In [1]:
```python
import os
import pathlib
from hw2_soln import *

%matplotlib inline
```

Problem a)

Using the training data in TrainingSamplesDCT 8.mat compute the histogram estimate of the prior $P_Y(i), i \in \{cheetah, grass\}$. Using the results of problem 2 compute the maximum likelihood estimate for the prior probabilities. Compare the result with the estimates that you obtained last week. If they are the same, interpret what you did last week. If they are different, explain the differences.

Answer for problem a)

$$\pi_i = \frac{C_i}{n}$$

$$\pi_1(Cheetah) = \frac{C_1}{n} = \frac{250}{250 + 1053} = 0.1918649270913277$$

$$\pi_2(Grass) = \frac{C_i}{n} = \frac{1053}{250 + 1053} = 0.8081350729086723$$

Last week, we calculate the prior based on the frequency of the occurancy of each class in the training set. This is the same as the maximum likelihood estimate.

Code Answers form HW1

In [2]:
```python
data_dir = os.path.join(os.getcwd(), 'data')
plot_dir = os.path.join(os.getcwd(), 'plots')
data_dir = pathlib.Path(data_dir)
old_mat_fname = data_dir / "TrainingSamplesDCT_8.mat"

load_and_compute_prior(old_mat_fname)
```

```
The prior P_Y_cheetah from HW1: 0.1918649270913277
The prior P_Y_grass from HW1: 0.8081350729086723
```

Answers from HW2 a)

In [3]:
```python
mat_contents = sio.loadmat(data_dir / "TrainingSamplesDCT_8_new.mat")
TrainsampleDCT_BG = mat_contents["TrainsampleDCT_BG"]
TrainsampleDCT_FG = mat_contents["TrainsampleDCT_FG"]
```

In [4]:
```python
m_FG, n_FG = TrainsampleDCT_FG.shape
m_BG, n_BG = TrainsampleDCT_BG.shape

# Using the results of problem 2 compute the maximum likelihood estimate for the prior probabilities.
P_FG = m_FG / (m_FG + m_BG)
P_BG = m_BG / (m_FG + m_BG)
assert P_FG + P_BG == 1

print(f"\nThe prior P_Y_cheetah: {P_FG}")
print(f"The prior P_Y_grass: {P_BG}")
```

```
The prior P_Y_cheetah: 0.1918649270913277
The prior P_Y_grass: 0.8081350729086723
```

Problem b)

Using the training data in TrainingSamplesDCT*8.mat, compute the maximum likelihood estimates for the parameters of the class conditional densities $P*{X|Y} (x|cheetah)andP_{X|Y}(x|grass)$ under the Gaussian assumption.*

Denoting by $X = \{X_1, \ldots, X_64\}$ the vector of DCT coefficients, create 64 plots with the marginal densities for the two classes $P_{X_k|Y}(x_k|cheetah)$ and $P_{X_k|Y}(x_k|grass)$, $k = 1, \ldots, 64$ on each.

Select, by visual inspection, what you think are the best 8 features for classification purposes and what you think are the worst 8 features. Hand in the plots of the marginal densities for the best-8 and worst-8 features In each subplot indicate the feature that it refers to

Answers for problem b)

The best 8 features are those that have the largest divergence from the distributions. For instance, if mean of Cheetah is visually saparated from mean of Grass or the variance of Cheetah is visually saparated.

On the other hand, the worst 8 features are those that have the smallest divergence from the distributions.

By visual inspection,

The index of best 8 features are $\{1, 18, 25, 27, 32, 33, 40, 41\}$.

The index of worst 8 features are $\{3, 4, 5, 59, 60, 62, 63, 64\}$.

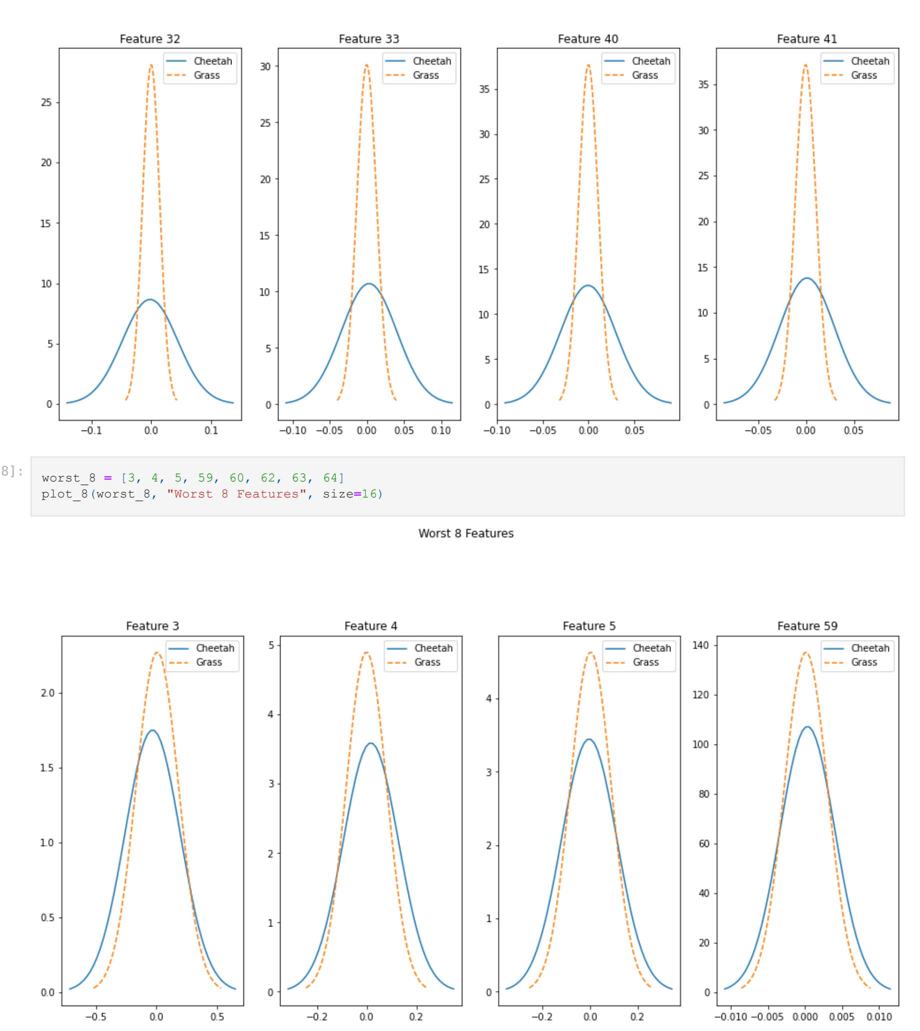
Note: The full 64 features plots are included in the appendix of the report.

