COSC522: Final Project - Catifier

Cameron Adkins, Purnachandra Anirudh Gajjala, Gabriel Abeyie

In [1]:
```
# Numpy.
import numpy as np
from numpy.lib.stride_tricks import sliding_window_view

# Need plots.
import matplotlib.pyplot as plt

# Pandas.
import pandas as pd

# Machine learning toolkit.
from sklearn.model_selection import KFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import *
from sklearn.preprocessing import Normalizer
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report, accuracy_score

# Scipy for fft's and the like.
import scipy as sc
import scipy.io.wavfile as wavfile
from scipy import signal
from scipy.fftpack import fft, fftfreq
from scipy import stats

# Seaborn for plots.
import seaborn as sns

# IPython for basic visual output types.
import IPython

# Imbalanced learn for rebalancing.
from imblearn.over_sampling import SMOTE

# Standard Python libs.
import os
import glob
import csv
import xml.etree.ElementTree as et
from dataclasses import dataclass

# Pillow.
import PIL as pil
from PIL import ImageDraw

# Tensorflow
#import tensorflow as tf
#from tensorflow.keras.preprocessing.image import ImageDataGenerator
#from tensorflow.keras.datasets import fashion_mnist
#from tensorflow.keras.layers import Dense
#from tensorflow.keras.optimizers import Adam
#from tensorflow.keras.layers import Conv2D, Flatten, Dense, AveragePooling2D, GlobalAveragePooling2D,Dropout
#from tensorflow.keras.applications.resnet import ResNet50
#from tensorflow.keras.models import Sequential
#from tensorflow.keras.preprocessing.image import ImageDataGenerator

# For images
from skimage.color import rgb2gray
import cv2
from scipy import ndimage
```

In [2]:
```python
# Utility functions.

def image_load(filename):
    loader = pil.Image.open(filename);
    ret = loader.copy();
    loader.close();

    return ret;

def image_resize(image, new_width):
    new_height = int(new_width * (image.height / image.width));

    return image.resize((new_width, new_height), pil.Image.Resampling.LANCZOS);

def trimap_to_mask(trimap, include_border = True):
    trimap_data = trimap.getdata();

    mask_data = np.zeros((trimap.height, trimap.width), dtype=np.uint8);

    for x in range(0, trimap.width):
        for y in range(0, trimap.height):
            idx = x + (y * trimap.width);
            tri = trimap_data[idx];

            if (tri == 1 or (include_border == True and tri == 3)):
                mask_data[y, x] = 255;
            else:
                mask_data[y, x] = 0;

    mask = pil.Image.fromarray(mask_data);

    return mask;
```

