Question 1:

```
import cv2

image = cv2.imread('img2.jpg', 1) // BGR mode

#resize image to 0.3 of its original size.
# INTER_AREA is used to shrink the image to look best
image = cv2.resize(image, None, fx=.3, fy=.3, interpolation=cv2.INTER_AREA)

# split the bgr image into its component (blue, green, red)
b,g,r = cv2.split(image)

# show the image
cv2.imshow(“Blue”, b)
cv2.imshow(“Green”, g)
cv2.imshow(“Red”, r)

# wait for a key press
k = cv2.waitKey(0)
if k==27:                # destroy if escape key is pressed
  cv2.destroyAllWindows()
elif k==ord('s')         # save otherwise if s is pressed
  cv2.imwrite('blue_img.jpg', b)
  cv2.imwrite('green_img.jpg', g)
  cv2.imwrite('red_img.jpg', r)
  cv2.destroyAllWindows()
```

Observations:
The pure red color in the red splitted image becomes white. The other far away component, blue and green, gets closer to black. The same happens to all the other two component images.

Question 2:

<u>Part a (HLS):</u>

```
import cv2

# read the image in BGR mode
image = cv2.imread('img2.jpg', 1)
```

```
# resize
image = cv2.resize(image, None, fx=.3, fy=.3, interpolation=cv2.INTER_AREA)

#  convert the image using opencv function
img_hls = cv2.cvtColor(image, cv2.COLOR_BGR2HLS)
# split the image into its channel
h,l,s = cv2.split(img)

# show the image
cv2.imshow(“Hue”, h)
cv2.imshow(“Lightness”, l)
cv2.imshow(“Saturation”, s)

# wait for a key press
k = cv2.waitKey(0)
if k==27:                # destroy if escape key is pressed
  cv2.destroyAllWindows()
elif k==ord('s')         # save otherwise if s is pressed
  cv2.imwrite('hue_img.jpg', h)
  cv2.imwrite('lightness_img.jpg', l)
  cv2.imwrite('saturation_img.jpg', s)
  cv2.destroyAllWindows()
```

Part b (HSV):

```
import cv2

# read the image in BGR mode
image = cv2.imread('img2.jpg', 1)

# resize
image = cv2.resize(image, None, fx=.3, fy=.3, interpolation=cv2.INTER_AREA)

#  convert the image using opencv function
img_hls = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
# split the image into its channel
h,s,v = cv2.split(img)

# show the image
cv2.imshow(“Hue”, h)
cv2.imshow(“Value”, v)
cv2.imshow(“Saturation”, s)
```

```
# wait for a key press
k = cv2.waitKey(0)
if k==27:                # destroy if escape key is pressed
  cv2.destroyAllWindows()
elif k==ord('s')         # save otherwise if s is pressed
  cv2.imwrite('hue_img.jpg', h)
  cv2.imwrite('value_img.jpg', v)
  cv2.imwrite('saturation_img.jpg', s)
  cv2.destroyAllWindows()
```

Expression for computing:

HLS:

$$V_{max} \leftarrow max(R, G, B)$$

$$V_{min} \leftarrow min(R, G, B)$$

$$L \leftarrow \frac{V_{max} + V_{min}}{2}$$

$$S \leftarrow \begin{cases} \frac{V_{max} - V_{min}}{V_{max} + V_{min}} & \text{if } L < 0.5 \\ \frac{V_{max} - V_{min}}{2 - (V_{max} + V_{min})} & \text{if } L \geq 0.5 \end{cases}$$

$$H \leftarrow \begin{cases} 60(G - B)/(V_{max} - V_{min}) & \text{if } V_{max} = R \\ 120 + 60(B - R)/(V_{max} - V_{min}) & \text{if } V_{max} = G \\ 240 + 60(R - G)/(V_{max} - V_{min}) & \text{if } V_{max} = B \end{cases}$$

If H<0 then H = H=360.

