Identifying Bird Calls using Content Based Image Retrieval Framework (CBIR)

Ritesh Sharma

PhD Student. EECS, UC Merced

1 Introduction

Birds serves as an important and significant member of the biodiversity prevalent in the ecosystem.The monitoring of bird populations can provide significant information on the state of the ecosystem. Manual monitoring of accurate yet approximate population data is very labor-intensive, time consuming and error-prone. Automated monitoring of population using computer vision technique sounds like a good option but not too reliable as the vision technique can depend on lighting conditions. So, a different approach is required for collecting information about birds.

In order to effectively estimate population of different types of birds, we plan to use well known image processing technique, Content-Based Image Retrieval(CBIR). In this project, 218 recording of five different class of birds are used. First, the recordings are converted into spectrograms. Next, we compute region of interest for the bird call. Then we extract different features and use technique to compute similarity between spectrogram of given bird call with the given dataset. In this work, we used two different features: (a) Mean and Standard Deviation Features and (b) Gabor Filter based features. To compute similarity, L1 distance, L2 distance and Minkowski distance measure are used. At the end, we compare different configurations to test our CBIR framework. In our experiment we found that the Gabor Filter based features with 3 scales and 6 rotations along with minkowski distance measures gave us the better result than most of the configurations.

2 Results

2.1 Configurations

Table below shows all the configuration that I have tested for this project.

				Spectrogam		
Configurations	**Features**	**Dist**	**Norm**	**Win.Size**	**Win.Over.**	**Mapping**
Gabor(4,6) L1	Gabor 4 Scales, 6 orientations	L1	None	1024	512	Log10
Gabor(4,6) L2	Gabor 4 Scales, 6 orientations	L2	None	1024	512	Log10
Gabor(4,6) Minkowski	Gabor 4 Scales, 6 orientations	Minkowski	None	1024	512	Log10
Gabor(3,6) L1	Gabor 3 Scales, 6 orientations	L1	None	1024	512	Log10
Gabor(3,6) L2	Gabor 3 Scales, 6 orientations	L2	None	1024	512	Log10
Gabor(3,6) Minkowski	Gabor 3 Scales, 6 orientations	Minkowski	None	1024	512	Log10
Gabor(2,3) L1	Gabor 2 Scales, 3 orientations	L1	None	1024	512	Log10
Gabor(2,3) L2	Gabor 2 Scales, 3 orientations	L2	None	1024	512	Log10
Gabor(2,3) Minkowski	Gabor 2 Scales, 3 orientations	Minkowski	None	1024	512	Log10
Mean StdDev L1	Mean & Standard Deviation	L1	None	1024	512	Log10
Mean StdDev L2	Mean & Standard Deviations	L2	None	1024	512	Log10
Mean StdDev Minkowski	Mean & Standard Deviation	Minkowski	None	1024	512	Log10

2.2 Plot

Figure 1 shows precision-recall curve for 12 different configuration shown in the section above.

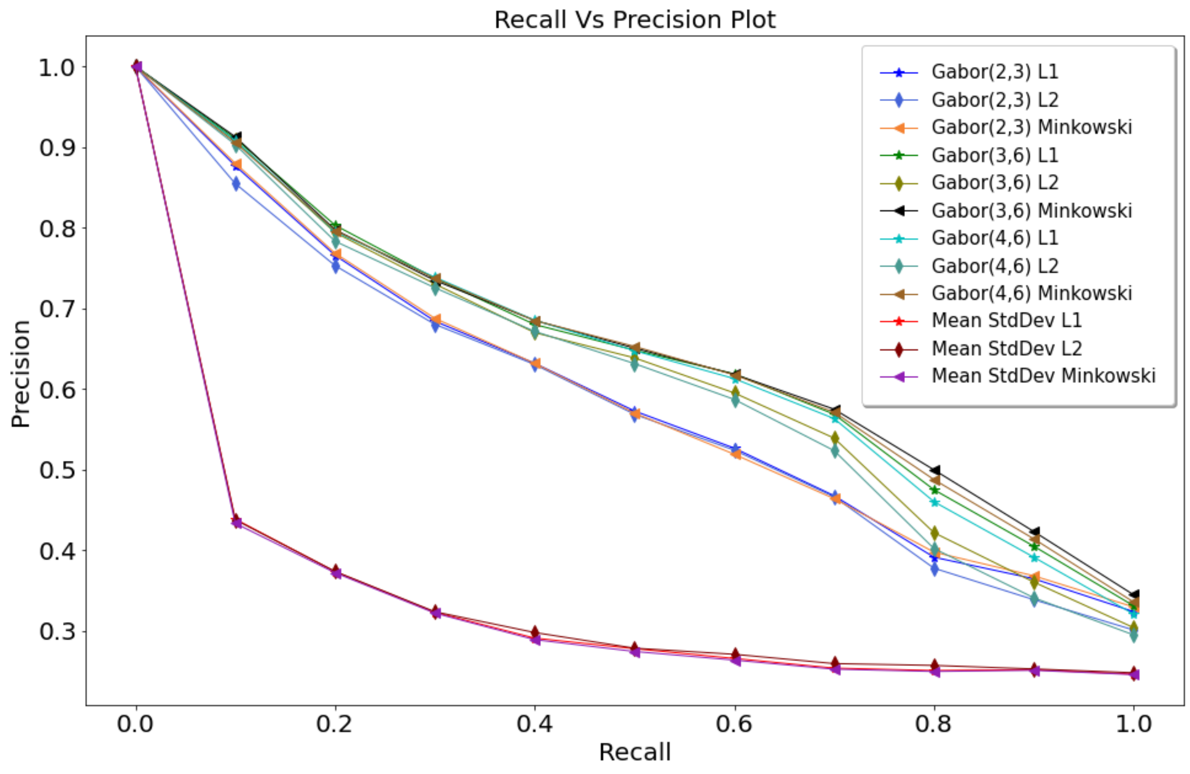
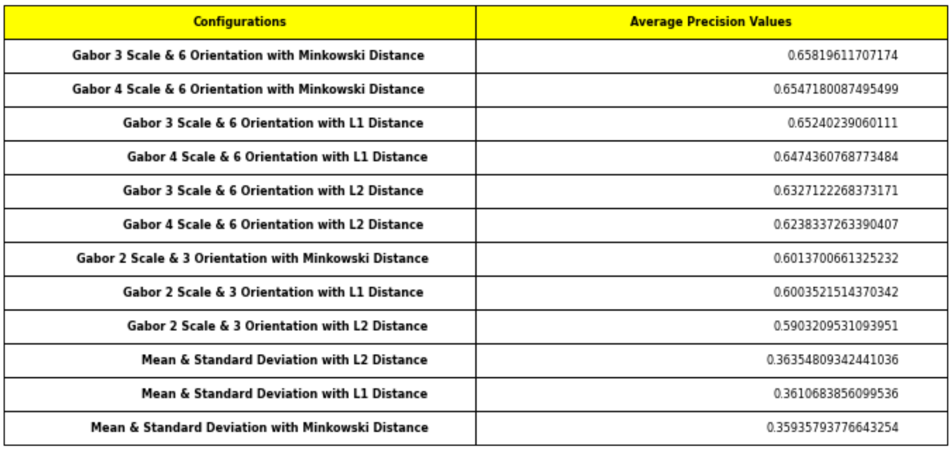


