

# ECE 271A Quiz 1

1<sup>st</sup> Yifan Wu  
yiw084@ucsd.edu

**Abstract**—The goal of this problem is to segment the “cheetah” image into its two components, cheetah (foreground) and grass (background).

## I. PROBLEMS

A.

Using the training data in TrainingSamplesDCT 8.mat, what are reasonable estimates for the prior probabilities?

There are two matrices, TrainsampleDCT\_BG and TrainsampleDCT\_FG for foreground (cheetah) and background (grass) samples respectively.

The prior:

$$P_{Y=Cheetah} = \frac{n_{FG}}{n_{FG} + n_{BG}} = 0.8081350729086723 \quad (1)$$

$$P_{Y=grass} = \frac{n_{BG}}{n_{FG} + n_{BG}} = 0.1918649270913277 \quad (2)$$

B.

Using the training data in TrainingSamplesDCT 8.mat, compute and plot the index histograms  $P_{X|Y}(x|cheetah)$  and  $P_{X|Y}(x|grass)$ .

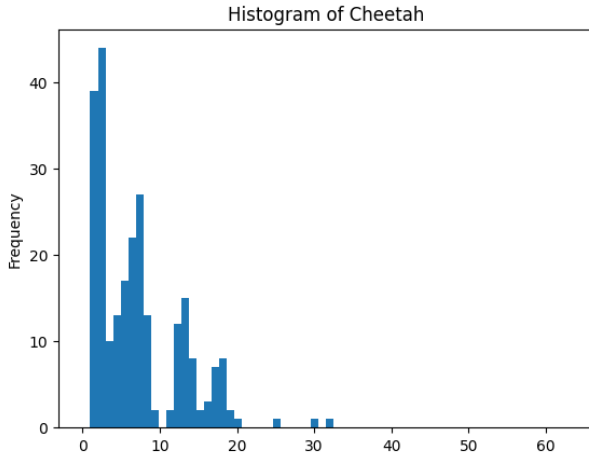


Fig. 1. Histogram of  $P_{X|Y}(x|cheetah)$

C.

For each block in the image cheetah.bmp, compute the feature X (index of the DCT coefficient with 2nd greatest energy). Compute the state variable Y using the minimum probability of error rule based on the probabilities obtained in a) and b). Store the state in an array A. Using the commands imagesc and colormap(gray(255)) create a picture of that array.

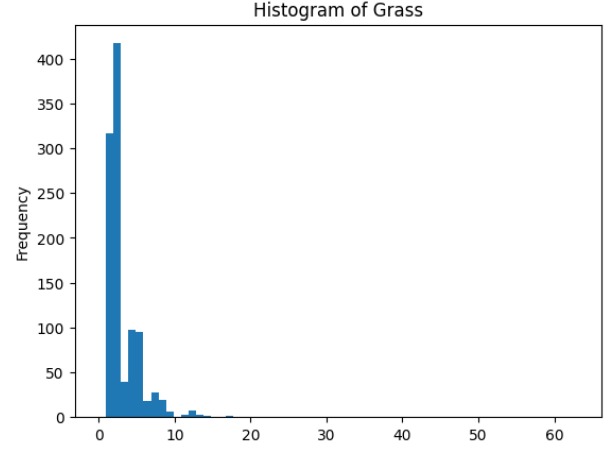


Fig. 2. Histogram of  $P_{X|Y}(x|grass)$

Denote class “Cheetah” as  $FG$  and class “Grass” as  $BG$ . Using MAP rule,

$$Y = \begin{cases} 1(\text{Cheetah}) & \text{if } P_{X|Y}(x|FG)P_{FG} \geq P_{X|Y}(x|BG)P_{BG} \\ 0(\text{Grass}) & \text{Otherwise} \end{cases} \quad (3)$$

D.

The array A contains a mask that indicates which blocks contain grass and which contain the cheetah. Compare it with the ground truth provided in image cheetah mask.bmp and compute the probability of error of your algorithm.