Code Answers from HW2 b)

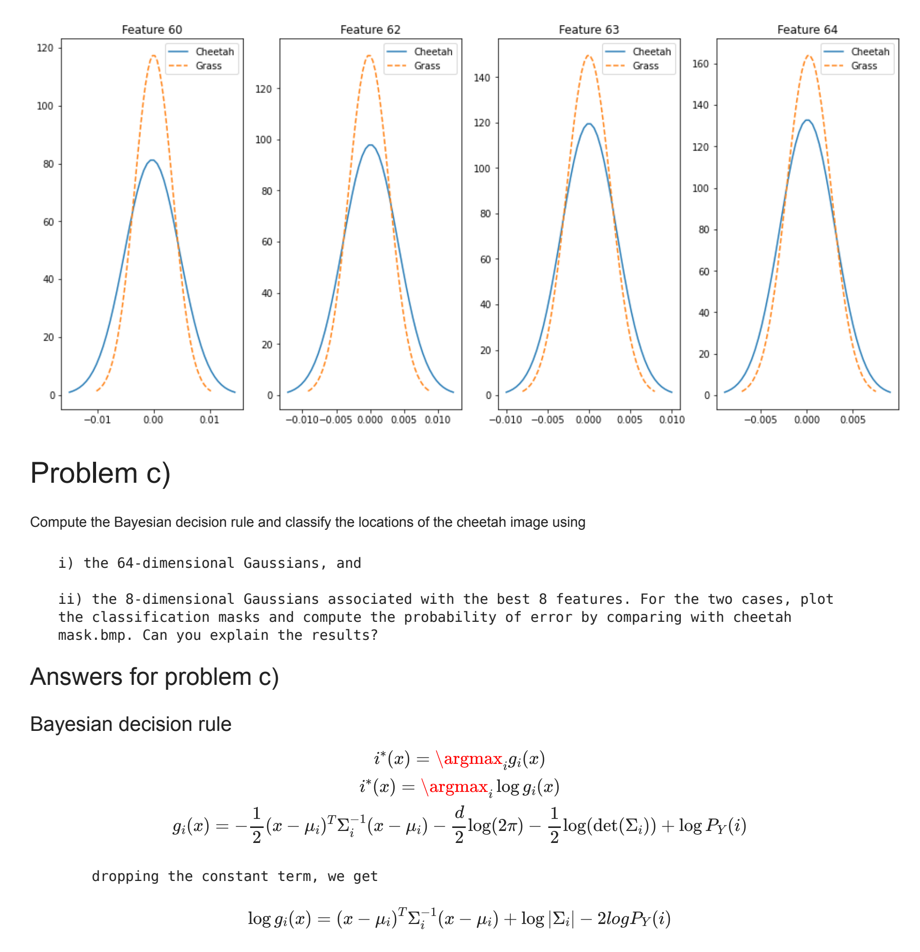
In [5]:
```python
mu_FG = np.mean(TrainsampleDCT_FG, axis=0).reshape(-1, 1)
mu_BG = np.mean(TrainsampleDCT_BG, axis=0).reshape(-1, 1)

# std sigma
std_FG = np.std(TrainsampleDCT_FG, axis=0)
std_BG = np.std(TrainsampleDCT_BG, axis=0)

# covariance Sigma
cov_FG, cov_BG = np.cov(TrainsampleDCT_FG.T), np.cov(TrainsampleDCT_BG.T)
```

In [6]:
```python
def plot_8(data, title: str, size) -> None:
    """
    Plot best8 or worst8 figures.
    """
    fig = plt.figure(title, figsize=(size, size))
    for plt_idx, j in enumerate(data):
        # since j start from 1, we need to subtract 1
        i = j - 1
        x_FG = np.linspace(-std_FG[i] * 3 + mu_FG[i], std_FG[i] * 3 + mu_FG[i])
        y_FG = univariate_gaussian_normpdf(x_FG, mu_FG[i], std_FG[i])

        x_BG = np.linspace(-std_BG[i] * 3 + mu_BG[i], std_BG[i] * 3 + mu_BG[i])
        y_BG = univariate_gaussian_normpdf(x_BG, mu_BG[i], std_BG[i])
        plt.subplot(2, 4, plt_idx + 1).set_title(f"Feature {j}")
        plt.plot(x_FG, y_FG, "-", label="Cheetah")
        plt.plot(x_BG, y_BG, "--", label="Grass")
        plt.legend(loc="best")
    fig.suptitle(title)
    plt.show()
```

In [7]:
```python
best_8 = [1, 18, 25, 27, 32, 33, 40, 41]
plot_8(best_8, "Best 8 Features", size=16)
```



In [8]:
```python
worst_8 = [3, 4, 5, 59, 60, 62, 63, 64]
plot_8(worst_8, "Worst 8 Features", size=16)
```



Problem c)