On output,

0<= L <= 1

0<= S <= 1

0<= H <= 360

HSV:

$$V \leftarrow max(R, G, B)$$

$$S \leftarrow \begin{cases} \frac{V - min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G - B)/(V - min(R, G, B)) & \text{if } V = R \\ 120 + 60(B - R)/(V - min(R, G, B)) & \text{if } V = G \\ 240 + 60(R - G)/(V - min(R, G, B)) & \text{if } V = B \end{cases}$$

If H<0 then H = H+360

On output,

0<= V <= 1

0<= S <= 1

0<= H <= 360

Question 3:

```
import cv2

# read the image in BGR mode
image = cv2.imread('img2.jpg', 1)

# resize
image = cv2.resize(image, None, fx=.3, fy=.3, interpolation=cv2.INTER_AREA)

#  convert the image using opencv function
img_lab = cv2.cvtColor(image, cv2.COLOR_BGR2Lab)
# split the image into its channel
L,a,b = cv2.split(img)

# show the image
cv2.imshow(“L*”, L)
cv2.imshow(“a*”, a)
cv2.imshow(“b*”, b)

# wait for a key press
k = cv2.waitKey(0)
if k==27:                # destroy if escape key is pressed
```

```
  cv2.destroyAllWindows()
elif k==ord('s')        # save otherwise if s is pressed
  cv2.imwrite('L*_img.jpg', L)
  cv2.imwrite('a*_img.jpg', a)
  cv2.imwrite('b*_img.jpg', b)
  cv2.destroyAllWindows()
```

Question 4:

```
import cv2

# read the image in BGR mode
image = cv2.imread('img2.jpg', 1)

# resize
image = cv2.resize(image, None, fx=.3, fy=.3, interpolation=cv2.INTER_AREA)

#  convert the image using opencv function
img_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# show the image
cv2.imshow(“Gray image”, img_gray)

# wait for a key press
k = cv2.waitKey(0)
if k==27:               # destroy if escape key is pressed
  cv2.destroyAllWindows()
elif k==ord('s')        # save otherwise if s is pressed
  cv2.imwrite('gray_img.jpg', img_gray)
  cv2.destroyAllWindows()
```

Expression for conversion used:

$$\mathrm{RGB[A]\ to\ Gray}: \quad Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

Question 5:

Part a (Whitening):

```
import cv2
```

```
import numpy as np

# Load the image as grayscale (0)
image = cv2.imread('img2.jpg', 0)
# resize the image
image = cv2.resize(image, None, fx=.3, fy=.3, interpolation=cv2.INTER_AREA)

mu = np.mean(image)
sigma = np.std(image)

# Whiten image
img = np.divide( np.subtract(image, mu), sigma)

cv2.imshow(“original image”, image)
cv2.imshow(“whitened image”, img)

# wait for a key press
k = cv2.waitKey(0)
if k==27:                # destroy if escape key is pressed
  cv2.destroyAllWindows()
elif k==ord('s')         # save otherwise if s is pressed
  cv2.imwrite('whitened_img.jpg', img)
  cv2.destroyAllWindows()
```

Part b (Histogram Equalization):

```
import cv2
import numpy as np

# Load the image as grayscale (0)
image = cv2.imread('img2.jpg', 0)
# resize the image
image = cv2.resize(image, None, fx=.3, fy=.3, interpolation=cv2.INTER_AREA)

# histogram equalization done using the opencv function
equ = cv2.equalizeHist(image)

# result image is made up by stacking horizontally both equ and image
result = np.hstack(image, equ)

# show the image
```

```
cv2.imshow(“histogram stacked image”, res)

# wait for a key press
k = cv2.waitKey(0)
if k==27:                # destroy if escape key is pressed
  cv2.destroyAllWindows()
elif k==ord('s')         # save otherwise if s is pressed
  cv2.imwrite('hist_stacked_img.jpg', res)
  cv2.destroyAllWindows()
```