In [3]:

```python
# Class definitions.

@dataclass
class Point:
    x: int;
    y: int;

@dataclass
class BoundingBox:
    ll: Point;
    lr: Point;
    ul: Point;
    ur: Point;

class CatBreedSample:
    def __init__(self, label, image_file, mask_file = None, bb_file = None):
        self.label = label;

        self.image_file = image_file;
        self.mask_file  = mask_file;
        self.bb_file    = bb_file;

        # Load the image
        self.image = image_load(self.image_file);

        # Composite if a mask is available.
        if (self.mask_file):
            self.mask = trimap_to_mask(image_load(self.mask_file), False);

            background = pil.Image.new("RGB", self.mask.size, 0);
            self.masked_image = pil.Image.composite(self.image, background, self.mask);
        else:
            self.mask = None;
            self.masked_image = None;

        # Get a bounding box.
        if (self.bb_file):
            tree = et.parse(self.bb_file);
            root = tree.getroot();

            xmin = int(root.findall("./object/bndbox/xmin")[0].text);
            xmax = int(root.findall("./object/bndbox/xmax")[0].text);
            ymin = int(root.findall("./object/bndbox/ymin")[0].text);
            ymax = int(root.findall("./object/bndbox/ymax")[0].text);

            self.bb = BoundingBox(0, 0, 0, 0);

            self.bb.ll = Point(xmin, ymin);
            self.bb.lr = Point(xmax, ymin);
            self.bb.ul = Point(xmin, ymax);
            self.bb.ur = Point(xmax, ymax);

            if (self.masked_image):
                self.bounded_image = self.masked_image.crop((xmin, ymin, xmax, ymax));
            else:
                self.bounded_image = self.image.crop((xmin, ymin, xmax, ymax));

        else:
            self.bb = None;
            self.bounded_image = None;

        self.image = image_resize(self.image, 256);
        self.masked_image = image_resize(self.masked_image, 256);
        self.bounded_image = image_resize(self.bounded_image, 256);

    def display(self):
        print("Image:", self.image_file);
        print("Label:", self.label);

        display(self.image);
```

```
        if (self.mask):
            display(self.masked_image);

        if (self.bb):
            display(self.bounded_image);

    def features(self):
        fv = [];

        # Determine eye positions.

        # Determine ear positions.
        # We just estimate by starting at corners of the region segmentation and working
        # our way towards the center. The first non-black pixel is our x/y.
        region_segmentation = self.segmentation_regionbased();

        #Left

        left_ear = (0, 0);

        #print(len(region_segmentation), len(region_segmentation[0]))
        #print(seg_data[0], seg_data[1], seg_data[2], seg_data[3]);

        for y in range(0, int(len(region_segmentation) / 2)):
            for x in range(0, int(len(region_segmentation[0]) / 2)):
                #print(region_segmentation[y, x], end = " ")
                if (region_segmentation[x, y] != 0):
                    #print(x, y, "=", region_segmentation[y, x]);
                    left_ear = (y, x);
                    break;

            if (left_ear != (0, 0)):
                break;

        #print(left_ear);

        #Right

        right_ear = (0, 0);

        for y in range(0, int(len(region_segmentation) / 2)):
            for x in reversed(range(int(len(region_segmentation[0]) / 2), int(len(region_segmentation[0])))):
                #print(region_segmentation[y, x], end = " ")
                if (region_segmentation[y, x] != 0):
                    #region_segmentation[y, x] = 0.5;
                    right_ear = (x, y);
                    break;

            if (right_ear != (0, 0)):
                break;

        #print(right_ear);

        #plt.imshow(region_segmentation, cmap='gray');

        #draw = ImageDraw.Draw(self.bounded_image);
        #ellipse = ((left_ear[0] - 2, left_ear[1] - 2, left_ear[0] + 2, left_ear[1] + 2));
        #draw.ellipse(ellipse, fill = "red", outline = "red");
        #ellipse = ((right_ear[0] - 2, right_ear[1] - 2, right_ear[0] + 2, right_ear[1] + 2));
        #draw.ellipse(ellipse, fill = "red", outline = "red");

        fv.append(right_ear[0] - left_ear[0]);
        fv.append(right_ear[1] - left_ear[1]);

        # Bin the colors that make up this image.
        #region_segmentation = self.segmentation_colorclustering();
        #unique, counts = np.unique(region_segmentation, return_counts = True);

        #unique_colors = dict(zip(unique, counts));

        #counts = counts[1:];
        #_, counts = np.unique(counts, return_counts = True);
```

```python
        cv2image = np.array(self.masked_image);

        channels = cv2.split(cv2image);

        for i in range(0, 3):
            hist = cv2.calcHist([channels[i]], [0], None, [5], [0, 255]);
            for val in hist:
                fv.append(int(val));
            #print(hist);
        #plt.figure();
        #plt.plot(hist);
        #plt.xlim([0, 256])
        #plt.ylim([0, 1200])
        #region_segmentation

        #print(unique);

        return fv;

    def segmentation_regionbased(self):
        gray = rgb2gray(np.array(self.bounded_image));
        #plt.imshow(gray, cmap = 'gray')

        gray_r = gray.reshape(gray.shape[0]*gray.shape[1])

        for i in range(gray_r.shape[0]):

            if gray_r[i] > gray_r.mean():

                gray_r[i] = 1

            else:

                gray_r[i] = 0

        gray = gray_r.reshape(gray.shape[0],gray.shape[1])
        #plt.figure();