Figure 1: Precision & Recall Curve

2.3 Summary

Figure 2 shows the average precision value for all the tested configurations. The rows in the table are sorted from top rows showing the best configuration with highest average precision value to the configuration with least average precision value at the bottom row.



Configurations	Average Precision Values
Gabor 3 Scale & 6 Orientation with Minkowski Distance	0.65819611707174
Gabor 4 Scale & 6 Orientation with Minkowski Distance	0.6547180087495499
Gabor 3 Scale & 6 Orientation with L1 Distance	0.65240239060111
Gabor 4 Scale & 6 Orientation with L1 Distance	0.6474360768773484
Gabor 3 Scale & 6 Orientation with L2 Distance	0.6327122268373171
Gabor 4 Scale & 6 Orientation with L2 Distance	0.6238337263390407
Gabor 2 Scale & 3 Orientation with Minkowski Distance	0.6013700661325232
Gabor 2 Scale & 3 Orientation with L1 Distance	0.6003521514370342
Gabor 2 Scale & 3 Orientation with L2 Distance	0.5903209531093951
Mean & Standard Deviation with L2 Distance	0.36354809342441036
Mean & Standard Deviation with L1 Distance	0.3610683856099536
Mean & Standard Deviation with Minkowski Distance	0.35935793776643254

Figure 2: Shows average precision values for all the tested configuration

3 Discussion

In this project, I tried 12 different configurations to achieve best possible average precision value. The configurations along with precision values in descending order is shown in Figure 2. It is observed that mean and standard deviation features gave the worst precision values where as features based on Gabor filter gave considerably better precision values. I tried with several scales and orientation for extracting featured based on Gabor Filter. I observed that the while increasing scale had less effect compared to changing orientation. I also found that when there were odd number of orientations, the precision values were better. Also increasing number of orientation gave better precision values. After experimenting with various scales and orientations, I chose following configuration for gabor filter based features: (a) Scales-4, Orientations-6 (b) Scales-3, Orientations-6, and (c) Scales-2, Orientations-3. The main observation from experiments are as follows:

- Gabor Filter based feature extraction are best for CBIR framework as compared to features based on mean and standard deviation
- Increasing Orientations are better than increasing scales in Gabor Filter based features
- Odd number of orientations works better than even number of orientations
- Minkowski distance measure gave better average precision values than L1 and L2 measure for Gabor filter based features
- L2 distance measure gave better average precision values than L1 and minkowski measure for mean and standard deviation features

4 Code

4.1 Initial set up

```
# Import section
import numpy as np
import IPython.display as ipd
import matplotlib.pyplot as plt
import librosa
import os
import pandas as pd
from scipy import signal
from scipy import ndimage as nd
from scipy import signal as sg
from skimage import data
from skimage.util import img_as_float
from skimage.filters import gabor_kernel

# Compute Spectogram from the audio recording

# Expect the sampling rate to be the same for all recordings.
sampling_rate_expected = 44100

# The window size for the spectrogram.
```

```
window_size = 1024

# The window overlap for the spectrogram.
window_overlap = 512

currentWorkingDirectory=os.getcwd()

print(currentWorkingDirectory)
# The directory where the .wav files are located. We will read
    these.
recording_dir = currentWorkingDirectory +'/Recordings/'

# The directory where we will write the computed spectrograms
   as .png image files.
spectrogram_dir = currentWorkingDirectory +'/Spectrograms/'

# Write spectrogram parameters.
parameters_filename = spectrogram_dir + '0
   _spectrogram_parameters.txt'
spectogram_param_file = open(parameters_filename,'w')
spectogram_param_file.write('sampling rate = '+ repr(
   sampling_rate_expected)+'\n')
spectogram_param_file.write('window size = '+ repr(window_size
   )+'\n')
spectogram_param_file.write('window overlap = '+ repr(
   window_overlap)+'\n')
spectogram_param_file.write('log10 of spectrum\n')
spectogram_param_file.close()

# Get list of recordings.
recording_list = os.listdir(recording_dir);
n_recordings = len(recording_list)
#print(n_recordings)

# Read each recording and compute spectrogram
recording_num = 0
for recording in recording_list:
    recording_num = recording_num + 1
    recording_filename = recording_dir + recording
    print('recording '+repr(recording_num)+' of '+repr(
       n_recordings)+': about to read '+recording)
    # Read the audio file. Don't change the sample rate.
    x, sampling_rate = librosa.load(recording_filename,sr=None
       )

    if sampling_rate != sampling_rate_expected:
        print('WRONG SAMPLING RATE: expected='+repr(
           sampling_rate_expected)+' recording='+repr(
            sampling_rate))
        break
```