Compute the Bayesian decision rule and classify the locations of the cheetah image using

```
i) the 64-dimensional Gaussians, and

ii) the 8-dimensional Gaussians associated with the best 8 features. For the two cases, plot
the classification masks and compute the probability of error by comparing with cheetah
mask.bmp. Can you explain the results?
```

Answers for problem c)

Bayesian decision rule

$$i^*(x) = \backslash\mathrm{argmax}_i g_i(x)$$
$$i^*(x) = \backslash\mathrm{argmax}_i \log g_i(x)$$
$$g_i(x) = -\frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i) - \frac{d}{2}\log(2\pi) - \frac{1}{2}\log(\det(\Sigma_i)) + \log P_Y(i)$$

```
dropping the constant term, we get
```

$$\log g_i(x) = (x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i) + \log|\Sigma_i| - 2logP_Y(i)$$

Decision boundary interpretation

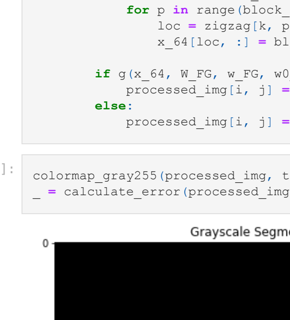
$$g_i(x) = x^T W_i x + w_i^T x + w_{i0}$$

Where

$$W_i = \Sigma_i^{-1}$$
$$w_i = -2\Sigma_i^{-1}\mu_i$$

$$w_{i0} = \mu_i^T\Sigma_i^{-1}\mu_i + \log det(\Sigma_i) - 2\log P_Y(i)$$

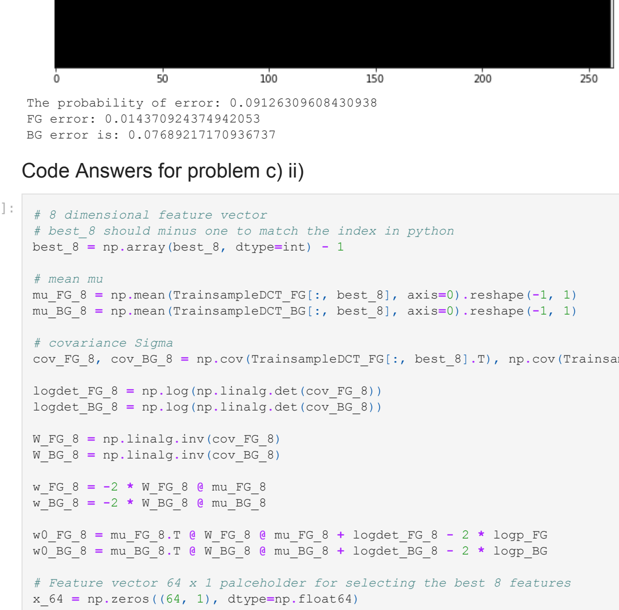
In [9]:
```python
#) 64-dimensional feature vector
img = np.asarray(Image.open(str(data_dir / "cheetah.bmp"), "r"))
img = im2double(img)

# cheetah_mask
ground_truth = np.asarray(Image.open(str(data_dir / "cheetah_mask.bmp"), "r"))
plt.imshow(ground_truth)
plt.title("Ground Truth")
plt.show()

# placeholder
processed_img = np.zeros([img.shape[0] - 8, img.shape[1] - 8], dtype=bool)

# zig-zag pattern
zigzag = np.loadtxt(data_dir / "Zig-Zag Pattern.txt", dtype=np.int64)

# log prior
logp_FG = np.log(P_FG)
logp_BG = np.log(P_BG)

# log determinant of covariance matrix
logdet_FG = np.log(np.linalg.det(cov_FG))
logdet_BG = np.log(np.linalg.det(cov_BG))

W_FG = np.linalg.inv(cov_FG)
W_BG = np.linalg.inv(cov_BG)

w_FG = -2 * W_FG @ mu_FG
w_BG = -2 * W_BG @ mu_BG

w0_FG = mu_FG.T @ W_FG @ mu_FG + logdet_FG - 2 * logp_FG
w0_BG = mu_BG.T @ W_BG @ mu_BG + logdet_BG - 2 * logp_BG
```



Code Answers for problem c) i)

In [10]:
```python
# Feature vector 64 x 1
x_64 = np.zeros((64, 1), dtype=np.float64)
for i in (range(processed_img.shape[0])):
    for j in range(processed_img.shape[1]):
        # 8 x 8 block
        block = img[i : i + 8, j : j + 8]
        # DCT transform on the block
        block_DCT = dct2(block)
        # zigzag pattern mapping
        for k in range(block_DCT.shape[0]):
            for p in range(block_DCT.shape[1]):
                loc = zigzag[k, p]
                x_64[loc, :] = block_DCT[k, p]

        if g(x_64, W_FG, w_FG, w0_FG) >= g(x_64, W_BG, w_BG, w0_BG):
            processed_img[i, j] = 0
        else:
            processed_img[i, j] = 1
```

In [11]:
```python
colormap_gray255(processed_img, title="Grayscale Segmented Image with 64D features")
_ = calculate_error(processed_img, ground_truth)
```

