in detecting motion, tracking, and applying features like HOG or object recognition.

```
import cv2  # OpenCV for video and image processing

import numpy as np  # NumPy for numerical operations


# Load a video file (ensure the video file is in the same directory or provide a full path)

cap = cv2.VideoCapture("video2.mp4")

if not cap.isOpened():

   raise FileNotFoundError("Video file not found or unable to open.")


# Read the first frame

ret, prev_frame = cap.read()

if not ret:

   raise ValueError("Failed to read the first frame of the video.")


# Convert the first frame to grayscale

prev_gray = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)


while cap.isOpened():

   # Read the next frame

   ret, frame = cap.read()

   if not ret:

      break


   # Convert current frame to grayscale

   gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)


   # Calculate optical flow using the Farneback method
```

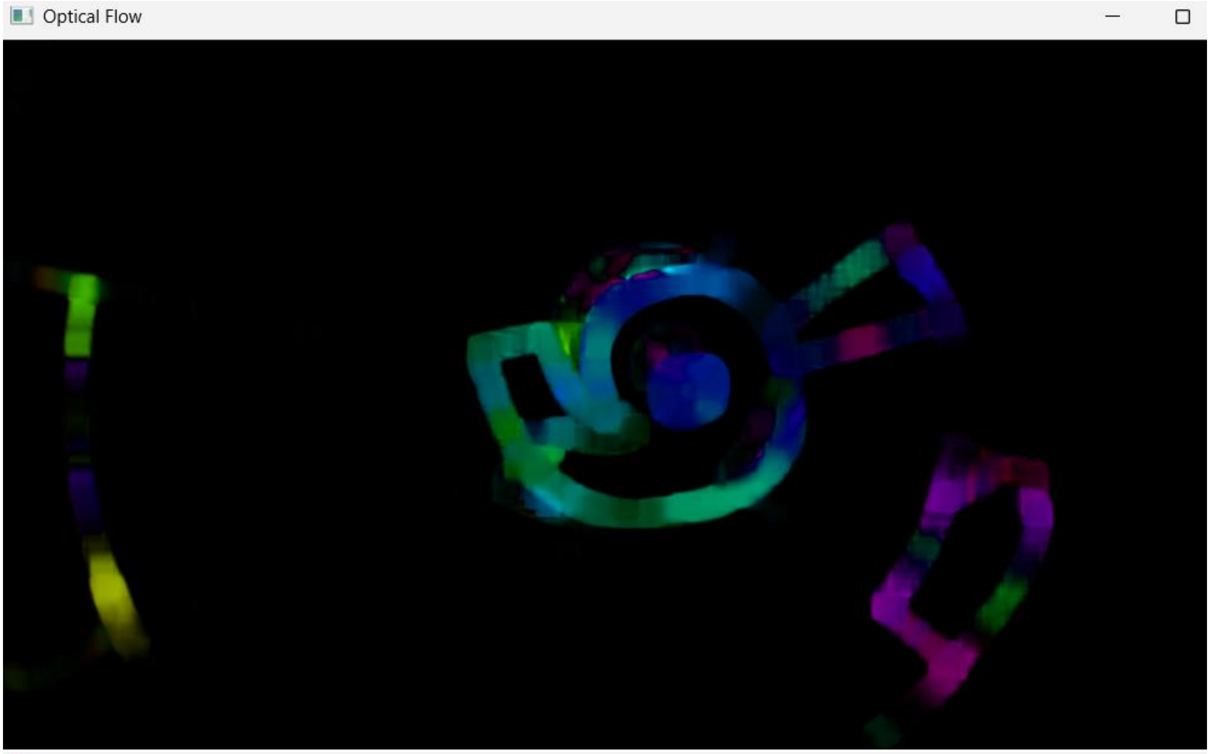