4.2 Compute Spectogram

```
# Compute the spectrogram.
w, t, s = signal.spectrogram(x, sampling_rate, noverlap=
    window_overlap, nperseg=window_size)

# s: matrix with the 2D spectrogram (will be complex).
# w: vector with the frequencies spacings of the computed DFT.
# t: vector with the time spacings of the windows.

# Compute the spectrum of the spectrogram.
s_spectrum = abs(s);

# Flip it vertically.
s_spectrum = np.flip(s_spectrum, 0)

# Compute log_10 of spectrogram.
s_spectrum_log = np.log10( s_spectrum + np.finfo(float).eps )
#plt.imshow(s_spectrum_log,cmap='gray')

# Normalize so values range from 0 to 1.
s_spectrum_log = (s_spectrum_log - np.amin(s_spectrum_log)) /
    (np.amax(s_spectrum_log) - np.amin(s_spectrum_log))

# Save as png file.
spectrogram_filename = spectrogram_dir + recording +'.png'
plt.imsave(spectrogram_filename, s_spectrum_log, cmap='gray')

# Save time and frequency indices.
spectrograminfo_filename = spectrogram_dir + recording + '
    _info.txt'
spectrograminfo_file = open(spectrograminfo_filename,'w')
for i in t:
    spectrograminfo_file.write(repr(i)+'\n')
for i in w:
    spectrograminfo_file.write(repr(i)+'\n')
spectrograminfo_file.close()
```

4.3 Compute Region of Interest (ROIs)

```
# Import section
import pandas as pd
from PIL import Image, ImageDraw
```

```
# root_dir of where to put annotated spectrograms, etc.
root_dir = currentWorkingDirectory+'/ROIs/'

# Where the computed spectrograms are located.
spectrogram_dir = currentWorkingDirectory+'/Spectrograms/'

# Where the ROI .csv files are located.
rois_dir = currentWorkingDirectory+'/ROIs/'

# The .csv file with the info about the ROIs.
rois_info_file = currentWorkingDirectory+'/ROIs/birds.csv'

# Read the ROI info.
rois_info = pd.read_csv(rois_info_file)

n_birds = 5;


# For each bird, read the ROI file and process the ROIs.
#for i in range(1):
for i in range(n_birds):
    print(i)
    # Read ROI .csv file for this bird.
    roi_csv_file = rois_dir + rois_info['roi_file'][i]
    bird_rois_info = pd.read_csv(roi_csv_file);

    n_rois = rois_info['roi_count'][i]

    # Process each ROI for this bird.
    for j in range(n_rois):
        # Always get original spectroram.
        recording = bird_rois_info['recording'][j]
        spectrogram = recording + '.png'
        spectrogram_filename = spectrogram_dir + spectrogram
        spectrogram_image = plt.imread(spectrogram_filename)

        # Check to see if this spectrogram has already been
           marked with an ROI.
        spectrogram_roi = recording + '.roi.png'
        spectrogram_roi_filename = root_dir + rois_info['bird
           '][i] + '/' + spectrogram_roi

        if os.path.exists(spectrogram_roi_filename):
            # Read already marked up spectrogram.
            spectrogram_roi_image_PIL = Image.open(
               spectrogram_roi_filename);
        else:
            spectrogram_roi_image_PIL = Image.open(
               spectrogram_filename);

        # Get size of spectrogram.
        n_rows = np.shape(spectrogram_image)[0]
        n_cols = np.shape(spectrogram_image)[1]

        # Get time and frequency coordinates of spectrogram
           from _info.txt file.
        spectrogram_info = recording + '_info.txt'
        spectrogram_info_filename = spectrogram_dir +
           spectrogram_info
        spectrograminfo_file = open(spectrogram_info_filename
           ,'r')

        time_coords = np.zeros(n_cols)
        for x in range(n_cols):
            time_coords[x] = spectrograminfo_file.readline()
        freq_coords = np.zeros(n_rows)
        for x in range(n_rows):
            freq_coords[x] = spectrograminfo_file.readline()
        spectrograminfo_file.close()
        # Reverse.
        freq_coords = np.flip(freq_coords)

        # Find pixel bounds of ROI.
        top_left_row = np.where(freq_coords < bird_rois_info['
           y2'][j])
        bottom_right_row = np.where(freq_coords <
           bird_rois_info['y1'][j])
        top_left_col = np.where(time_coords > bird_rois_info['
           x1'][j])
        bottom_right_col = np.where(time_coords >
           bird_rois_info['x2'][j])

        width = bottom_right_col[0][0] - top_left_col[0][0] +
           1;
        height = bottom_right_row[0][0] - top_left_row[0][0] +
            1;



        # Draw rectangle.
        img1 = ImageDraw.Draw(spectrogram_roi_image_PIL)
        img1.rectangle([(top_left_col[0][0],top_left_row
           [0][0]),(bottom_right_col[0][0],bottom_right_row
           [0][0])], outline ='white')

        # Save spectrogram with marked ROIs.
        spectrogram_roi_image_PIL.save(
            spectrogram_roi_filename)

        # Extract and save ROI.
        image_roi = spectrogram_image[top_left_row[0][0]-1:
            bottom_right_row[0][0]-1, top_left_col[0][0]-1:
            bottom_right_col[0][0]-1]
        image_roi_filename = root_dir+rois_info['bird'][i]+'/
            ROIs/'+rois_info['bird'][i]+'_'+repr(bird_rois_info
            ['id'][j])+'.png'
        plt.imsave(image_roi_filename, image_roi, cmap='gray')

print("ROI Computation Completed")
```