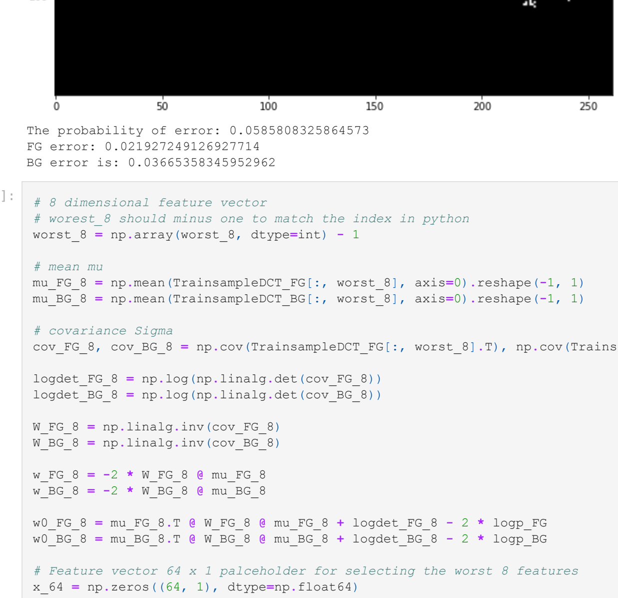



```
The probability of error: 0.09126309608430938
FG error: 0.014370924374942053
BG error is: 0.07689217170936737
```

Code Answers for problem c) ii)

In [12]:
```python
# 8 dimensional feature vector
# best_8 should minus one to match the index in python
best_8 = np.array(best_8, dtype=int) - 1

# mean mu
mu_FG_8 = np.mean(TrainsampleDCT_FG[:, best_8], axis=0).reshape(-1, 1)
mu_BG_8 = np.mean(TrainsampleDCT_BG[:, best_8], axis=0).reshape(-1, 1)

# covariance Sigma
cov_FG_8, cov_BG_8 = np.cov(TrainsampleDCT_FG[:, best_8].T), np.cov(TrainsampleDCT_BG[:, best_8].T)

logdet_FG_8 = np.log(np.linalg.det(cov_FG_8))
logdet_BG_8 = np.log(np.linalg.det(cov_BG_8))

W_FG_8 = np.linalg.inv(cov_FG_8)
W_BG_8 = np.linalg.inv(cov_BG_8)

w_FG_8 = -2 * W_FG_8 @ mu_FG_8
w_BG_8 = -2 * W_BG_8 @ mu_BG_8

w0_FG_8 = mu_FG_8.T @ W_FG_8 @ mu_FG_8 + logdet_FG_8 - 2 * logp_FG
w0_BG_8 = mu_BG_8.T @ W_BG_8 @ mu_BG_8 + logdet_BG_8 - 2 * logp_BG

# Feature vector 64 x 1 palceholder for selecting the best 8 features
x_64 = np.zeros((64, 1), dtype=np.float64)
for i in (range(processed_img.shape[0])):
    for j in range(processed_img.shape[1]):
        # 8 x 8 block
        block = img[i : i + 8, j : j + 8]
        # DCT transform on the block
        block_DCT = dct2(block)
        # zigzag pattern mapping
        for k in range(block_DCT.shape[0]):
            for p in range(block_DCT.shape[1]):
                loc = zigzag[k, p]
                x_64[loc, :] = block_DCT[k, p]
        x_8 = x_64[best_8, :]
        if g(x_8, W_FG_8, w_FG_8, w0_FG_8) > g(x_8, W_BG_8, w_BG_8, w0_BG_8):
            processed_img[i, j] = 0
        else:
            processed_img[i, j] = 1
```

In [13]:
```python
colormap_gray255(processed_img, title="Grayscale Segmented Image with best 8D features")
_ = calculate_error(processed_img, ground_truth)
```




```
The probability of error: 0.0585808325864573
FG error: 0.021927249126927714
BG error is: 0.03665358345952962
```

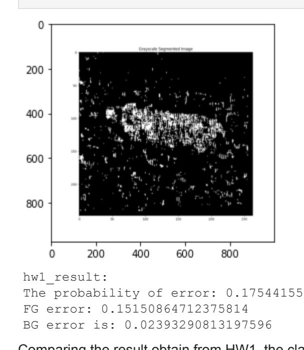
In [14]:
```python
# 8 dimensional feature vector
# worest_8 should minus one to match the index in python
worst_8 = np.array(worst_8, dtype=int) - 1

# mean mu
mu_FG_8 = np.mean(TrainsampleDCT_FG[:, worst_8], axis=0).reshape(-1, 1)
mu_BG_8 = np.mean(TrainsampleDCT_BG[:, worst_8], axis=0).reshape(-1, 1)

# covariance Sigma
cov_FG_8, cov_BG_8 = np.cov(TrainsampleDCT_FG[:, worst_8].T), np.cov(TrainsampleDCT_BG[:, worst_8].T)

logdet_FG_8 = np.log(np.linalg.det(cov_FG_8))
logdet_BG_8 = np.log(np.linalg.det(cov_BG_8))

W_FG_8 = np.linalg.inv(cov_FG_8)
W_BG_8 = np.linalg.inv(cov_BG_8)

w_FG_8 = -2 * W_FG_8 @ mu_FG_8
w_BG_8 = -2 * W_BG_8 @ mu_BG_8

w0_FG_8 = mu_FG_8.T @ W_FG_8 @ mu_FG_8 + logdet_FG_8 - 2 * logp_FG
w0_BG_8 = mu_BG_8.T @ W_BG_8 @ mu_BG_8 + logdet_BG_8 - 2 * logp_BG

# Feature vector 64 x 1 palceholder for selecting the worst 8 features
x_64 = np.zeros((64, 1), dtype=np.float64)
for i in (range(processed_img.shape[0])):
    for j in range(processed_img.shape[1]):
        # 8 x 8 block
        block = img[i : i + 8, j : j + 8]
        # DCT transform on the block
        block_DCT = dct2(block)
        # zigzag pattern mapping
        for k in range(block_DCT.shape[0]):
            for p in range(block_DCT.shape[1]):
                loc = zigzag[k, p]
                x_64[loc, :] = block_DCT[k, p]
        x_8 = x_64[worst_8, :]
        if g(x_8, W_FG_8, w_FG_8, w0_FG_8) > g(x_8, W_BG_8, w_BG_8, w0_BG_8):
            processed_img[i, j] = 0
        else:
            processed_img[i, j] = 1
```