        #plt.imshow(gray, cmap='gray')

        return gray;

        # The darker region (black) represents the background and the brighter (white) region is the foreground. We can define multiple thresholds as well to detect multiple ob

        # gray_r = gray.reshape(gray.shape[0]*gray.shape[1])

        #for i in range(gray_r.shape[0]):
            #if gray_r[i] > gray_r.mean():
                #gray_r[i] = 3
            #elif gray_r[i] > 0.5:
                #gray_r[i] = 2
            #elif gray_r[i] > 0.25:
                #gray_r[i] = 1
            #else:
                #gray_r[i] = 0

        #gray = gray_r.reshape(gray.shape[0],gray.shape[1])
        #plt.imshow(gray, cmap='gray')

    def segmentation_edgebased(self):
        gray = rgb2gray(np.array(self.bounded_image));

        #plt.figure();
        #plt.imshow(gray, cmap='gray');

        # defining the sobel filters

        # [
        #  [ 1  2  1]
        #  [ 0  0  0]
        #  [-1 -2 -1]
        # ]
```

```
    # ]
    # is a kernel for detecting horizontal edges

    # [
    #  [-1  0  1]
    #  [-2  0  2]
    #  [-1  0  1]
    # ]
    # is a kernel for detecting vertical edges

    sobel_horizontal = np.array([np.array([1, 2, 1]), np.array([0, 0, 0]), np.array([-1, -2, -1])])
    print('Kernel for detecting horizontal edges:\n', sobel_horizontal)

    sobel_vertical = np.array([np.array([-1, 0, 1]), np.array([-2, 0, 2]), np.array([-1, 0, 1])])
    print('Kernel for detecting vertical edges:\n', sobel_vertical)

    out_h = ndimage.convolve(gray, sobel_horizontal, mode='reflect')
    out_v = ndimage.convolve(gray, sobel_vertical, mode='reflect')

    # here mode determines how the input array is extended when the filter overlaps a border.

    plt.figure();
    plt.imshow(out_h, cmap='gray')
    plt.imshow(out_v, cmap='gray')

    # Here, we are able to identify the horizontal as well as the vertical edges. There is one more type of filter that can detect both horizontal and vertical edges at the

    # [
    #  [1  1  1]
    #  [1 -8  1]
    #  [1  1  1]
    # ]

    kernel_laplace = np.array([np.array([1, 1, 1]), np.array([1, -8, 1]), np.array([1, 1, 1])])
    print("Laplacian kernel:\n", kernel_laplace)

    out_l = ndimage.convolve(gray, kernel_laplace, mode='reflect')
    plt.figure();
    plt.imshow(out_l, cmap='gray')

def segmentation_colorclustering(self):
    # According to wikipedia the R, G, and B components of an object’s color in a digital image are all correlated with the amount of light hitting the object,
    # and therefore with each other, image descriptions in terms of those components make object discrimination difficult.
    # Descriptions in terms of hue/lightness/chroma or hue/lightness/saturation are often more relevant. So, we need to convert our image from RGB Colours Space to HSV to wo

    cv2img = np.array(self.masked_image);

    vectorized = np.float32(cv2img.reshape((-1,3)))
    vectorized.shape

    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 20, 1.0)

    K = 8
    attempts=10
    ret, label, center = cv2.kmeans(vectorized, K, None, criteria, attempts, cv2.KMEANS_PP_CENTERS)

    center = np.uint8(center)
    res = center[label.flatten()]
    result_image = res.reshape((cv2img.shape))

    # result_image is the output result. Lets see how the image looks after k-means clustering

    #figure_size = 15
    #plt.figure(figsize=(figure_size, figure_size))
    #plt.subplot(2,3,1),plt.imshow(cv2img)
    #plt.title('Original Image'), plt.xticks([]), plt.yticks([])
    #plt.subplot(2,3,2),plt.imshow(result_image)
    #plt.title('Segmented Image when K = %i' % K), plt.xticks([]), plt.yticks([])
    #plt.show()

    return result_image;
```

```python
    def haarcascade_classifier(self):
        cv2img  = np.array(self.image);
        cv2gray = cv2.cvtColor(cv2img, cv2.COLOR_RGB2GRAY);

        cascade = cv2.CascadeClassifier("pretrained/haarcascade_frontalcatface.xml");

        bounding_rects = cascade.detectMultiScale(cv2gray, scaleFactor = 1.3, minNeighbors = 1, minSize = (25, 25));

        print(bounding_rects);

        for (x, y, w, h) in bounding_rects:
            cv2.rectangle(cv2gray, (x, y), (x + w, y + h), (0, 0, 255), 2);

        plt.figure();
        plt.imshow(cv2gray, cmap='gray');

    def contours(self):
        #region_segmentation = self.segmentation_regionbased();
        region_segmentation = np.array(self.bounded_image);
        region_segmentation = cv2.cvtColor(region_segmentation, cv2.COLOR_RGB2GRAY)


        ret, threshold = cv2.threshold(region_segmentation, 16, 255, cv2.THRESH_BINARY);
        image, contours, hierarchy = cv2.findContours(threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE);


        cv2.drawContours(threshold, contours, -1, (0, 255, 0), 3);

        approx = cv2.approxPolyDP(contours[1], 0.01 * cv2.arcLength(contours[1], True), True)

        print(approx);

        plt.figure();
        plt.imshow(threshold);

        #i = 0;

        #for contour in contours:
        #    # here we are ignoring first counter because
        #    # findcontour function detects whole image as shape
        #    if i == 0:
        #        i = 1
        #        continue
        #
        #    # cv2.approxPloyDP() function to approximate the shape
        #    #approx = cv2.approxPolyDP(contour, 0.01 * cv2.arcLength(contour, True), True)
        #
        #    print(contour)
        #
        #    # using drawContours() function
        #    cv2.drawContours(region_segmentation, [contour], 0, (0, 0, 255), 5)

        #plt.figure();
        #plt.imshow(region_segmentation);
```