4.4 Feature Extraction

```
import shutil

# Location of images.
image_dir = currentWorkingDirectory + '/images/'

# Location of where to write features.
feature_dir = currentWorkingDirectory + '/features/'

# The .csv file containing the image names and classes.
image_file = currentWorkingDirectory + '/image_names_classes.
    csv'

# Number of images.
n_images = 218

# Read image names and classes .csv file.
image_names_classes = pd.read_csv(image_file, header=None )
```

4.4.1 Mean & Standard Deviation

```
# Simple features have dimension 2
fdim = 2

features = np.zeros((n_images, fdim))

# Extract features std_mean for each image.
for i in range(n_images):

    # Read the image.
    filename = image_dir + image_names_classes[0][i]
    im = plt.imread(filename)
#     print(im.shape)
```

```
    # It turns out that the spectrogram images saved using plt
       .imsave have four channels
    # RGBA. The RGB channels are each equal to the grayscale
       value so we can use any of them.

    # Compute the mean of the grayscale image as the first
       feature dimension.
    features[i,0] = np.mean(im[:,:,0])

    # Compute the standard deviation of the grayscale image as
        the second feature dimension.
    features[i,1] = np.std(im[:,:,0])

# Save the features as a .csv file.
feature_filename = feature_dir + 'mean_stddev.txt'

np.savetxt(feature_filename, features, delimiter=',')
```

4.4.2 Gabor Filter with 2 Scale & 3 Orientation

```
# prepare filter bank kernels
kernels = []
# EXPERIMENT WITH THE NUMBER OF ORIENTATIONS.
# MAKE SURE TO CHANGE THE FEATURE FILENAME BELOW TO INDICATE
   THE NUMBER OF ORIENTATIONS: Gabor_Y_X_.txt
# WHERE X IS THE NUMBER OF SCALES AND Y IS THE NUMBER OF
   ORIENTATIONS.
norientations = 3
scales=len([0.05, 0.25])
for theta in range(norientations):
    theta = theta / norientations * np.pi
    # EXPERIMENT WITH THE NUMBER OF SCALES AND THE FREQUENCIES
        OF THE FILTERS.
#     for frequency in (0.05, 0.1, 0.20, 0.4):
    for frequency in (0.05, 0.25):
        kernel = gabor_kernel(frequency, theta=theta)
        kernels.append(kernel)

fig, axs = plt.subplots(1, len(kernels), figsize=(20,20))
for k, kernel in enumerate(kernels):
    axs[k].imshow(np.real(kernel),cmap='gray')

fig, axs = plt.subplots(1, len(kernels), figsize=(20,20))
for k, kernel in enumerate(kernels):
    axs[k].imshow(np.imag(kernel),cmap='gray')


# Mean and standard deviation will be computed for each filter
```

```
    output.
fdim = 2 * len(kernels)

features = np.zeros((n_images, fdim))

def compute_feats(image, kernels):
    feats = np.zeros((len(kernels) * 2), dtype=np.double)
    for k, kernel in enumerate(kernels):
        filtered = sg.convolve(image, kernel)
        feats[2*k] = np.abs(filtered).mean()
        feats[2*k+1] = np.abs(filtered).std()
    return feats

# Extract features for each image.
for i in range(n_images):

    # Read the image.
    filename = image_dir + image_names_classes[0][i]
    im = plt.imread(filename)

    # It turns out that the spectrogram images saved using plt
       .imsave have four channels
    # RGBA. The RGB channels are each equal to the grayscale
       value so we can use any of them.

    features[i, :] = compute_feats(im[:,:,0], kernels)

# Save the features as a .csv file.
gabor_filename='Gabor'+'_S'+str(scales)+'_R'+str(norientations
   )+'.txt'
feature_filename = feature_dir +  gabor_filename

np.savetxt(feature_filename, features, delimiter=',')
```

4.4.3 Gabor Filter with 3 Scale & 6 Orientation

```
# prepare filter bank kernels
kernels = []
# EXPERIMENT WITH THE NUMBER OF ORIENTATIONS.
# MAKE SURE TO CHANGE THE FEATURE FILENAME BELOW TO INDICATE
   THE NUMBER OF ORIENTATIONS: Gabor_Y_X_.txt
# WHERE X IS THE NUMBER OF SCALES AND Y IS THE NUMBER OF
   ORIENTATIONS.
norientations = 6
scales=len([0.05, 0.1, 0.20])
for theta in range(norientations):
    theta = theta / norientations * np.pi
    # EXPERIMENT WITH THE NUMBER OF SCALES AND THE FREQUENCIES
```

```
        OF THE FILTERS.
#    for frequency in (0.05, 0.1, 0.20, 0.4):
    for frequency in (0.05, 0.1, 0.20):
        kernel = gabor_kernel(frequency, theta=theta)
        kernels.append(kernel)

fig, axs = plt.subplots(1, len(kernels), figsize=(20,20))
for k, kernel in enumerate(kernels):
    axs[k].imshow(np.real(kernel),cmap='gray')

fig, axs = plt.subplots(1, len(kernels), figsize=(20,20))
for k, kernel in enumerate(kernels):
    axs[k].imshow(np.imag(kernel),cmap='gray')


# Mean and standard deviation will be computed for each filter
    output.
fdim = 2 * len(kernels)

features = np.zeros((n_images, fdim))

def compute_feats(image, kernels):
    feats = np.zeros((len(kernels) * 2), dtype=np.double)
    for k, kernel in enumerate(kernels):
        filtered = sg.convolve(image, kernel)
        feats[2*k] = np.abs(filtered).mean()
        feats[2*k+1] = np.abs(filtered).std()
    return feats

# Extract features for each image.
for i in range(n_images):

    # Read the image.
    filename = image_dir + image_names_classes[0][i]
    im = plt.imread(filename)

    # It turns out that the spectrogram images saved using plt
        .imsave have four channels
    # RGBA. The RGB channels are each equal to the grayscale
        value so we can use any of them.

    features[i, :] = compute_feats(im[:,:,0], kernels)

# Save the features as a .csv file.
gabor_filename='Gabor'+'_S'+str(scales)+'_R'+str(norientations
    )+'.txt'
feature_filename = feature_dir + gabor_filename
```

```
np.savetxt(feature_filename, features, delimiter=',')
```