In [15]:
```python
colormap_gray255(processed_img, title="Grayscale Segmented Image with worst 8D features")
_ = calculate_error(processed_img, ground_truth)
```


```
The probability of error: 0.18371604289643662
FG error: 0.14200945699539513
BG error is: 0.0417065859010415
```

Discussion for problem c)

In [16]:
```python
hw1_result = Image.open(str(os.path.join(plot_dir, "grayscale.png")), "r")
plt.imshow(hw1_result)
plt.show()
print("hw1_result:")
print("The probability of error: 0.17544155525573413\nFG error: 0.15150864712375814\nBG error is: 0.0239329081
```



```
hw1_result:
The probability of error: 0.17544155525573413
FG error: 0.15150864712375814
BG error is: 0.02393290813197596
```

Comparing the result obtain from HW1, the classification results is improved by adding more features. We reduced error from 0.1754 to 0.09126. To further improve the classification results, we need to do feature selection. Some feature does not help to better classify the image. Although using 8 worst features we are still able to have 0.182 error rate, this is even larger than result from HW1.

As shown in the results of 8-dimensional Gaussian feature, we select the best 8 features which have the largest divergence between the distribution of Cheetah and grass. We are able to achive probability of error of 0.05858.

Since we visually inspect the best 8 features, the selected features might not be optimal. It is possible to measure the KL-divergence between two distributions in each feature. Therefore, the result selection should be better.

```python
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 matplotlib.image import imread
from PIL import Image


try:
    from icecream import ic
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


sqrt_2_PI = np.sqrt(2 * np.pi)


def dct2(block: np.ndarray) -> np.ndarray:
    """
    Compute the DCT2 of the data.
    """
    return dct(dct(block.T, norm="ortho").T, norm="ortho")

def im2double(img: np.ndarray) -> np.ndarray:
    """
    Converts the image to double.
    """
    return img.astype(np.float64) / 255

def padding(img: np.ndarray, pad_size: int) -> np.ndarray:
    """
    Pads the image with zeros.
    """
    return np.pad(img, ((pad_size, pad_size), (pad_size, pad_size)),
"constant")


def imagesc(img: np.ndarray, title: str = "imagesc Segmented Image") ->
None:
    # equavalent to imagesc
    plt.figure(figsize=(10, 10))
    plt.imshow(img, extent=[-1, 1, -1, 1])
    plt.title(title)
    plt.show()

def colormap_gray255(img: np.ndarray, title: str = "Grayscale Segmented
Image") -> None:
```

```
    """equivalent to colormap(gray(255))"""
    plt.figure(figsize=(10, 10))
    plt.imshow(img, cmap="gray")
    plt.title(title)
    plt.show()

def load_and_compute_prior(mat_file: str) -> None:
    """
    Loads the data from the given mat file and computes the prior.

    :param mat_file: The mat file containing the data.
    """
    # Load the data from the mat file.
    # From HW1
    old_mat_contents = sio.loadmat(mat_file)
    TrainsampleDCT_BG_old = old_mat_contents["TrainsampleDCT_BG"]
    TrainsampleDCT_FG_old = old_mat_contents["TrainsampleDCT_FG"]

    # Mehode in HW1:
    m_cheetah_old = TrainsampleDCT_FG_old.shape[0]
    m_grass_old = TrainsampleDCT_BG_old.shape[0]
    P_cheetah_old = m_cheetah_old / (m_cheetah_old + m_grass_old)
    P_grass_old = m_grass_old / (m_cheetah_old + m_grass_old)
    print(f"\nThe prior P_Y_cheetah from HW1: {P_cheetah_old}")
    print(f"The prior P_Y_grass from HW1: {P_grass_old}")

def univariate_gaussian_normpdf(x, mu, sigma):
    """
    G(x, mu, sigma) = 1 / sqrt(2*pi*sigma^2) * exp(-(x-mu)^2 / (2*sigma^2))
    """
    return 1 / (sigma * sqrt_2_PI) * np.exp(-((x - mu) ** 2) / (2 * sigma
** 2))

def plot_8(data, title: str, size:int = 16) -> None:
    """
    Plot best8 or worst8 figures.
    """
    fig = plt.figure(title, figsize=(size, size))
    for plt_idx, j in enumerate(data):
        # since j start from 1, we need to subtract 1
        i = j - 1
        x_FG = np.linspace(-std_FG[i] * 3 + mu_FG[i], std_FG[i] * 3 +
mu_FG[i])
        y_FG = univariate_gaussian_normpdf(x_FG, mu_FG[i], std_FG[i])

        x_BG = np.linspace(-std_BG[i] * 3 + mu_BG[i], std_BG[i] * 3 +
mu_BG[i])
        y_BG = univariate_gaussian_normpdf(x_BG, mu_BG[i], std_BG[i])
        plt.subplot(2, 4, plt_idx + 1).set_title(f"Feature {j}")
        plt.plot(x_FG, y_FG, "-", label="Cheetah")
        plt.plot(x_BG, y_BG, "--", label="Grass")
        plt.legend(loc="best")
    fig.suptitle(title)
    plt.show()
```

```
def g(x, W, w, w0):
    """
    Decision boundary function g_i(x).
    """
    return x.T @ W @ x + w.T @ x + w0

def calculate_error(A: np.ndarray, ground_truth: np.ndarray) ->
Tuple[float, float, float]:
    """
    compute the probability of error by comparing with cheetah mask.bmp.
    """
    # 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_idex = np.where((ground_truth - A) == 1)[0]
    FG_error = len(error_idex) / A.size
    print(f"FG error: {FG_error}")

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

    return error, FG_error, BG_error


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="HW2")
    parser.add_argument("--plot", "-p", action="store_true", help="Plot the
data")
    parser.add_argument(
        "--all", "-a", action="store_true", help="combine with --plot to
plot all data"
    )
    parser.add_argument(
        "--num", "-n", type = int, help="number of features", choices=[64,
8]
    )
    args = parser.parse_args()

    #
==========================================================================
============
    # Current directory
    current_dir = pathlib.Path(__file__).parent.resolve()
    data_dir = current_dir / "data"
    old_mat_fname = data_dir / "TrainingSamplesDCT_8.mat"
    mat_fname = data_dir / "TrainingSamplesDCT_8_new.mat"
```