Question 6:

```
import cv2
import numpy as np

# load the image in bgr mode
image = cv2.imread('img5.jpg', 1)
# resize the image
image = cv2.resize(image, None, fx=.5, fy=.5, interpolation=cv2.INTER_AREA)

# blur/smoothen the image using the gaussian blur function of opencv
blur1 = cv2.GaussianBlur(image, (1,1), cv2.BORDER_DEFAULT)
blur2 = cv2.GaussianBlur(image, (3,3), cv2.BORDER_DEFAULT)
blur3 = cv2.GaussianBlur(image, (5,5), cv2.BORDER_DEFAULT)
blur4 = cv2.GaussianBlur(image, (7,7), cv2.BORDER_DEFAULT)

# horizontal stack the images
i = np.hstack((image, blur1))
i = np.hstack((i, blur2))
i = np.hstack((i,blur3))
i = np.hstack((i,blur4))

cv2.imshow(“gaussian blur image with different scales”, i)
# wait for a key press
k = cv2.waitKey(0)
if k==27:                # destroy if escape key is pressed
  cv2.destroyAllWindows()
elif k==ord('s')         # save otherwise if s is pressed
  cv2.imwrite('gaussian_blur_img_with_diff_scales.jpg', i)
  cv2.destroyAllWindows()
```

Observation :

The cv2.GaussianBlur function takes arguments as (

```
InputArray src,
OutputArray dest,
Size ksize,
double sigmaX,
double sigmaY,
int borderType=BORDER_DEFAULT
```

)

- The function convolves the image with the specified Gaussian kernel and then summing them all to produce the output array.
- Ksize consists of ksize.width and ksize.height. Both can differ but we have used same here, and both has to be positive and odd. If they are zero's they are computed from sigma.
- As we scale up from (1,1) to (7,7) the image gets more smooth and the noise gets removed from the original low illuminated noisy image.

Question 7:

```
import cv2
import numpy as np

# load the image in bgr mode
image = cv2.imread('img3.jpg')
# resize the image
image = cv2.resize(image, None, fx=.3, fy=.3, interpolation=cv2.INTER_AREA)

# adding salt and pepper noise
row, col, ch = image.shape
s_vs_p = 0.5
amount = 0.2
out = np.copy(image)
# Salt mode
num_salt = np.ceil(amout * image.size * s_vs_p)
coords = [np.random.randint(0, i-1, int(num_salt)) for i in image.shape]
out[coords] = 1
```

```
# Pepper mode
num_pepper = np.ceil(amount * image_size * (1. - s_vs_p)
coords = [np.random.randint(0, i-1, int(num_pepper)) for i in image.shape]
out[coords] = 0

i = np.hstack((image, out))

# Median filtering
filt = cv2.medianFilter(out, 3)

i = np.hstack((i, filt))

cv2.imshow(“median filtered image for salt and pepper noise”, i)
# wait for a key press
k = cv2.waitKey(0)
if k==27:                # destroy if escape key is pressed
  cv2.destroyAllWindows()
elif k==ord('s')         # save otherwise if s is pressed
  cv2.imwrite('median_filter_img_for_salt_pepper_noise.jpg', i)
  cv2.destroyAllWindows()
```

Queestion 8:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import math

def region_of_interest(img, vertices):
    mask = np.zeros_like(img)
    match_mask_color = 255
    cv2.fillPoly(mask, vertices, match_mask_color)
    masked_image = cv2.bitwise_and(img, mask)
    return masked_image

def draw_lines(img, lines, color=[0,255,0], thickness=3):
    line_img = np.zeros(
          (img.shape[0],
           img.shape[1],
           3
```