NAME: VIDYA SINHA
ROLL no: 92200133021

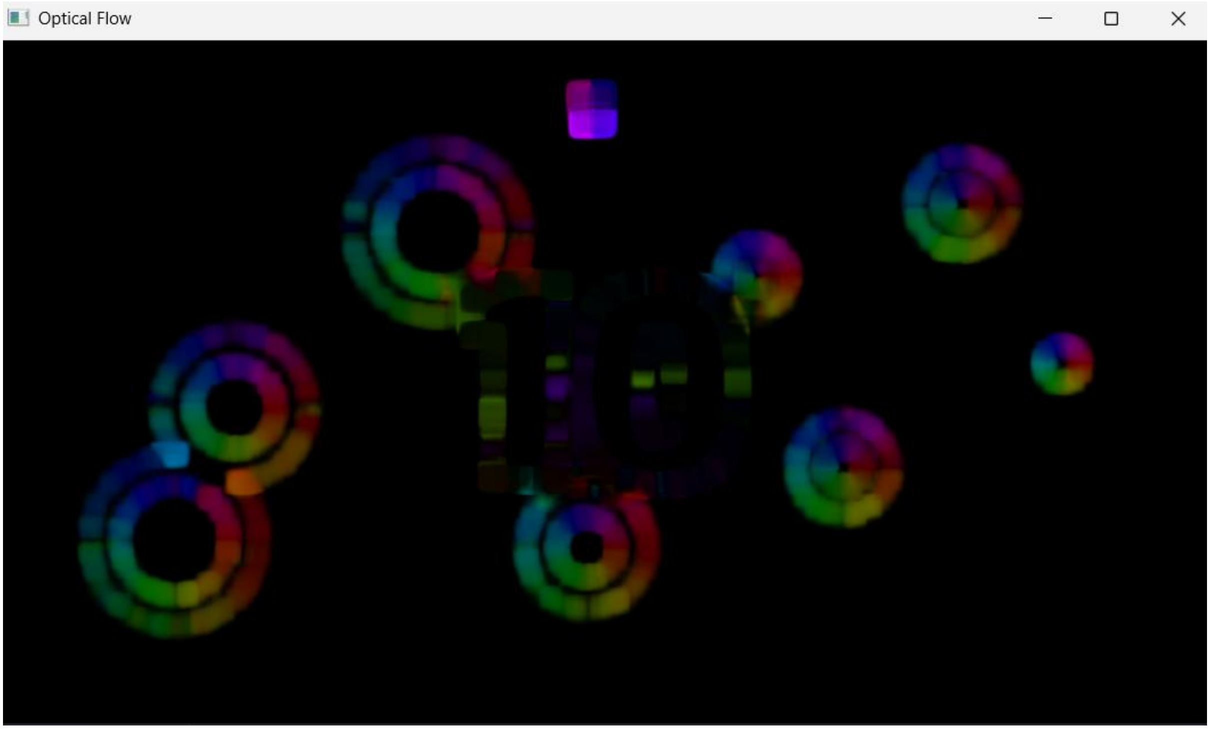
```
    flow = cv2.calcOpticalFlowFarneback(prev_gray, gray, None, 0.5, 3, 15, 3, 5, 1.2, 0)


    # Compute magnitude and angle of the optical flow
    magnitude, angle = cv2.cartToPolar(flow[..., 0], flow[..., 1])


    # Create an HSV image to represent motion
    hsv = np.zeros_like(frame)
    hsv[..., 0] = angle * 180 / np.pi / 2  # Hue represents direction
    hsv[..., 1] = 255  # Saturation
    hsv[..., 2] = cv2.normalize(magnitude, None, 0, 255, cv2.NORM_MINMAX)  # Value represents
speed


    # Convert HSV image to BGR for display
    rgb_flow = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)


    # Display the optical flow
    cv2.imshow('Optical Flow', rgb_flow)


    # Update the previous frame
    prev_gray = gray


    # Break the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break


# Release resources
cap.release()
cv2.destroyAllWindows()
```