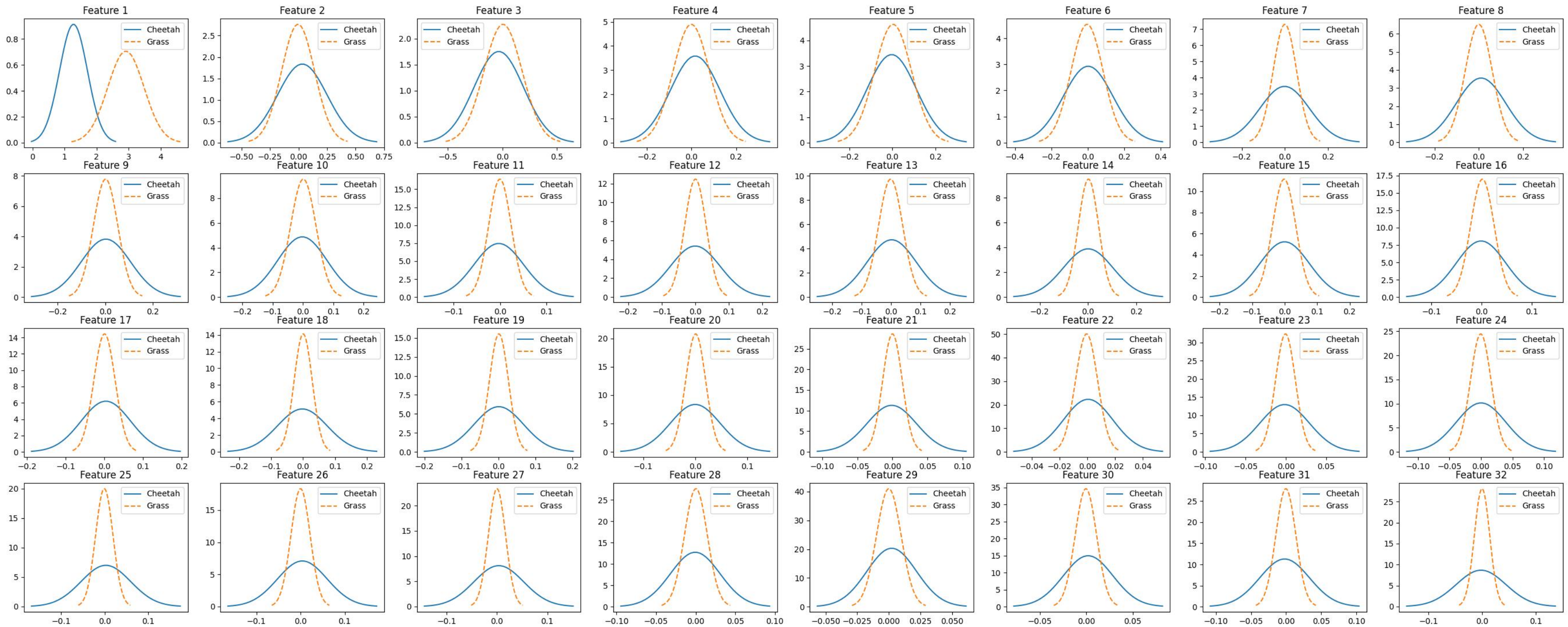
```
    zig_fname = data_dir / "Zig-Zag Pattern.txt"
    plot_dir = current_dir / "plots"

    # Create the directory if it does not exist
    for d in [data_dir, plot_dir]:
        if not os.path.exists(d):
            os.mkdir(d)

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

    # zig-zag pattern
    zigzag = np.loadtxt(zig_fname, dtype=np.int64)

    #
============================================================================
============
    # a)
    load_and_compute_prior(old_mat_fname)

    m_FG, n_FG = TrainsampleDCT_FG.shape
    m_BG, n_BG = TrainsampleDCT_BG.shape

    # Using the results of problem 2 compute the maximum likelihood
estimate for the prior probabilities.
    # $$\pi_{j} = \frac{c_i}{n}$$
    P_FG = m_FG / (m_FG + m_BG)
    P_BG = m_BG / (m_FG + m_BG)
    print(f"\nThe prior P_Y_cheetah: {P_FG}")
    print(f"The prior P_Y_grass: {P_BG}")
    assert P_FG + P_BG == 1

    #
============================================================================
============
    # b)
    # mean mu
    mu_FG = np.mean(TrainsampleDCT_FG, axis=0).reshape(-1, 1)
    mu_BG = np.mean(TrainsampleDCT_BG, axis=0).reshape(-1, 1)
    ic(mu_FG.shape, mu_BG.shape)

    # std sigma
    std_FG = np.std(TrainsampleDCT_FG, axis=0)
    std_BG = np.std(TrainsampleDCT_BG, axis=0)

    # covariance Sigma
    cov_FG, cov_BG = np.cov(TrainsampleDCT_FG.T),
```

```
np.cov(TrainsampleDCT_BG.T)

    ic(cov_FG.shape, cov_BG.shape)

    if args.plot and args.all:
        fig1 = plt.figure(figsize=(32, 32))
        for i in range(64):
            # 99.7% of data following a normal dist lies within 3 std.
Should be enough to get a good estimate.
            g_x_FG = np.linspace(-std_FG[i] * 3 + mu_FG[i], std_FG[i] * 3 +
mu_FG[i])
            y_FG = univariate_gaussian_normpdf(g_x_FG, mu_FG[i], std_FG[i])

            g_x_BG = np.linspace(-std_BG[i] * 3 + mu_BG[i], std_BG[i] * 3 +
mu_BG[i])
            y_BG = univariate_gaussian_normpdf(g_x_BG, mu_BG[i], std_BG[i])

            # Split into subplots for clarity
            if i < 32:
                plt.subplot(4, 8, i + 1).set_title(f"Feature {i+1}")
            else:
                if i == 32:
                    fig2 = plt.figure(figsize=(32, 32))
                plt.subplot(4, 8, i + 1 - 32).set_title(f"Feature {i+1}")

            plt.plot(g_x_FG, y_FG, "-", label="Cheetah")
            plt.plot(g_x_BG, y_BG, "--", label="Grass")
            plt.legend(loc="best")
        plt.show()

    # By visual inspection,
    best_8 = [1, 18, 25, 27, 32, 33, 40, 41]
    worst_8 = [3, 4, 5, 59, 60, 62, 63, 64]

    if args.plot:
        plot_8(best_8, "Best 8 Features")
        plot_8(worst_8, "Worst 8 Features")

    #
============================================================================
============
    # c)
    # load Image (original_img has dtype=uint8)
    # img = imread(data_dir/'cheetah.bmp')[:,:,0]
    img = np.asarray(Image.open(str(data_dir / "cheetah.bmp"), "r"))

    # Convert to double and / 255
    img = im2double(img)
    # ic(img.shape)  # (255, 270)
    # plt.imshow(img)
    # plt.show()
    assert img.min() == 0 and img.max() <= 1

    ground_truth = np.asarray(Image.open(str(data_dir /
```