4.4.4 Gabor Filter with 4 Scale & 6 Orientation

```
# prepare filter bank kernels
kernels = []
# EXPERIMENT WITH THE NUMBER OF ORIENTATIONS.
# MAKE SURE TO CHANGE THE FEATURE FILENAME BELOW TO INDICATE
   THE NUMBER OF ORIENTATIONS: Gabor_Y_X_.txt
# WHERE X IS THE NUMBER OF SCALES AND Y IS THE NUMBER OF
   ORIENTATIONS.
norientations = 6
scales=len([0.05, 0.1, 0.20, 0.4])
for theta in range(norientations):
    theta = theta / norientations * np.pi
    # EXPERIMENT WITH THE NUMBER OF SCALES AND THE FREQUENCIES
        OF THE FILTERS.
    for frequency in (0.05, 0.1, 0.20, 0.4):
        kernel = gabor_kernel(frequency, theta=theta)
        kernels.append(kernel)

fig, axs = plt.subplots(1, len(kernels), figsize=(20,20))
for k, kernel in enumerate(kernels):
    axs[k].imshow(np.real(kernel),cmap='gray')

fig, axs = plt.subplots(1, len(kernels), figsize=(20,20))
for k, kernel in enumerate(kernels):
    axs[k].imshow(np.imag(kernel),cmap='gray')

# Mean and standard deviation will be computed for each filter
    output.
fdim = 2 * len(kernels)

features = np.zeros((n_images, fdim))

def compute_feats(image, kernels):
    feats = np.zeros((len(kernels) * 2), dtype=np.double)
    for k, kernel in enumerate(kernels):
        filtered = sg.convolve(image, kernel)
        feats[2*k] = np.abs(filtered).mean()
        feats[2*k+1] = np.abs(filtered).std()
    return feats

# Extract features for each image.
for i in range(n_images):

    # Read the image.
    filename = image_dir + image_names_classes[0][i]
    im = plt.imread(filename)

    # It turns out that the spectrogram images saved using plt
       .imsave have four channels
    # RGBA. The RGB channels are each equal to the grayscale
       value so we can use any of them.

    features[i, :] = compute_feats(im[:,:,0], kernels)

# Save the features as a .csv file.
gabor_filename='Gabor'+'_S'+str(scales)+'_R'+str(norientations
   )+'.txt'
feature_filename = feature_dir +  gabor_filename

np.savetxt(feature_filename, features, delimiter=',')
```

4.5 Compute Distance Measures

```
# Declare dictionaries to store precision and recall values
from collections import OrderedDict
import math
import os
mean_stddev_L1= OrderedDict()
mean= OrderedDict()
std= OrderedDict()
Gabor_S2_R3_L1= OrderedDict()
Gabor_S3_R6_L1= OrderedDict()
Gabor_S4_R6_L1= OrderedDict()

mean_stddev_L2= OrderedDict()
Gabor_S2_R3_L2= OrderedDict()
Gabor_S3_R6_L2= OrderedDict()
Gabor_S4_R6_L2= OrderedDict()

mean_stddev_minkowski= OrderedDict()
Gabor_S2_R3_minkowski= OrderedDict()
Gabor_S3_R6_minkowski= OrderedDict()
Gabor_S4_R6_minkowski= OrderedDict()

def prec_recal(prec_num,prec_denom,recal_num,recal_denom):
    return (prec_num/prec_denom,recal_num/recal_denom)

#Dictionary to store the class and class size of birds
bird_class_size_dict=OrderedDict()
rois_info = pd.read_csv(rois_info_file)

#Store the class and class size of birds
for i in range(len(list(rois_info['bird']))):
    bird_class_size_dict[i+1]=rois_info['roi_count'].iloc[i]

# Number of images.
n_images = 218
```

4.5.1 L1 Distance Measure

```
for i in os.listdir(feature_dir):
    if i.split(".")[1]=='txt':
        feature_filename = feature_dir + i
        features = np.genfromtxt(feature_filename, delimiter
           =',')
        fdim = np.shape(features)[1]
        print(i,fdim)

        for query_image in range(n_images):
            # Compute Euclidean distance between query feature
                vector and each image's feature vector.
            distances = np.zeros(n_images)

            for j in range(n_images):
                distances[j] = 0
                for k in range(fdim):
                    distances[j] = distances[j] + ((features[
                       query_image][k] - features[j][k])**2)
                distances[j] = distances[j] ** 0.5

            # Get the indices of the sorted distances.
            sorted_index = np.argsort(distances)
            #Get the class and class size of query image
            class_qimg=image_names_classes[1][query_image]
            class_qimg_size= bird_class_size_dict[class_qimg]

            temp_prec_arr=[]
            temp_recal_arr=[]

            # Retrieve top k images.
            for k in range(1,n_images+1):
                prec_num=0
                prec_denom=k
                recal_denom=class_qimg_size
                recal_num=0
                for z in sorted_index[0:k]:
                    if class_qimg==image_names_classes[1][z]:
                        prec_num+=1
                        recal_num+=1
                temp_prec_arr.append(prec_num/prec_denom)
                temp_recal_arr.append(recal_num/recal_denom)
            if i=="Gabor_S2_R3.txt":
                Gabor_S2_R3_L1[query_image]=[temp_prec_arr,
                   temp_recal_arr]
            if i=="Gabor_S3_R6.txt":
                Gabor_S3_R6_L1[query_image]=[temp_prec_arr,
                   temp_recal_arr]
            if i=="Gabor_S4_R6.txt":
                Gabor_S4_R6_L1[query_image]=[temp_prec_arr,
                   temp_recal_arr]
            if i=="mean_stddev.txt":
                mean_stddev_L1[query_image]=[temp_prec_arr,
                   temp_recal_arr]
```