Denote ground truth as  $T$ , we are going in to trim the ground truth image to have the same shape as  $A$  with height  $M$  and weight  $N$ . Since we are working with mask image, in each sample:

$$correct = \begin{cases} True & \text{if } T_{ij} - A_{ij} = 0 \quad \forall i \in M \quad \forall j \in N \\ False & \text{if } |T_{ij} - A_{ij}| = 1 \quad \forall i \in M \quad \forall j \in N \end{cases} \quad (4)$$

$$\text{The probability of error} = \frac{\text{Total number of wrong predictions}}{\text{Total number of samples}} \quad (5)$$

- The probability of error: **0.17544155525573413**
- The probability of error in foreground: **0.15150864712375814**
- The probability of error in background: **0.02393290813197596**

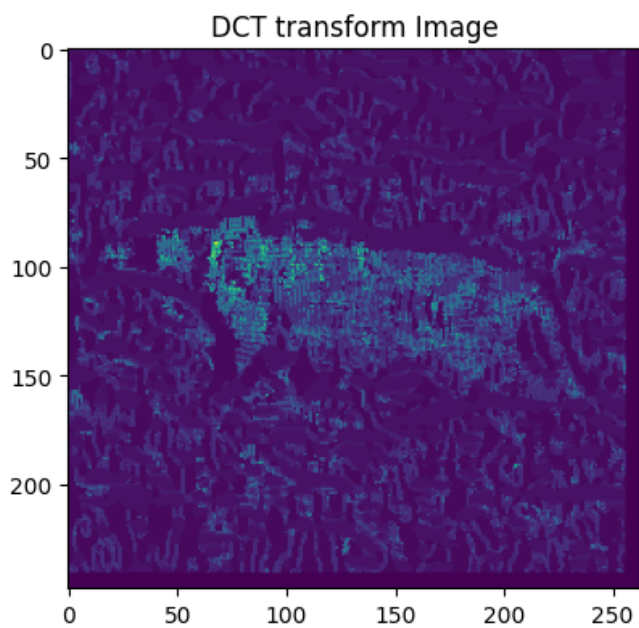


Fig. 3. *DCT Transformed Image*

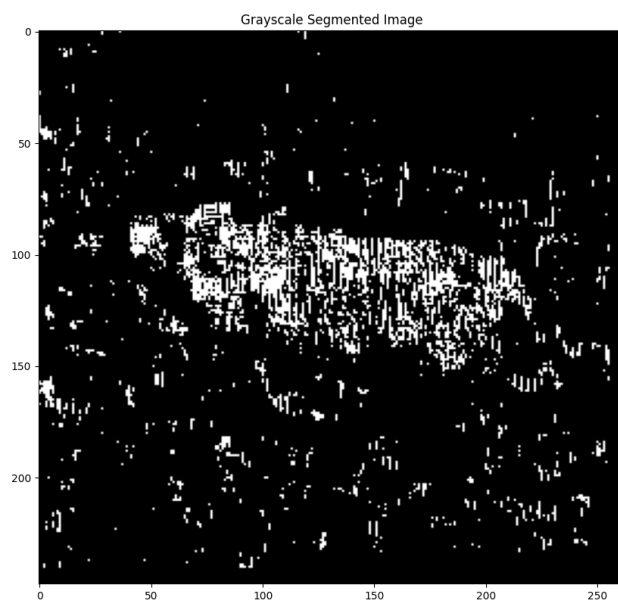


Fig. 5. *Gray-scale Segment*

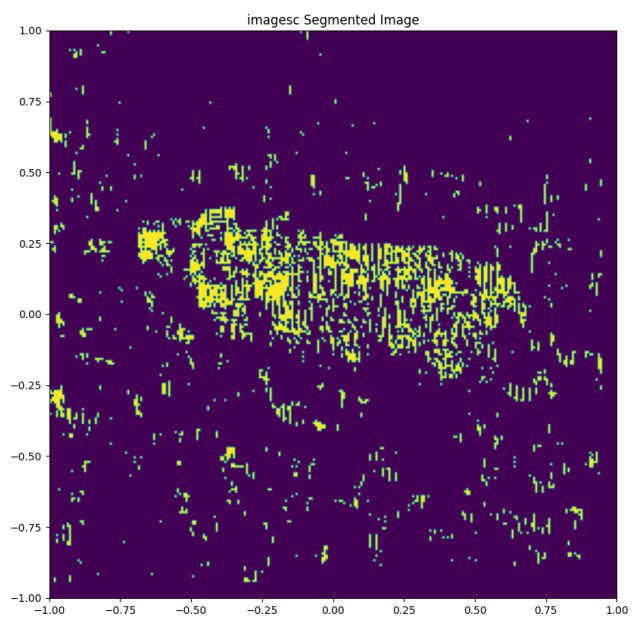


Fig. 4. *Imagesc Segment*

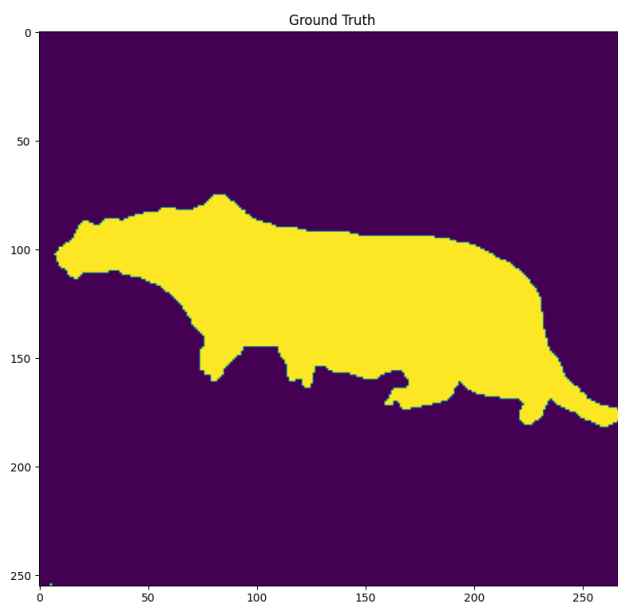


Fig. 6. *Ground Truth*

```
import argparse
import os
import pathlib
from typing import Tuple

import numpy as np
import scipy.io as sio
from scipy.fftpack import dct
import matplotlib.pyplot as plt
from PIL import Image

try:
    from icecream import install # noqa

    install()
except ImportError: # Graceful fallback if IceCream isn't installed.
    ic = lambda *a: None if not a else (a[0] if len(a) == 1 else a) # noqa

def get_2nd_largest(x, axis=1):
    """
    Get the index of second largest energy value in data
    """
    ind = np.argmax(abs(x), axis) + 1
    return ind

def plot_hist(
    input,
    n_bins: int,
    ranges: Tuple[int, int],
    title: str,
    save_path: str = "",
    show: bool = False,
):
    """
    plot the histogram of the input data
    """
    plt.figure()
    hist, bin_edges, *_ = plt.hist(input, bins=n_bins, range=ranges)