```
            ),
            dtype = np.uint8
        )
        img = np.copy(img)
        if lines is None:
            return

        for line in lines:
            for x1,y1,x2,y2 in line:
                cv2.line(line_img, (x1,y1), (x2,y2), color, thickness)

        img = cv2.addWeighted(img, 0.8, line_img, 1.0, 0.0)
        return img

    def show_image(image):
        plt.figure()
        plt.imshow(image)
        plt.show()

    def pipeline(image):
        height = image.shape[0]
        width = image.shape[1]
        region_of_interest_vertices = [
            (0, height),
            (width/2, height/2),
            (width, height),
        ]

        gray_img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        blur_img = cv2.GaussianBlur(gray_img, (7,7), 0)
        cannyed_img = cv2.Canny(blur_img, 50, 125)
        cropped_img = region_of_interest(cannyed_img,
            np.array([region_of_interest_vertices], np.int32),
            )
        show_image(cropped_img)

        lines = cv2.HoughLinesP(
            cropped_img,
            rho = .2,
            theta = np.pi/180,
            threshold = 20,
```

```
        lines = np.array([]),
        minLineLength = 30,
        maxLineGap = 40
)

left_line_x = []
left_line_y = []
right_line_x = []
right_line_y = []

for line in lines:
    print(line)
    for x1,y1,x2,y2 in line:
        slope = (y2 - y1) / (x2 - x1)
        print(slope)
        if math.fabs(slope) < 0.5:
            continue
        if slope <= 0:
            left_line_x.extend([x1,x2])
            left_line_y.extend([y1,y2])
        else:
            right_line_x.extend([x1,x2])
            right_line_y.extend([y1,y2])

min_y = int(image.shape[0] * (3 / 5)) # just below the horizon
max_y = int(image.shape[0])          # bottom of the image

if len(left_line_y)>0:
    poly_left = np.poly1d(np.polyfit(
        left_line_y,
        left_line_x,
        deg = 1
    ))

    left_start_x = int(poly_left(max_y))
    left_end_x = int(poly_left(min_y))
else:
    left_start_x = 0
    left_end_x = 0

if(len(right_line_y)>0):
```

```
            poly_right = np.poly1d(np.polyfit(
                right_line_y,
                right_line_x,
                deg = 1
            ))

            right_start_x = int(poly_right(max_y))
            right_end_x = int(poly_right(min_y))
        else:
            right_start_x = 0
            right_end_x = 0

        line_image = draw_lines(image,
                [[
                    [left_start_x, max_y, left_end_x, min_y],
                    [right_start_x, max_y, right_end_x, min_y],
                ]],
                thickness=3,
            )

        cv2.imshow("road lane marked image", line_image)
        k = cv2.waitKey(0)
        if k==27:
            cv2.destroyAllWindows()
        elif k==ord('s'):
            cv2.imwrite("road_lane_marked_image.jpg", line_image)
            cv2.destroyAllWindows()


    # read the image
    image = cv2.imread('road_lane2.jpg')
    #image = cv2.resize(image, None, fx=.2, fy=.2, interpolation = cv2.INTER_AREA)
    print(image.shape)

    pipeline(image)
```

Observations:

- The correct or estimated output depends on a lot of factors such as the region of interest used which in this case is a triangle.

- The edge detection and plotting of the edges to line depends on the various parameters in Hough Line function also.
- So this function may work perfectly for a given image where the lane lies perfectly in the region of interest defined (triangle in this case) but may not give desired results for other images unless the region of interest and Hough Line parameters are changed to suit the image in consideration.

Question 9:

```
# import the necessary packages
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import train_test_split
from imutils import paths
import numpy as np
import argparse
import imutils
import cv2
import os

def image_to_feature_vector(image, size=(32, 32)):
	# resize the image to a fixed size, then flatten the image into
	# a list of raw pixel intensities
	return cv2.resize(image, size).flatten()

def extract_color_histogram(image, bins=(8, 8, 8)):
	# extract a 3D color histogram from the HSV color space using
	# the supplied number of `bins` per channel
	hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
	hist = cv2.calcHist([hsv], [0, 1, 2], None, bins,
		[0, 180, 0, 256, 0, 256])