In [4]:

```python
#test_sample = CatBreedSample(
#    "Abyssinian",
#    "/home/cva/catifier/training_data/Abyssinian/Abyssinian_115.jpg",
#    "/home/cva/catifier/training_data/Abyssinian/Abyssinian_115_mask.png",
#    "/home/cva/catifier/training_data/Abyssinian/Abyssinian_115_bb.xml"
#)

#test_sample.segmentation_regionbased();
#test_sample.segmentation_edgebased();
#test_sample.segmentation_colorclustering();
#test_sample.haarcascade_classifier();
#test_sample.contours();

#fv = test_sample.features();

#test_sample.display();
```

In [5]:
```
# Load everything.
def load_samples(samples_dir):
    class_dirs = glob.glob(samples_dir + "/*")

    samples = [];

    for class_dir in class_dirs:
        label = os.path.basename(class_dir);
        class_dir_glob = glob.glob(class_dir + "/*.jpg");

        print("Reading files for label '" + label + "'");

        for sample_image in class_dir_glob:
            basename  = os.path.splitext(sample_image)[0];
            mask_file = basename + "_mask.png";
            bb_file   = basename + "_bb.xml";

            if (not os.path.exists(mask_file)):
                mask_file = None;

            if (not os.path.exists(bb_file)):
                continue;
                #bb_file = None;

            print(
                len(samples),
                ":",
                os.path.basename(sample_image),
                "\t(mask:",
                (mask_file != None),
                "| bb:",
                (bb_file != None),
                ")"
            );

            sample = CatBreedSample(label, sample_image, mask_file, bb_file);
            samples.append(sample);

    return samples;

PWD = os.getcwd();
TRAINING_DATA = PWD + "/training_data";
samples = load_samples(TRAINING_DATA);
```

```
(... trimmed for brevity ...)
Reading files for label 'Abyssinian'

Reading files for label 'Sphynx'

Reading files for label 'Egyptian_Mau'

Reading files for label 'Persian'

Reading files for label 'Siamese'

Reading files for label 'Birman'

Reading files for label 'Bombay'

Reading files for label 'Russian_Blue'

Reading files for label 'Bengal'

Reading files for label 'Maine_Coon'

Reading files for label 'Ragdoll'

Reading files for label 'British_Shorthair'
```