    plt.title(title)
    plt.ylabel("Frequency")
    if save_path:
        plt.savefig(save_path, bbox_inches="tight")
        plt.close()
    elif show:
        plt.show()
    else:
        plt.show()
        plt.close()
    return hist, bin_edges
```

```

def find_2nd_largest_index(data, pattern):
    data[0, 0] = 0
    num = pattern.reshape(-1)
    index = np.argmax(abs(data))
    second_largest_index = num[index]
    return second_largest_index

def MAP_rule(data, prob_cheetah, prob_grass, cheetah_prior, grass_prior):
    """
    Compute the MAP rule for the image where cheetah = 1 and grass = 0
    """
    img = np.zeros_like(data)

    prior = grass_prior / cheetah_prior

    for i in range(data.shape[0]):
        for j in range(data.shape[1]):
            index = int(data[i, j])
            P_FG = prob_cheetah[index]
            P_BG = prob_grass[index]
            if P_FG * cheetah_prior > P_BG * grass_prior:
                img[i, j] = 1
            else:
                img[i, j] = 0
    return img

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--plot", action="store_true")
    args = parser.parse_args()

    current_dir = pathlib.Path(__file__).parent.resolve()
    data_dir = current_dir / "data"
    mat_fname = data_dir / "TrainingSamplesDCT_8.mat"
    zig_fname = data_dir / "Zig-Zag Pattern.txt"
    plot_dir = current_dir / "plots"

    for d in [data_dir, plot_dir]:
        if not os.path.exists(d):
            os.mkdir(d)

    pattern = np.loadtxt(zig_fname, dtype=np.int64)

    mat_contents = sio.loadmat(mat_fname)
    TrainsampleDCT_BG = mat_contents["TrainsampleDCT_BG"]
    TrainsampleDCT_FG = mat_contents["TrainsampleDCT_FG"]
    print(f"The amount of FG data: {TrainsampleDCT_FG.shape[0]}")
    print(f"The amount of BG data: {TrainsampleDCT_BG.shape[0]}")

    m_cheetah, n_cheetah = np.shape(TrainsampleDCT_BG)