4.5.2 L2 Distance Measure

```
for i in os.listdir(feature_dir):
    if i.split(".")[1]=='txt':
        feature_filename = feature_dir + i
        features = np.genfromtxt(feature_filename, delimiter
           =',')
        fdim = np.shape(features)[1]
        print(i,fdim)

        for query_image in range(n_images):
            # Compute Euclidean distance between query feature
                vector and each image's feature vector.
            distances = np.zeros(n_images)

            for j in range(n_images):
                distances[j] = 0
                for k in range(fdim):
                    distances[j] = distances[j] + (abs(
                       features[query_image][k] - features[j][k
                       ]))

            # Get the indices of the sorted distances.
            sorted_index = np.argsort(distances)
            #Get the class and class size of query image
            class_qimg=image_names_classes[1][query_image]
            class_qimg_size= bird_class_size_dict[class_qimg]

            temp_prec_arr=[]
            temp_recal_arr=[]

            # Retrieve top k images.
            for k in range(1,n_images+1):
                prec_num=0
                prec_denom=k
                recal_denom=class_qimg_size
                recal_num=0
                for z in sorted_index[0:k]:
                    if class_qimg==image_names_classes[1][z]:
                        prec_num+=1
                        recal_num+=1
                temp_prec_arr.append(prec_num/prec_denom)
                temp_recal_arr.append(recal_num/recal_denom)

            if i=="Gabor_S2_R3.txt":
                Gabor_S2_R3_L2[query_image]=[temp_prec_arr,
                   temp_recal_arr]
            if i=="Gabor_S3_R6.txt":
                Gabor_S3_R6_L2[query_image]=[temp_prec_arr,
                   temp_recal_arr]
            if i=="Gabor_S4_R6.txt":
                Gabor_S4_R6_L2[query_image]=[temp_prec_arr,
                   temp_recal_arr]
            if i=="mean_stddev.txt":
                mean_stddev_L2[query_image]=[temp_prec_arr,
                   temp_recal_arr]
```

4.5.3 Minkowski Sum Distance Measure

```
p=3;
for i in os.listdir(feature_dir):
    if i.split(".")[1]=='txt':
        feature_filename = feature_dir + i
        features = np.genfromtxt(feature_filename, delimiter
           =',')
        fdim = np.shape(features)[1]
        print(i,fdim)

        for query_image in range(n_images):
            # Compute Euclidean distance between query feature
                vector and each image's feature vector.
            distances = np.zeros(n_images)

            for j in range(n_images):
                distances[j] = 0
                for k in range(fdim):
                    distances[j] = distances[j] + pow((abs(
                       features[query_image][k] - features[j][k
                       ])),p)
                distances[j] = pow(distances[j],1/p)
            # Get the indices of the sorted distances.
            sorted_index = np.argsort(distances)
            #Get the class and class size of query image
            class_qimg=image_names_classes[1][query_image]
            class_qimg_size= bird_class_size_dict[class_qimg]

            temp_prec_arr=[]
            temp_recal_arr=[]

            # Retrieve top k images.
            for k in range(1,n_images+1):
                prec_num=0
                prec_denom=k
                recal_denom=class_qimg_size
                recal_num=0
                for z in sorted_index[0:k]:
                    if class_qimg==image_names_classes[1][z]:
                        prec_num+=1
                        recal_num+=1
                temp_prec_arr.append(prec_num/prec_denom)
                temp_recal_arr.append(recal_num/recal_denom)

            if i=="Gabor_S2_R3.txt":
                Gabor_S2_R3_minkowski[query_image]=[
                   temp_prec_arr,temp_recal_arr]
            if i=="Gabor_S3_R6.txt":
                Gabor_S3_R6_minkowski[query_image]=[
                   temp_prec_arr,temp_recal_arr]
            if i=="Gabor_S4_R6.txt":
                Gabor_S4_R6_minkowski[query_image]=[
                   temp_prec_arr,temp_recal_arr]
            if i=="mean_stddev.txt":
                mean_stddev_minkowski[query_image]=[
                   temp_prec_arr,temp_recal_arr]
```