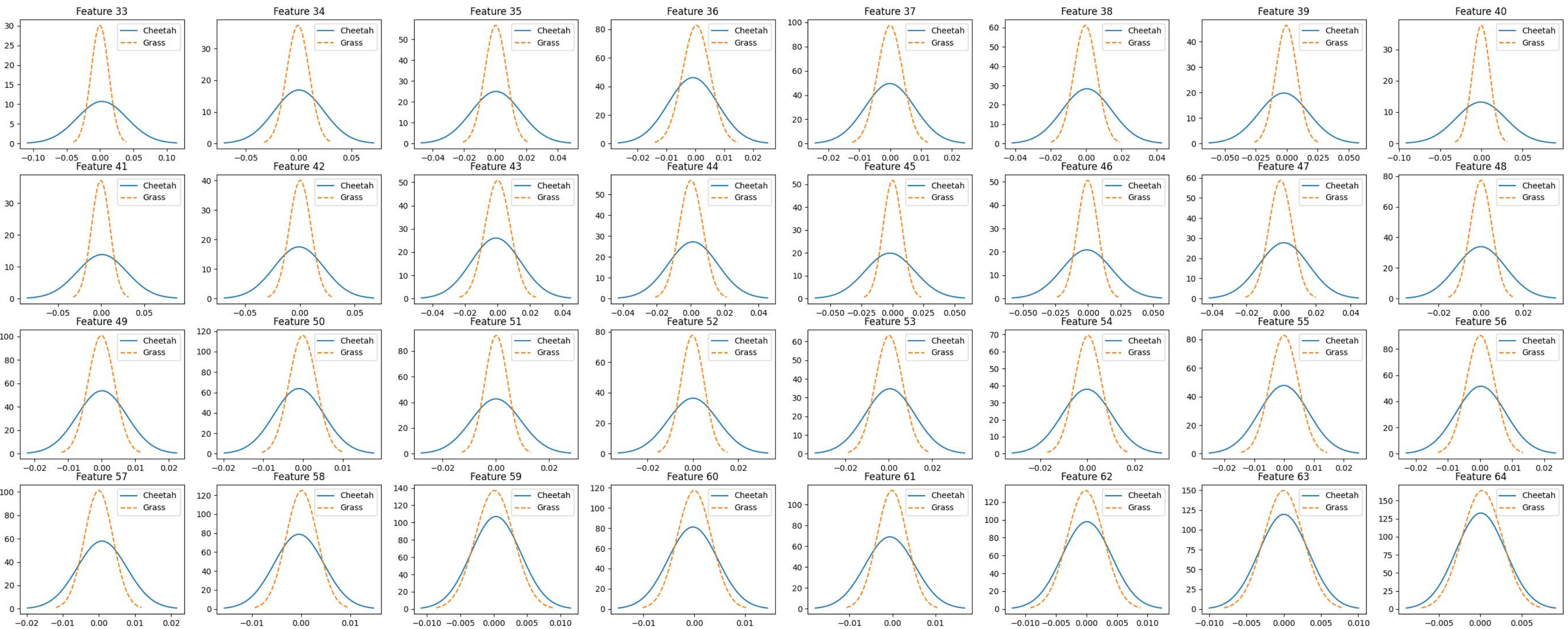
```
"cheetah_mask.bmp"), "r"))
    # ground_truth = imread(data_dir/"cheetah_mask.bmp")
    # plt.imshow(ground_truth)
    # plt.title("Ground Truth")
    # plt.show()
    # ic(ground_truth)

    processed_img = np.zeros([img.shape[0] - 8, img.shape[1] - 8],
dtype=bool)
    # ic(processed_img.shape)  # (248, 263)

    '''
    Bayesian decision rule
        i^*(x) = \argmax_i g_i(x)
        i^*(x) = \argmax_i \log g_i(x)
        g_i(x) = - \frac{1}{2} (x-\mu_i)^T \Sigma_i^{-1} (x-\mu_i) -
\frac{d}{2} \log(2 \pi) - \frac{1}{2} \log(\det(\Sigma_i)) + \log P_Y(i)
    dropping the constant term, we get
        \log g_i(x) = (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) +
\log|\Sigma_i| - 2logP_Y(i)
    '''

    '''
    Decision boundary interpretation
        g_i(x) = x^T W_i x + w_i^T x + w_{i0}

        W_i = \Sigma_i^{-1}
        w_i = -2 \Sigma_i^{-1} \mu_i    # Remember that w_i need to be
transposed
        w_{i0} = \mu_i^T \Sigma_i^{-1} \mu_i + \log det(\sigma_i) - 2 \log
P_Y(i)
    '''
    # constants

    logp_FG = np.log(P_FG)
    logp_BG = np.log(P_BG)

    if args.num == 64:

        logdet_FG = np.log(np.linalg.det(cov_FG))
        logdet_BG = np.log(np.linalg.det(cov_BG))

        W_FG = np.linalg.inv(cov_FG)
        W_BG = np.linalg.inv(cov_BG)

        w_FG = -2 * W_FG @ mu_FG
        w_BG = -2 * W_BG @ mu_BG

        w0_FG = mu_FG.T @ W_FG @ mu_FG + logdet_FG - 2 * logp_FG
        w0_BG = mu_BG.T @ W_BG @ mu_BG + logdet_BG - 2 * logp_BG


        # Feature vector 64 x 1
        x_64 = np.zeros((64, 1), dtype=np.float64)
```

```
        for i in (range(processed_img.shape[0])):
            for j in range(processed_img.shape[1]):
                # 8 x 8 block
                block = img[i : i + 8, j : j + 8]
                # DCT transform on the block
                block_DCT = dct2(block)
                # zigzag pattern mapping
                for k in range(block_DCT.shape[0]):
                    for p in range(block_DCT.shape[1]):
                        loc = zigzag[k, p]
                        x_64[loc, :] = block_DCT[k, p]

                if g(x_64, W_FG, w_FG, w0_FG) >= g(x_64, W_BG, w_BG,
w0_BG):
                    processed_img[i, j] = 0
                else:
                    processed_img[i, j] = 1

    elif args.num == 8:
        # best_8 should minus one to match the index in python
        best_8 = np.array(best_8, dtype=int) - 1

        # mean mu
        mu_FG_8 = np.mean(TrainsampleDCT_FG[:, best_8], axis=0).reshape(-1,
1)
        mu_BG_8 = np.mean(TrainsampleDCT_BG[:, best_8], axis=0).reshape(-1,
1)
        ic(mu_FG_8.shape, mu_BG_8.shape)

        # covariance Sigma
        cov_FG_8, cov_BG_8 = np.cov(TrainsampleDCT_FG[:, best_8].T),
np.cov(TrainsampleDCT_BG[:, best_8].T)
        ic(cov_FG_8.shape, cov_BG_8.shape)

        logdet_FG_8 = np.log(np.linalg.det(cov_FG_8))
        logdet_BG_8 = np.log(np.linalg.det(cov_BG_8))

        W_FG_8 = np.linalg.inv(cov_FG_8)
        W_BG_8 = np.linalg.inv(cov_BG_8)

        w_FG_8 = -2 * W_FG_8 @ mu_FG_8
        w_BG_8 = -2 * W_BG_8 @ mu_BG_8

        w0_FG_8 = mu_FG_8.T @ W_FG_8 @ mu_FG_8 + logdet_FG_8 - 2 * logp_FG
        w0_BG_8 = mu_BG_8.T @ W_BG_8 @ mu_BG_8 + logdet_BG_8 - 2 * logp_BG

        # Feature vector 64 x 1 palceholder for selecting the best 8
features
        x_64 = np.zeros((64, 1), dtype=np.float64)
        for i in (range(processed_img.shape[0])):
            for j in range(processed_img.shape[1]):
                # 8 x 8 block
                block = img[i : i + 8, j : j + 8]
                # DCT transform on the block
```

```
                block_DCT = dct2(block)
                # zigzag pattern mapping
                for k in range(block_DCT.shape[0]):
                    for p in range(block_DCT.shape[1]):
                        loc = zigzag[k, p]
                        x_64[loc, :] = block_DCT[k, p]
                x_8 = x_64[best_8, :]
                if g(x_8, W_FG_8, w_FG_8, w0_FG_8) > g(x_8, W_BG_8, w_BG_8,
w0_BG_8):
                    processed_img[i, j] = 0
                else:
                    processed_img[i, j] = 1

    else:
        raise ValueError("Invalid number of features")


    #
============================================================================
============
    colormap_gray255(processed_img)
    calculate_error(processed_img, ground_truth)
    #
============================================================================
============
```



Feature 1
Feature 2
Feature 3
Feature 4
Feature 5
Feature 6
Feature 7
Feature 8
Feature 9
Feature 10
Feature 11
Feature 12
Feature 13
Feature 14
Feature 15
Feature 16
Feature 17
Feature 18
Feature 19
Feature 20
Feature 21
Feature 22
Feature 23
Feature 24
Feature 25
Feature 26
Feature 27
Feature 28
Feature 29
Feature 30
Feature 31
Feature 32
Cheetah
Grass



Feature 33
Feature 34
Feature 35
Feature 36
Feature 37
Feature 38
Feature 39
Feature 40
Feature 41
Feature 42
Feature 43
Feature 44
Feature 45
Feature 46
Feature 47
Feature 48
Feature 49
Feature 50
Feature 51
Feature 52
Feature 53
Feature 54
Feature 55
Feature 56
Feature 57
Feature 58
Feature 59
Feature 60
Feature 61
Feature 62
Feature 63
Feature 64
Cheetah
Grass