	# handle normalizing the histogram
	cv2.normalize(hist, hist)

	# return the flattened histogram as the feature vector
	return hist.flatten()

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
```

```
        help="path to input dataset")
ap.add_argument("-k", "--neighbors", type=int, default=1,
        help="# of nearest neighbors for classification")
args = vars(ap.parse_args())

# grab the list of images that we'll be describing
print("[INFO] describing images...")
imagePaths = list(paths.list_images(args["dataset"]))

# initialize the raw pixel intensities matrix, the features matrix,
# and labels list
rawImages = []
features = []
labels = []

# loop over the input images
for (i, imagePath) in enumerate(imagePaths):
    # load the image and extract the class label (assuming that our
    # path as the format: /path/to/dataset/{class}.{image_num}.jpg
    image = cv2.imread(imagePath)
    label_tmp = imagePath.split(os.path.sep)[-1].split(".")[0]
    label = label_tmp.split("_")[0]
    print("label"+label)

    # extract raw pixel intensity "features", followed by a color
    # histogram to characterize the color distribution of the pixels
    # in the image
    pixels = image_to_feature_vector(image)
    hist = extract_color_histogram(image)

    # update the raw images, features, and labels matricies,
    # respectively
    rawImages.append(pixels)
    features.append(hist)
    labels.append(label)

    # show an update every 10 images
    if i > 0 and i % 10 == 0:
        print("[INFO] processed {}/{}".format(i, len(imagePaths)))

# show some information on the memory consumed by the raw images
```

```
    # matrix and features matrix
    rawImages = np.array(rawImages)
    features = np.array(features)
    labels = np.array(labels)
    print("[INFO] pixels matrix: {:.2f}MB".format(
        rawImages.nbytes / (1024 * 1000.0)))
    print("[INFO] features matrix: {:.2f}MB".format(
        features.nbytes / (1024 * 1000.0)))

    # partition the data into training and testing splits
    (trainRI, testRI, trainRL, testRL) = train_test_split(
        rawImages, labels, test_size=0.20, random_state=42)
    (trainFeat, testFeat, trainLabels, testLabels) = train_test_split(
        features, labels, test_size=0.20, random_state=42)

    # train and evaluate a k-NN classifer on the raw pixel intensities
    print("[INFO] evaluating raw pixel accuracy...")
    model = KNeighborsClassifier(n_neighbors=args["neighbors"],
        n_jobs=args["jobs"])
    model.fit(trainRI, trainRL)
    acc = model.score(testRI, testRL)
    print("[INFO] raw pixel accuracy: {:.2f}%".format(acc * 100))

    # train and evaluate a k-NN classifer on the histogram
    # representations
    print("[INFO] evaluating histogram accuracy...")
    model = KNeighborsClassifier(n_neighbors=args["neighbors"],
        n_jobs=args["jobs"])
    model.fit(trainFeat, trainLabels)
    acc = model.score(testFeat, testLabels)
    print("[INFO] histogram accuracy: {:.2f}%".format(acc * 100))
```

Observations:

- The above method uses two type of process to classify images between landscape, portrait and night images. First is using raw intensities of the pixels of the images as feature vectors and the other as using color histogram of the image as feature vector which, then, in both cases is finally fed into a K-NN classifier for classification into 3 categories.
- The first method just resizes the image and then flattens it before returning back as the feature vector.

- The second method converts the image into HSV and then calculates its histogram, normalizes it and then flattens it before returning it back as the feature vector.
- The accuracy of the process is very low, sometimes as low as random guessing, which results in 33% accuracy. The accuracy depends on the kind of images being used.
- If the landscape labelled image is dark it has difficulty in calculating its neighbors as the other landscape images because of the intensities of the raw pixels and/or histogram will match more with that of the night images which may or may not be landscape.
- Also, apart from lightning, occulusion and some other factors play a role in the accuracy as can be seen by using different type of portrait images and then re-calculating accuracy.