4.6 Interpolate Precision for fixed recall values

```
ref_recall=[0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]

#Create Empty dictionaries to hold the interpolated values
mean_stddev_pr_L1= OrderedDict()
mean_stddev_pr_L2= OrderedDict()
mean_stddev_pr_minkowski= OrderedDict()
Gabor_S2_R3_pr_L1= OrderedDict()
Gabor_S2_R3_pr_L2= OrderedDict()
Gabor_S2_R3_pr_minkowski= OrderedDict()
Gabor_S3_R6_pr_L1= OrderedDict()
Gabor_S3_R6_pr_L2= OrderedDict()
Gabor_S3_R6_pr_minkowski= OrderedDict()
Gabor_S4_R6_pr_L1= OrderedDict()
```

```
Gabor_S4_R6_pr_L2= OrderedDict()
Gabor_S4_R6_pr_minkowski= OrderedDict()

#Function to compute the interpolated precision value at given
    recall value
def interpolatePRvalues(p_arr,r_arr):
    temp_precision_values=[]
    for i in ref_recall:
        for j in range(len(r_arr)):
            if r_arr[j]>=i:
                temp_precision_values.append(p_arr[j])
                break
    return temp_precision_values

#interpolate values for L1 distance measure
for k,v in mean_stddev_L1.items():
    mean_stddev_pr_L1[k]=interpolatePRvalues(v[0],v[1])

for k,v in Gabor_S2_R3_L1.items():
    Gabor_S2_R3_pr_L1[k]=interpolatePRvalues(v[0],v[1])

for k,v in Gabor_S3_R6_L1.items():
    Gabor_S3_R6_pr_L1[k]=interpolatePRvalues(v[0],v[1])

for k,v in Gabor_S4_R6_L1.items():
    Gabor_S4_R6_pr_L1[k]=interpolatePRvalues(v[0],v[1])


#interpolate values for L2 distance measure
for k,v in mean_stddev_L2.items():
    mean_stddev_pr_L2[k]=interpolatePRvalues(v[0],v[1])

for k,v in Gabor_S2_R3_L2.items():
    Gabor_S2_R3_pr_L2[k]=interpolatePRvalues(v[0],v[1])

for k,v in Gabor_S3_R6_L2.items():
    Gabor_S3_R6_pr_L2[k]=interpolatePRvalues(v[0],v[1])

for k,v in Gabor_S4_R6_L2.items():
    Gabor_S4_R6_pr_L2[k]=interpolatePRvalues(v[0],v[1])

#interpolate value for minkowski distance measure
for k,v in mean_stddev_minkowski.items():
    mean_stddev_pr_minkowski[k]=interpolatePRvalues(v[0],v[1])

for k,v in Gabor_S2_R3_minkowski.items():
    Gabor_S2_R3_pr_minkowski[k]=interpolatePRvalues(v[0],v[1])
```

```
for k,v in Gabor_S3_R6_minkowski.items():
    Gabor_S3_R6_pr_minkowski[k]=interpolatePRvalues(v[0],v[1])

for k,v in Gabor_S4_R6_minkowski.items():
    Gabor_S4_R6_pr_minkowski[k]=interpolatePRvalues(v[0],v[1])

#Collect the interpolated precision values for corresponding
   recall values for L1 distance measure
mean_stddev_L1_result=np.zeros(11)
for k,v in mean_stddev_pr_L1.items():
    mean_stddev_L1_result=np.add(mean_stddev_L1_result,v)
for i in range(len(mean_stddev_L1_result)):
    mean_stddev_L1_result[i]=mean_stddev_L1_result[i]/218

Gabor_S2_R3_L1_result=np.zeros(11)
for k,v in Gabor_S2_R3_pr_L1.items():
    Gabor_S2_R3_L1_result=np.add(Gabor_S2_R3_L1_result,v)
for i in range(len(Gabor_S2_R3_L1_result)):
    Gabor_S2_R3_L1_result[i]=Gabor_S2_R3_L1_result[i]/218

Gabor_S3_R6_L1_result=np.zeros(11)
for k,v in Gabor_S3_R6_pr_L1.items():
    Gabor_S3_R6_L1_result=np.add(Gabor_S3_R6_L1_result,v)
for i in range(len(Gabor_S3_R6_L1_result)):
    Gabor_S3_R6_L1_result[i]=Gabor_S3_R6_L1_result[i]/218

Gabor_S4_R6_L1_result=np.zeros(11)
for k,v in Gabor_S4_R6_pr_L1.items():
    Gabor_S4_R6_L1_result=np.add(Gabor_S4_R6_L1_result,v)
for i in range(len(Gabor_S4_R6_L1_result)):
    Gabor_S4_R6_L1_result[i]=Gabor_S4_R6_L1_result[i]/218

#Collect the interpolated precision values for corresponding
   recall values for L2 distance measure
mean_stddev_L2_result=np.zeros(11)
for k,v in mean_stddev_pr_L2.items():
    mean_stddev_L2_result=np.add(mean_stddev_L2_result,v)
for i in range(len(mean_stddev_L2_result)):
    mean_stddev_L2_result[i]=mean_stddev_L2_result[i]/218

Gabor_S2_R3_L2_result=np.zeros(11)
for k,v in Gabor_S2_R3_pr_L2.items():
    Gabor_S2_R3_L2_result=np.add(Gabor_S2_R3_L2_result,v)
for i in range(len(Gabor_S2_R3_L2_result)):
    Gabor_S2_R3_L2_result[i]=Gabor_S2_R3_L2_result[i]/218

Gabor_S3_R6_L2_result=np.zeros(11)
for k,v in Gabor_S3_R6_pr_L2.items():
    Gabor_S3_R6_L2_result=np.add(Gabor_S3_R6_L2_result,v)
for i in range(len(Gabor_S3_R6_L2_result)):
    Gabor_S3_R6_L2_result[i]=Gabor_S3_R6_L2_result[i]/218

Gabor_S4_R6_L2_result=np.zeros(11)
for k,v in Gabor_S4_R6_pr_L2.items():
    Gabor_S4_R6_L2_result=np.add(Gabor_S4_R6_L2_result,v)
for i in range(len(Gabor_S4_R6_L2_result)):
    Gabor_S4_R6_L2_result[i]=Gabor_S4_R6_L2_result[i]/218

#Collect the interpolated precision values for corresponding
   recall values for Minkowski distance measure
mean_stddev_minkowski_result=np.zeros(11)
for k,v in mean_stddev_pr_minkowski.items():
    mean_stddev_minkowski_result=np.add(
       mean_stddev_minkowski_result,v)
for i in range(len(mean_stddev_minkowski_result)):
    mean_stddev_minkowski_result[i]=
       mean_stddev_minkowski_result[i]/218

Gabor_S2_R3_minkowski_result=np.zeros(11)
for k,v in Gabor_S2_R3_pr_minkowski.items():
    Gabor_S2_R3_minkowski_result=np.add(
       Gabor_S2_R3_minkowski_result,v)
for i in range(len(Gabor_S2_R3_minkowski_result)):
    Gabor_S2_R3_minkowski_result[i]=
       Gabor_S2_R3_minkowski_result[i]/218

Gabor_S3_R6_minkowski_result=np.zeros(11)
for k,v in Gabor_S3_R6_pr_minkowski.items():
    Gabor_S3_R6_minkowski_result=np.add(
       Gabor_S3_R6_minkowski_result,v)
for i in range(len(Gabor_S3_R6_minkowski_result)):
    Gabor_S3_R6_minkowski_result[i]=
       Gabor_S3_R6_minkowski_result[i]/218

Gabor_S4_R6_minkowski_result=np.zeros(11)
for k,v in Gabor_S4_R6_pr_minkowski.items():
    Gabor_S4_R6_minkowski_result=np.add(
       Gabor_S4_R6_minkowski_result,v)
for i in range(len(Gabor_S4_R6_minkowski_result)):
    Gabor_S4_R6_minkowski_result[i]=
       Gabor_S4_R6_minkowski_result[i]/218
```