```

```

m_grass, n_grass = np.shape(TrainsampleDCT_FG)

# (a)
P_cheetah = m_cheetah / (m_cheetah + m_grass)
P_grass = m_grass / (m_cheetah + m_grass)

assert P_cheetah + P_grass == 1
print(f"The prior P_Y_cheetah: {P_cheetah}")
print(f"The prior P_Y_grass: {P_grass}")

# (b)
cheetah_index = get_2nd_largest(TrainsampleDCT_FG[:, 1:])
grass_index = get_2nd_largest(TrainsampleDCT_BG[:, 1:])

cheetah_hist, cheetah_bin_edges = plot_hist(
    cheetah_index,
    n_bins=n_cheetah,
    ranges=(0, n_cheetah - 1),
    title="Histogram of Cheetah",
    save_path=plot_dir / "hist_FG",
    show=args.plot,
)
grass_hist, grass_bin_edge = plot_hist(
    grass_index,
    n_bins=n_grass,
    ranges=(0, n_grass - 1),
    title="Histogram of Grass",
    save_path=plot_dir / "hist_BG",
    show=args.plot,
)

# (c)
# fig = plt.figure(figsize=(10,10))
img = Image.open(str(data_dir / "cheetah.bmp"), "r")
img = np.asarray(img)
img = img.astype(np.float64) / 255
assert img.min() == 0 and img.max() <= 1

processed_img = np.zeros([img.shape[0] - 7, img.shape[1] - 7],
dtype="uint8")

for i in range(processed_img.shape[0] - 7):
    for j in range(processed_img.shape[1] - 7):
        # 8 x 8 block
        block = img.copy()[i : i + 8, j : j + 8]
        # DCT transform on the block
        block_DCT = dct(dct(block.T, norm="ortho").T, norm="ortho")
        index = find_2nd_largest_index(block_DCT, pattern)
        processed_img[i, j] = index
plt.imshow(processed_img)
plt.title("DCT transform Image")

# P_X|Cheetah
prob_cheetah = cheetah_hist / m_cheetah

```

```

# P_X|Grass
prob_grass = grass_hist / m_grass

A = MAP_rule(processed_img, prob_cheetah, prob_grass, P_cheetah,
P_grass)

# equivalent to imagesc
plt.figure(figsize=(10, 10))
plt.imshow(A, extent=[-1, 1, -1, 1])
plt.title("imagesc Segmented Image")

# equivalent to colormap(gray(255))
plt.figure(figsize=(10, 10))
plt.imshow(A, cmap="gray")
plt.title("Grayscale Segmented Image")
plt.show()

# (d)
ground_truth = Image.open(str(data_dir / "cheetah_mask.bmp"), "r")
ground_truth = np.asarray(ground_truth)
plt.imshow(ground_truth)
plt.title("Ground Truth")
plt.show()
# Truncate ground truth to have same size as segmented image
ground_truth = ground_truth[: A.shape[0], : A.shape[1]] / 255

# calculate the error
error = 1 - np.sum(ground_truth == A) / A.size
print(f"The probability of error: {error}")

# error in the FG
error_idx = np.where((ground_truth - A) == 1)[0]
FG_error = len(error_idx) / A.size
print(f"FG error: {FG_error}")

# error in the BG
error_idx = np.where((ground_truth - A) == -1)[0]
BG_error = len(error_idx) / A.size
print(f"BG error is: {BG_error}")

plt.gcf().canvas.mpl_connect(
    "key_release_event",
    lambda event: [plt.close() if event.key in ["escape", "Q"] else
None],
)
plt.show()

```