NAME: VIDYA SINHA
ROLL no: 92200133021



NAME: VIDYA SINHA
ROLL no: 92200133021





NAME: VIDYA SINHA
ROLL no: 92200133021

```
R Histogram: [0.00630246 0.00036835 0.00028718 0.00072108 0.00074293] ...
B Histogram: [0.00635536 0.06711756 0.00020886 0.00246656 0.00042435] ...
G Histogram: [8.3592094e-02 2.6341701e-05 2.3048988e-05 1.3170850e-05 2.9634413e-05] ...
R Histogram: [0.00640807 0.00038336 0.00074028 0.00124592 0.00176148] ...
B Histogram: [0.00592407 0.05361442 0.00066477 0.00172079 0.00092098] ...
G Histogram: [7.2611123e-02 3.4373756e-05 3.0936382e-05 2.7499005e-05 2.4061630e-05] ...
R Histogram: [0.00512746 0.00064568 0.00030385 0.00097715 0.00062496] ...
B Histogram: [0.00859982 0.04099368 0.00072082 0.00267273 0.00071011] ...
G Histogram: [6.8799533e-02 9.9201134e-05 8.5029547e-05 1.2400142e-04 1.1337273e-04] ...
R Histogram: [0.00342552 0.00123561 0.00037389 0.00151335 0.00157389] ...
B Histogram: [0.00914015 0.03007521 0.00068662 0.00180412 0.00065049] ...
G Histogram: [0.05224937 0.0001526  0.00010821 0.0002719  0.00027745] ...
R Histogram: [0.0048572  0.00245093 0.00221645 0.00715739 0.00678612] ...
B Histogram: [0.00896323 0.0290069  0.00066902 0.00208766 0.00062872] ...
G Histogram: [0.05163301 0.00012397 0.00012936 0.0003207  0.0002695 ] ...
R Histogram: [0.00340392 0.00201804 0.00049978 0.00218823 0.00347415] ...
B Histogram: [0.00770599 0.0283051  0.00057101 0.0026416  0.00061637] ...
G Histogram: [4.9725749e-02 8.0423335e-05 1.5816590e-04 3.2437412e-04 2.6539702e-04] ...
R Histogram: [0.00282041 0.00184628 0.0004401  0.00232396 0.00307267] ...
B Histogram: [0.00708534 0.02754908 0.00048755 0.00235825 0.00049285] ...
G Histogram: [4.6795391e-02 8.2663231e-05 1.7865925e-04 2.2665726e-04 2.6932216e-04] ...
R Histogram: [0.00256919 0.00186486 0.0004242  0.00246514 0.00270525] ...
B Histogram: [0.00588679 0.02704935 0.0003253  0.00215115 0.00052205] ...
G Histogram: [4.3304905e-02 7.9139078e-05 1.2398456e-04 1.7674395e-04 2.3214130e-04] ...
R Histogram: [0.00156484 0.00157276 0.00022694 0.0020583  0.00198705] ...
B Histogram: [0.00527064 0.02651976 0.00026028 0.00195729 0.00035398] ...
G Histogram: [4.0128388e-02 5.4877317e-05 1.2282067e-04 1.1759425e-04 2.0383004e-04] ...
R Histogram: [0.00121363 0.00158504 0.00020401 0.00156673 0.00151703] ...
B Histogram: [0.00458784 0.02609271 0.00019159 0.00183048 0.00027703] ...
G Histogram: [3.74337360e-02 4.15612922e-05 1.01305646e-04 4.15612922e-05
 1.92220978e-04] ...
R Histogram: [0.00121974 0.00144081 0.00014824 0.00144601 0.00111051] ...
B Histogram: [0.00393379 0.02581018 0.00014922 0.00171605 0.00018009] ...
G Histogram: [3.5237581e-02 3.0950883e-05 8.7694170e-05 2.8371644e-05 1.8054682e-04] ...
R Histogram: [8.1863173e-04 1.2808875e-03 8.0055470e-05 1.2757226e-03 8.1088440e-04] ...
```