4.7 Plot the Interpolated Precision-Recall Curve

```
#Plot the Interpolated Precision-Recall Curve
```

```
fig=plt.figure(figsize=(16,10))
plt.plot(ref_recall ,Gabor_S2_R3_L1_result ,'*b-',linewidth=1,
   markersize=8,label="Gabor(2,3) L1")
plt.plot(ref_recall ,Gabor_S2_R3_L2_result ,'d-',color="#4363d8
   ",markersize=8,linewidth=1,label="Gabor(2,3) L2")
plt.plot(ref_recall ,Gabor_S2_R3_minkowski_result,'<-',color="#
   f58231",markersize=8,linewidth=1,label="Gabor(2,3) Minkowski
   ")
plt.plot(ref_recall ,Gabor_S3_R6_L1_result ,'*g-',linewidth=1,
   markersize=8,label="Gabor(3,6) L1")
plt.plot(ref_recall ,Gabor_S3_R6_L2_result ,'d-',color
   ="#808000",markersize=8,linewidth=1,label="Gabor(3,6) L2")
plt.plot(ref_recall ,Gabor_S3_R6_minkowski_result,'<-',color
   ="#000000",markersize=8,linewidth=1,label="Gabor(3,6)
   Minkowski")
plt.plot(ref_recall ,Gabor_S4_R6_L1_result ,'*c-',linewidth=1,
   markersize=8,label="Gabor(4,6) L1")
plt.plot(ref_recall ,Gabor_S4_R6_L2_result ,'d-',color
   ="#469990",markersize=8,linewidth=1,label="Gabor(4,6) L2")
plt.plot(ref_recall ,Gabor_S4_R6_minkowski_result,'<-',color
   ="#9A6324",markersize=8,linewidth=1,label="Gabor(4,6)
   Minkowski")
plt.plot(ref_recall ,mean_stddev_L1_result ,'*r-',linewidth=1,
   markersize=8,label="Mean StdDev L1")
plt.plot(ref_recall ,mean_stddev_L2_result ,'d-',color
   ="#800000",markersize=8,linewidth=1,label="Mean StdDev L2")
plt.plot(ref_recall ,mean_stddev_minkowski_result,'<-',color
   ="#911eb4",markersize=8,linewidth=1,label="Mean StdDev
   Minkowski")
plt.legend(fancybox=True, framealpha=1, shadow=True, borderpad
   =1,fontsize=15)
plt.title("Recall Vs Precision Plot",fontsize=20)
plt.xlabel("Recall",fontsize=20)
plt.ylabel("Precision",fontsize=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.savefig(currentWorkingDirectory+"/"+"PrecisionRecall_Curve
   .png")
plt.show()
```

4.8 Create Table to show Average Precisions Values

```
#create Table to show Average Precisions Values
import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties
from operator import itemgetter
fig, ax =plt.subplots(1,1)
data=[['Mean & Standard Deviation with L1 Distance', np.mean(
    mean_stddev_L1_result)],
        ['Mean & Standard Deviation with L2 Distance', np.mean(
            mean_stddev_L2_result)],
        ['Mean & Standard Deviation with Minkowski Distance', np
            .mean(mean_stddev_minkowski_result)],
        ['Gabor 2 Scale & 3 Orientation with L1 Distance ', np.
            mean(Gabor_S2_R3_L1_result)],
        ['Gabor 2 Scale & 3 Orientation with L2 Distance', np.
            mean(Gabor_S2_R3_L2_result)],
        ['Gabor 2 Scale & 3 Orientation with Minkowski Distance
            ', np.mean(Gabor_S2_R3_minkowski_result)],
        ['Gabor 3 Scale & 6 Orientation with L1 Distance ', np.
            mean(Gabor_S3_R6_L1_result)],
        ['Gabor 3 Scale & 6 Orientation with L2 Distance ', np.
            mean(Gabor_S3_R6_L2_result)],
        ['Gabor 3 Scale & 6 Orientation with Minkowski Distance
            ', np.mean(Gabor_S3_R6_minkowski_result)],
        ['Gabor 4 Scale & 6 Orientation with L1 Distance', np.
            mean(Gabor_S4_R6_L1_result)],
        ['Gabor 4 Scale & 6 Orientation with L2 Distance ', np.
            mean(Gabor_S4_R6_L2_result)],
        ['Gabor 4 Scale & 6 Orientation with Minkowski Distance
            ', np.mean(Gabor_S4_R6_minkowski_result)],

      ]

data.sort(key=itemgetter(1))
data.reverse()
#print(data)
column_labels=["Configurations", "Average Precision Values"]
ax.axis('off')
table = ax.table(cellText=data, colWidths=[0.5] * 3,
    colLabels=column_labels, colColours=["yellow"] * 2, loc="
    center",)
table.scale(2, 2)
table.set_fontsize(15)


for (row, col), cell in table.get_celld().items():
    if (row == 0 or col ==0):
        cell.set_text_props(fontproperties=FontProperties(
            weight='bold'))

plt.savefig(currentWorkingDirectory+"/"+"AveragePrecisionTable
    .png")
plt.show()
```