In [6]:

```
#sample_id = 120;
#samples[sample_id].display();
#samples[sample_id].segmentation_regionbased();
#samples[sample_id].segmentation_edgebased();
#samples[sample_id].segmentation_colorclustering();
```

In [8]:

```
# Try the model.
fv = [];
labels = [];

for sample in samples:
    print("Extract features from:", sample.image_file);
    fv.append(sample.features());
    labels.append(sample.label);

# Rebalance.
oversampler = SMOTE();
(fv, labels) = oversampler.fit_resample(fv, labels);

# Scale.
scaler = RobustScaler();
fv = scaler.fit_transform(fv);

# Split the data.
x_train, x_test, y_train, y_test = train_test_split(fv, labels, test_size = 0.30, random_state = 64);

# Train.
dt = RandomForestClassifier();
dt.fit(x_train, y_train);

# Testing the model.
cv_scores = cross_val_score(dt, x_train, y_train, cv = 10);

print('Average Cross Validation Score from Training:', cv_scores.mean(), sep = '\n', end = '\n\n\n');

y_pred = dt.predict(x_test);
cm = confusion_matrix(y_test, y_pred);
cr = classification_report(y_test, y_pred);

print('Confusion Matrix:', cm, sep = '\n', end = '\n\n\n');
print('Missing classifications (if any):', set(y_test) - set(y_pred));
print('Test Statistics:', cr, sep = '\n', end = '\n\n\n');
print('Testing Accuracy:', accuracy_score(y_test, y_pred));
```

```
(... omit loading messages for brevity ...)
Average Cross Validation Score from Training:
0.3547619047619047


Confusion Matrix:
[[10  6  1  1  1  1  3  1  0  2  1  3]
 [ 4 10  2  3  0  0  5  2  0  0  1  0]
 [ 0  1  6  1  2  4  1  2  6  1  5  1]
 [ 0  0  0 24  1  0  1  0  0  1  0  0]
 [ 0  2  1  1 15  1  5  3  3  4  0  0]
 [ 1  1  2  0  0  7  0  0  0  5  7  2]
 [ 5  7  2  1  0  0  4  4  2  1  3  1]
 [ 4  1  0  0  1  0  1  9  4  2  1  3]
 [ 3  0  4  0  1  4  0  5  6  0  5  3]
 [ 1  1  0  2  8  4  1  1  1 11  0  0]
 [ 0  5  7  1  0  3  7  1  3  0  5  3]
 [10  1  3  0  2  3  2  1  1  1  2  8]]


Missing classifications (if any): set()
Test Statistics:
                   precision    recall  f1-score   support

       Abyssinian       0.26      0.33      0.29        30
           Bengal       0.29      0.37      0.32        27
           Birman       0.21      0.20      0.21        30
           Bombay       0.71      0.89      0.79        27
British_Shorthair       0.48      0.43      0.45        35
      Egyptian_Mau      0.26      0.28      0.27        25
       Maine_Coon       0.13      0.13      0.13        30
          Persian       0.31      0.35      0.33        26
          Ragdoll       0.23      0.19      0.21        31
     Russian_Blue       0.39      0.37      0.38        30
          Siamese       0.17      0.14      0.15        35
           Sphynx       0.33      0.24      0.28        34

         accuracy                           0.32       360
        macro avg       0.31      0.33      0.32       360
     weighted avg       0.31      0.32      0.31       360



Testing Accuracy: 0.3194444444444444
```

In [ ]: