ECMM426 Computer Vision

Course Assessment

This is an autogradable course assessment (CA) for the ECMM426 Computer Vision module, which represents 60% of the overall module assessment.

This is an individual exercise and your attention is drawn to the College and University guidelines on collaboration and plagiarism, which are available from the University of Exeter website.

Important:

1. Do not change the name of this notebook and the containing folder. The notebook and the folder should respectively be named as **CA.ipynb** and **CA**.
2. Do not add and remove/delete any cell. You can work on a draft notebook and only copy the functions/implementations here.
3. Do not add your name or student code in the notebook or in the file name.
4. Each question asks for one or more functions to be implemented.
5. Each question is associated with appropriate marks and clearly specifies the marking criteria. Most of the questions have partial grading.
6. Each question specifies a particular type of inputs and outputs which you should regard.
7. Each question specifies data for your experimentation and test which you can consider.
8. A hidden unit test is going to evaluate if all the desired properties of the required function(s) are met or not.
9. If the test passes all the associated marks will be rewarded, if it fails 0 marks will be awarded.
10. There is no restriction on the usage of any function from the packages from pip3 distribution.
11. While uploading your work on e-Bart, please do not upload the EXCV10 and MaskedFace datasets you use for training your model.

Question 1 (3 marks)

Write a function `add_gaussian_noise(im, m, std)` which will add Gaussian noise with mean `m` and standard deviation `std` to the input image `im` and will return the noisy image. Note that the output image must be of `uint8` type and the pixel values should be normalized in $[0, 255]$.

Inputs

- `im` is a 3 dimensional numpy array of type `uint8` with values in $[0, 255]$.
- `m` is a real number.
- `std` is a real number.

Outputs

- The expected output is a 3 dimensional numpy array of type `uint8` with values in $[0, 255]$.

Data

- You can work with the image at `data/books.jpg`.

Marking Criteria

- The output with a particular `m` and `std` should exactly match with the correct noisy image with that `m` and `std` to obtain the full marks. There is no partial marking for this question.

In [70]:
```
import cv2


def add_gaussian_noise(im, m, std):
    """
    Add Gaussian noise to an input image.
    """
    # Generate noise matrix
    shape = im.shape
    noise = np.random.normal(m, std, shape)

    # Add noise to image
    noisy_image = np.add(im, noise)

    # Normalise to uint8 (0-255)
    norm_noisy_image = cv2.normalize(
        noisy_image, None, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV
    )

    return norm_noisy_image
```

In [2]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [3]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 2 (3 marks)

Speckle noise is defined as multiplicative noise, having a granular pattern, it is the inherent property of Synthetic Aperture Radar (SAR) imagery. More details on Speckle noise can be found here). Write a function `add_speckle_noise(im, m, std)` which will add Speckle noise with mean `m` and standard deviation `std` to the input image `im` and will return the noisy image. Note that the output image must be of `uint8` type and the pixel values should be normalized in $[0, 255]$.

Inputs

- `im` is a 3 dimensional numpy array of type `uint8` with values in $[0, 255]$.
- `m` is a real number.
- `std` is a real number.

Outputs

- The expected output is a 3 dimensional numpy array of type `uint8` with values in $[0, 255]$.

Data

- You can work with the image at `data/books.jpg` .

Marking Criteria

- The output with a particular `m` and `std` should exactly match with correct noisy image with that `m` and `std` to obtain the full marks. There is no partial marking for this question.

In [71]:
```
import cv2


def add_speckle_noise(im, m, std):
    """
    Add speckle noise to an input image.
    """
    # Generate noise matrix
    shape = im.shape
    noise = np.random.normal(m, std, shape)

    # Create speckle noise
    speckle_noise = np.multiply(im, noise)

    # Add noise to image
    noisy_image = np.add(im, speckle_noise)

    # Normalise to uint8 (0-255)
    norm_noisy_image = cv2.normalize(
        noisy_image, None, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX, dtype=cv2.C\
    )

    return norm_noisy_image
```

In [5]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 3 (2 marks)

Write a function `cal_image_hist(gr_im)` which will calculate the histogram of pixel intensities of a gray image `gr_im` . Note that the histogram will be a one dimensional array whose length must be equal to the maximum intensity value of `gr_im` .

Inputs

- `gr_im` is a 2 dimensional numpy array of type `uint8` with values in $[0, 255]$.

Outputs

- The expected output is a 1 dimensional numpy array of type `uint8` .

Data

- You can play with the image at `data/books.jpg`.

Marking Criteria

- The output should exactly match with correct histogram of a given gray image `gr_im` to obtain the full marks. There is no partial marking for this question.

In [72]:
```
import cv2


def cal_image_hist(gr_im):
    """
    Calculate the histogram of pixel intensities of a gray image.
    """
    # Generate histogram based on grey channel
    histogram = cv2.calcHist([gr_im], [0], None, [256], [0, 256])

    # Normalise to uint8 (0-255)
    histogram = cv2.normalize(
        src=histogram,
        dst=None,
        alpha=0,
        beta=255,
        norm_type=cv2.NORM_MINMAX,
        dtype=cv2.CV_8U,
    )

    return histogram.ravel().astype("uint8")
```

In [7]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 4 (3 marks)

Write a function `compute_gradient_magnitude(gr_im, kx, ky)` to compute gradient magnitude of the gray image `gr_im` with the horizontal kernel `kx` and vertical kernel `ky`.

Inputs

- `gr_im` is a 2 dimensional numpy array of data type `uint8` with values in $[0, 255]$.
- `kx` and `ky` are 2 dimensional numpy arrays of data type `uint8`.

Outputs

- The expected output is a 2 dimensional numpy array of the same shape as of `gr_im` and of data type `float64`.

Data

- You can work with the image at `data/shapes.png`.

Marking Criteria

- The output should exactly match with the correct gradient magnitude of a given gray image `gr_im` to obtain the full marks. There is no partial marking for this question.

In [73]:
```python
import cv2


def compute_gradient_magnitude(gr_im, kx, ky):
    """
    Compute direction of gradient of the gray image.
    """
    # Convolution
    I_x = cv2.filter2D(gr_im, cv2.CV_64F, kx)
    I_y = cv2.filter2D(gr_im, cv2.CV_64F, ky)

    # Get magnitude
    mag = np.sqrt(I_x ** 2 + I_y ** 2)

    return mag.astype("float64")
```

In [9]:
```python
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 5 (2 marks)

Write a function `compute_gradient_direction(gr_im, kx, ky)` to compute direction of gradient of the gray image `gr_im` with the horizontal kernel `kx` and vertical kernel `ky`.

Inputs

- `gr_im` is a 2 dimensional numpy array of data type `uint8` with values in $[0, 255]$.
- `kx` and `ky` are 2 dimensional numpy arrays of data type `uint8`.

Outputs

- The expected output is a 2 dimensional numpy array of same shape as of `gr_im` and of data type `float64`.

Data

- You can work with the image at `data/shapes.png`.

Marking Criteria

- The output should exactly match with the correct gradient direction of a given gray image `gr_im` to obtain the full marks. There is no partial marking for this question.

In [74]:
```python
def compute_gradient_direction(gr_im, kx, ky):
    """
    Compute direction of gradient of the gray image.
    """
    # Convolution
    I_x = cv2.filter2D(gr_im, cv2.CV_64F, kx)
    I_y = cv2.filter2D(gr_im, cv2.CV_64F, ky)

    # Get angles
    direction = np.arctan2(I_y, I_x)

    return direction.astype("float64")
```

In [11]:
```python
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 6 (8 marks)

Write a function `detect_harris_corner(im, ksize, sigmaX, sigmaY, k)` which will detect the corners in the image `im`. Here `ksize` is the kernel size for smoothing the image, `sigmaX` and `sigmaY` are respectively the standard deviation of the kernal along the horizontal and vertical direction, and `k` is the constant in the Harris criteria. Experiment with your corner detection function on the following image (located at `data/shapes.png`): Shapes Adjust the parameters of your function so that it can detect all the corners in that image. Please feel free to change the given default parameters and set your best parameters as default. You must not resize the above image and note that the returned output should be an $N \times 2$ array of type `int64`, where N is the total number of existing corner points in the image; each row of that $N \times 2$ array should be a Cartesian coordinate of the form (x, y). Also please make sure that your function is rotation invariant which is the fundamental property of the Harris corner detection algorithm.

Inputs

- `im` is a 3 dimensional numpy array of type `uint8` with values in $[0, 255]$.
- `ksize` is an integer number.
- `sigmaX` is an integer number.
- `sigmaY` is an integer number.
- `k` is a floating number.

Outputs

- The expected output is 2 dimension numpy array of data type `int64` of size $N \times 2$, whose each row should be a Cartesian coordinate of the form (x, y).

Data

- You can work with the image at `data/shapes.png`.

Marking Criteria

- You will obtain full marks if your function can detect all the existing corners in the image, while the image is being rotated to different angles. There is partial marking for this question, which will depend on the performance of the function on that image rotated to different angles.

In [75]:
```
from skimage.feature import corner_peaks


def detect_harris_corner(im, ksize=5, sigmaX=3, sigmaY=3, k=0.01):
    # Convert to grayscale
    gray_img = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY).astype(np.float32)

    # Light Gaussian smoothing
    gray_img = cv2.GaussianBlur(gray_img, (ksize, ksize), sigmaX=sigmaX, sigmaY=si

    # Construct Sobel kernels
    sobelX = np.array(([-1, 0, 1], [-2, 0, 2], [-1, 0, 1]), dtype=np.float64)
    sobelY = np.array(([-1, -2, -1], [0, 0, 0], [1, 2, 1]), dtype=np.float64)
```

```
    # Convolution
    I_x = cv2.filter2D(gray_img, -1, sobelX)
    I_y = cv2.filter2D(gray_img, -1, sobelY)

    # Gradient covariances and light Gaussian smoothing
    I_x_I_x = cv2.GaussianBlur(I_x * I_x, (ksize, ksize), sigmaX=sigmaX, sigmaY=si
    I_y_I_y = cv2.GaussianBlur(I_y * I_y, (ksize, ksize), sigmaX=sigmaX, sigmaY=si
    I_x_I_y = cv2.GaussianBlur(I_x * I_y, (ksize, ksize), sigmaX=sigmaX, sigmaY=si

    # Determinant
    detA = I_x_I_x * I_y_I_y - I_x_I_y**2

    # Trace
    traceA = I_x_I_x + I_y_I_y

    # Harris criteria
    R = detA - k * traceA**2
    corners = corner_peaks(R, min_distance=1, threshold_abs=1000000)
    x, y = corners[:, 1], corners[:, 0]

    return np.array((x, y)).astype("int64")
```

In [13]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [14]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [15]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [16]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [17]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [18]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [19]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [20]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 7 (6 marks)

Write a function `compute_homogeneous_rotation_matrix(points, theta)` to compute the rotation matrix in homogeneous coordinate system to rotate a shape depicted with 2 dimensional (x, y) coordinates `points` to an angle θ (`theta` in the definition) in the anticlockwise direction about the center of the shape.

Inputs

- `points` is a 2 dimensional numpy array of data type `uint8` with shape $k \times 2$. Each row of `points` is a Cartesian coordinate (x, y).
- `theta` is a floating point number denoting the angle of rotation in degree.

Outputs

- The expected output is a 2 dimensional numpy array of data type `float64` with shape 3×3.

Data

- You can work with the 2 dimentional numpy array at `data/points.npy`.

Marking Criteria

- You will obtain the full mark if your rotation matrix exactly matches with the actual rotation matrix. If your matrix does not exactly match, you will not get any mark and there is no martial mark for this question.

In [76]:
```
def compute_homogeneous_rotation_matrix(points, theta):
    """
    Compute the rotation matrix in a homogeneous coordinate system to rotate a shap
    """
    # Get centroid by taking mean of all points in each dimension
    xs = points[:, 0]
    ys = points[:, 1]
    x_bar = np.mean(xs)
    y_bar = np.mean(ys)

    # Find translation to origin
    T1 = [1, 0, -x_bar]
    T2 = [0, 1, -y_bar]
    T3 = [0, 0, 1]
    T = [T1, T2, T3]
    T = np.array(T)

    # Find translation back from origin
    T1_inv = [1, 0, x_bar]
    T2_inv = [0, 1, y_bar]
    T3_inv = [0, 0, 1]
    T_inv = [T1_inv, T2_inv, T3_inv]
    T_inv = np.array(T_inv)

    # Convert theta to radians
    theta = (np.pi / 180) * theta

    # Put together the rotation matrix
    R1 = [np.cos(theta), -np.sin(theta), 0]
    R2 = [np.sin(theta), np.cos(theta), 0]
    R3 = [0, 0, 1]
    R = [R1, R2, R3]
    R = np.array(R)

    # Compose matrix of moving to origin, rotating, then moving back
    M = T_inv @ R @ T

    return np.array(M).astype("float64")
```

In [22]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 8 (5 marks)

Write a function `compute_sift(im, x, y, feature_width)` to compute a basic version of SIFT-like local features at the locations (x, y) of the RGB image `im` as described in the lecture materials and chapter 7.1.2 of the 2nd edition of Szeliski's book. The parameter `feature_width` is an integer representing the local feature width in pixels. You can assume that `feature_width` will be a multiple of 4 (i.e. every cell of your local SIFT-like feature will have an integer width and height). This is the initial window size you examine around each keypoint. Your implemented function should return a numpy array of shape $k \times 128$, where k is the number of keypoints (x, y) input to the function.

Interest Points

Please feel free to follow all the minute details of the SIFT paper in your implementation, but please note that your implementation does not need to exactly match all the details to achieve a good performance. Instead a basic version of SIFT implementation is asked in this exercise, which should achieve a reasonable result. The following three steps could be considered as the basic steps: (1) a 4×4 grid of cells, each feature_width/4. It is simply the terminology used in the feature literature to describe the spatial bins where gradient distributions will be described. (2) each cell should have a histogram of the local distribution of gradients in 8 orientations. Appending these histograms together will give you $4 \times 4 \times 8 = 128$ dimensions. (3) Each feature should be normalized to unit length.

Inputs

- `im` is a 3 dimensional numpy array of data type `uint8` with values in $[0, 255]$.
- `x` is a 2 dimensional numpy array of data type `float64` with shape $k \times 1$.
- `y` is a 2 dimensional numpy array of data type `float64` with shape $k \times 1$.
- `feature_width` is an integer.

Outputs

- The expected output is a 2 dimensional numpy array of data type `float64` with shape $k \times d$, where $d = 128$ is the length of SIFT feature vector.

Data

- You can tune your algorithm/parameters with the image at `data/notre_dame_1.jpg` and interest points at `data/notre_dame_1_to_notre_dame_2.pkl`.

Marking Criteria

- You will get full marks if your output is shape wise consistent with the expected output. This function will further be tested together with the feature matching function to be implemented in the next question. There is no partial marking for this question.

In [77]:

```
def compute_sift(im, x, y, feature_width=16, scales=None):
    # Convert one image to gray
    img = cv2.cvtColor(im, cv2.COLOR_RGB2GRAY).astype(np.float32)

    # Gaussian smoothing
    img = cv2.GaussianBlur(img, (3, 3), 2)

    # Get Sobel filters
    I_x = cv2.Sobel(gray_img1, cv2.CV_64F, 1, 0, ksize=3)
    I_y = cv2.Sobel(gray_img1, cv2.CV_64F, 0, 1, ksize=3)
```

```
    # Compute angles and magnitudes of image
    angles = np.arctan2(I_y, I_x)
    magnitudes = np.sqrt(I_x**2 + I_y**2)

    # Initialise array to store results
    features = np.zeros((x.shape[0], 128))

    # Find a random interest point
    idx = np.random.randint(x1.shape[0])
    xi = int(x1[idx])
    yi = int(y1[idx])

    # Get patch
    half_win_rng = 8
    angles_patch = angles[
        yi - half_win_rng : yi + half_win_rng, xi - half_win_rng : xi + half_win_r
    ]
    magnitudes_patch = magnitudes[
        yi - half_win_rng : yi + half_win_rng, xi - half_win_rng : xi + half_win_r
    ]

    step = floor(feature_width / 4)
    for i in range(x.shape[0]):
        xi = int(x[i])
        yi = int(y[i])

        # Get patch
        angles_patch = angles[yi - win_rng : yi + win_rng, xi - win_rng : xi + win_
        magnitudes_patch = magnitudes[
            yi - win_rng : yi + win_rng, xi - win_rng : xi + win_rng
        ]

        # Use patch for each key point
        sift_features = []
        for j in range(0, feature_width, step):
            for k in range(0, feature_width, step):
                # Get bins for each patch
                angles_patch = angles[
                    yi - half_win_rng : yi + half_win_rng,
                    xi - half_win_rng : xi + half_win_rng,
                ]
                magnitudes_patch = magnitudes[
                    yi - half_win_rng : yi + half_win_rng,
                    xi - half_win_rng : xi + half_win_rng,
                ]
```

In [24]:
```
# This cell is reserved for the unit tests. Please Leave this cell as it is.
```

Question 9 (10 marks)

Write a function `match_features(features1, features2, x1, y1, x2, y2, threshold)` to implement the "ratio test" or "nearest neighbor distance ratio test" method of matching two sets of local features `features1` at the locations `(x1, y1)` and `features2` at the locations `(x2, y2)` as described in the lecture materials and in the chapter 7.1.3 of the 2nd edition of Szeliski's book.

Feature Matching

The parameters `features1` and `features2` are numpy arrays of shape $k \times 128$, each representing one set of features. `x1` and `x2` are two numpy arrays of shape $k \times 1$ respcectively containing the x-locations of `features1` and `features2`. `y1` and `y2` are two numpy arrays of shape $k \times 1$ respcectively containing the y-locations of `features1` and `features2`. Your function should return two outputs: `matches` and `confidences`, where `matches` is a numpy array of shape $n \times 2$, where n is the number of matches. The first column of `matches` is an index in `features1`, and the second column is an index in `features2`. `confidences` is a numpy array of shape $k \times 1$ with the real valued confidence for every match.

This function does not need to be symmetric (e.g. it can produce different numbers of matches depending on the order of the arguments). To start with, simply implement the "ratio test", equation 7.18 in section 7.1.3 of Szeliski. There are a lot of repetitive features in these images, and all of their descriptors will look similar. The ratio test helps us resolve this issue (also see Figure 11 of David Lowe's IJCV paper). Please try to tune your SIFT descriptors and matching algorithm together to obtain a better matching score. You can use the images and correspondences below to tune your algorithm.

Inputs

- `features1` is a 2 dimensional numpy array of data type `float64` with shape $m \times d$.
- `features2` is a 2 dimensional numpy array of data type `float64` with shape $n \times d$.
- `x1` is a 2 dimensional numpy array of data type `float64` with shape $m \times 1$.
- `y1` is a 2 dimensional numpy array of data type `float64` with shape $m \times 1$.
- `x2` is a 2 dimensional numpy array of data type `float64` with shape $n \times 1$.
- `y2` is a 2 dimensional numpy array of data type `float64` with shape $n \times 1$.
- `threshold` is a real number of data type `float64`.

Outputs

- `matches` is a 2 dimensional numpy array of data type `int64`.
- `confidences` is a 1 dimensional numpy array of data type `float64`.

Data

- You can tune your algorithm on the images at `data/notre_dame_1.jpg` and `data/notre_dame_2.jpg`, and interest points at `data/notre_dame_1_to_notre_dame_2.pkl` and also on the images at `data/mount_rushmore_1.jpg` and `data/mount_rushmore_2.jpg`, and interest points at `data/mount_rushmore_1_to_mount_rushmore_2.pkl`. Note that the corresponding points within the pickle files are the matching points.

Marking Criteria

- The marking will be based on matching accuracy obtained by the feature description and matching algorithm implemented by you respectively in the previous and this question. There are two test cases (5 marks each) with two different pairs of images and corresponding points, which are provided in the Data section. You will obtain 60% marks if your algorithm can obtain matching accuracy greater than or equal to 50%, 80% marks if your algorithm obtains 70% accuracy or more, and full marks if your algorithm secures 90% matching accuracy or more. You will not obtain any mark if your algorithm can not achieve 50% matching accuracy.

In [78]:
```
# Feature matching
def match_features(features1, features2, x1, y1, x2, y2, threshold=1.0):

    # Use FLANN to perform feature matching
    FLANN_INDEX_KDTREE = 0
    index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
    search_params = dict(checks=50)
    flann = cv2.FlannBasedMatcher(index_params, search_params)
    matches = flann.knnMatch(features1, features2, k=2)

    # Store good matches using ratio test.
    good = []
    for m, n in matches:
        if m.distance < 0.5 * n.distance:
            good.append(m)
    if len(good) > MIN_MATCH_COUNT:
        p1 = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1, 2)
        p2 = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1, 2)
    draw_params = dict(
        matchColor=(0, 255, 0),  # draw matches in green color
        singlePointColor=None,
        flags=2,
    )

    img_siftmatch = cv2.drawMatches(img1, kp1, img2, kp2, good, None, **draw_param

    return img_siftmatch
```

In [26]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [27]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [28]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [29]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [30]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [31]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 10 (5 marks)

Write a function `find_affine_transform(x1, y1, x2, y2)` which will return the homogeneous affine transformation matrix T from (x_1, y_1) to (x_2, y_2), where (x_1, y_1) and (x_2, y_2) are the 2 dimensional corresponding/matching points from two different images. The technique for computing transformation matrix was covered in the lectures, which is an approximation of any generic affine transformation matrix and can be done with the help of homogeneous coordinate.

Inputs

- `x1`, `y1`, `x2`, `y2` are 2 dimensional numpy arrays of shape $N \times 1$ of data type `float64`.

Outputs

- This function should return a 2 dimensional numpy array of shape 3×3 of data type `float64`.

Data

- You can consider the matching points at `data/notre_dame_1_to_notre_dame_2.pkl` for tuning your algorithm.

Marking Criteria

- You will obtain full marks if and only if the homogeneous affine transformation matrix calculated by your algorithm exactly matches with the correct one. There is no partial marking for this question.

In [79]:
```
def find_affine_transform(x1, y1, x2, y2):
    """
    Return the homogeneous affine transformation between two sets of points.
    """
    # Stack each set of co-ordinates into a single 2d array
    coordinates = [(x1, y1), (x2, y2)]
    xy_from = np.column_stack(coordinates[0])
    xy_to = np.column_stack(coordinates[1])

    # least-squares approximation (minimise sum of squares of residuals)
    # https://math.stackexchange.com/questions/725185/minimize-a-x-b
    xy_from_T = xy_from.transpose()
    T1 = xy_from_T @ xy_from
    T2 = xy_from_T @ xy_to
    T1_inv = np.linalg.inv(T1)
    T = T1_inv @ T2

    # Make homogeneous
    T = np.pad(T, ((0, 1), (0, 1)))
    T[2][2] = 1

    return T.astype("float64")
```

In [33]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 11 (10 marks)

Write a function `make_bovw_spatial_histogram(im, locations, clusters, division)` to create bag of visual words representation of an image `im` whose features are located at `locations` and the quantized labels of those features are stored in `clusters`. You have to build the histogram based on the division information provided in `division`. For example, if `division = [2, 3]`, you have to imagine dividing the image along Y-axis in 2 parts and along X-axis in 3 parts (as shown in the right most figure below), else if `division = [2, 2]`, you have to imagine dividing the image in 2 parts along both the axes, else if `division = [1, 1]`, you just compute the bag-of-visual-words histogram on the entire image without dividing into any parts.

Spatial Histogram

Inputs

- `im` is a 3 dimensional numpy array of data type `uint8`.
- `locations` is a 2 dimensional numpy array of shape $N \times 2$ of data type `int64`, whose each row is a Cartesian coordinate (x, y).
- `clusters` is a 1 dimensional numpy array of shape $(N,)$ of data type `int64`, whose each element indicates the quantized cluster id.
- `division` is a list of integer of length 2.

Outputs

- This function should return a 1 dimensional numpy array of data type `int64`.

Data

- There is no specific data for this question. However, you can create data on one of the images available inside the `data` folder.

Marking Criteria

- There are four test cases which will call the above functionn to calculate bag-of-visual-words spatial histograms on the image `im` imagining its coarse and fine divisions which will be provided while calling the function. In each test case, your spatial histogram should be exactly matched with the correct spatial histogram to obtain the full marks. Coarser test cases contain lower weightage compared to their finer counter parts.

In [80]:

```
def make_bovw_spatial_histogram(im, locations, clusters, division):
    """
    Create bag of visual words to represent an image as a set of features.
    """
    # Get useful values
    width = im.shape[1]
    height = im.shape[0]
    segments_x = division[1]
    segments_y = division[0]

    # Initialise histogram
    histogram = []

    # Iterate through each segment of the image
    for j in range(segments_y):
        for i in range(segments_x):
            # Get top-left co-ordinate for current segment
            x1 = round(width / segments_x) * i
            y1 = round(height / segments_y) * j

            # Get bottom-right co-ordinate for current segment
            x2 = round(width / segments_x * (i + 1))
            y2 = round(height / segments_y * (j + 1))
```

```
            # Check each location for if it falls into current segment
            selected_clusters = []
            for key, xy in enumerate(locations):
                if xy[0] >= x1 and xy[1] >= y1 and xy[0] < x2 and xy[1] < y2:
                    selected_clusters.append(clusters[key])

            # Turn selected clusters to numpy array
            selected_clusters = np.array(selected_clusters)

            # Plot
            unique = np.unique(clusters)
            bins = len(unique)
            segment_histogram = np.histogram(selected_clusters, bins=bins)[0]
            histogram.extend(segment_histogram)

    return np.array(histogram).astype("int64")
```

In [35]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [36]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [37]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [38]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 12 (3 marks)

Write a function `histogram_intersection_kernel(X, Y)` to compute Histogram Intersection Kernel which is also known as the Min Kernel and is calculated by

$$k(x, y) = \sum_{i=1}^{d} \min(x_i, y_i)$$

where d is the length of the feature vector.

Inputs

- `X` and `Y` are 2 dimensional numpy arrays of shape $M \times d$ and $N \times d$ respectively of data type `int64`.

Outputs

- This function should return a 2 dimensional numpy array of shape $M \times N$ of data type `float64`.

Data

- There is no specific data for this question. However, you can create your own data `X` and `Y` satisfying the input criteria.

Marking Criteria

- You will obtain full marks if and only if the kernel matrix calculated by your function exactly matches with the correct one. There is no partial marking for this question.

In [81]:
```
def histogram_intersection_kernel(X, Y):
    """
    Calculate the intersection (min) kernel between two sets of histograms.
    """
    # Get dimensions from input co-ordinates
    x_dim = X.shape[0]
    y_dim = Y.shape[0]

    # Initialise array to store results
    kernel = np.zeros((x_dim, y_dim))

    # Iterate through each pair of histograms
    for i, x in enumerate(X):
        for j, y in enumerate(Y):
            kernel[i][j] = np.sum(np.minimum(x, y))
    return kernel.astype("float64")
```

In [40]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 13 (1 mark)

Write a function `generalized_histogram_intersection_kernel(X, Y, alpha)` to compute Generalized Histogram Intersection Kernel which is computed by

$$k(x, y) = \sum_{i=1}^{d} \min(|x_i|^\alpha, |y_i|^\alpha)$$

where d is the length of the feature vector.

Inputs

- `X` and `Y` are 2 dimensional numpy arrays of shape $M \times d$ and $N \times d$ respectively of data type `int64`.
- `alpha` is a real number of data type `float`.

Outputs

- This function should return a 2 dimensional numpy array of shape $M \times N$ of data type `float64`.

Data

- There is no specific data for this question. However, you can create your own data `X` and `Y` satisfying the input criteria.

Marking Criteria

- You will obtain full marks if and only if the kernel matrix calculated by your function exactly matches with the correct one. There is no partial marking for this question.

In [82]:
```
def generalized_histogram_intersection_kernel(X, Y, alpha):
    """
    Calculate the intersection (min) kernel between two sets of histograms.
    """
    # Get dimensions from input co-ordinates
    x_dim = X.shape[0]
```

```
    y_dim = Y.shape[0]

    # Initialise array to store results
    kernel = np.zeros((x_dim, y_dim))

    # Iterate through each pair of histograms
    for i, x in enumerate(X):
        for j, y in enumerate(Y):
            kernel[i][j] = np.sum(np.minimum((abs(x) ** alpha), (abs(y) ** alpha))
    return kernel.astype("float64")
```

In [42]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 14 (1 mark)

Write a function `train_gram_matrix(X_tr, X_te)` which will compute the train gram matrix using the Histogram Intersection Kernel implemented above.

Inputs

- `X_tr` and `X_te` are 2 dimensional numpy arrays of shape $M \times d$ and $N \times d$ respectively of data type `int64`.

Outputs

- This function should return a 2 dimensional numpy array of data type `float64`.

Data

- There is no specific data for this question. However, you can create your own data `X_tr` and `X_te` satisfying the input criteria.

Marking Criteria

- You will obtain full marks if and only if the kernel matrix calculated by your function exactly matches with the correct one. There is no partial marking for this question.

In [83]:
```
def train_gram_matrix(X_tr, X_te):
    """
    Compute the train gram matrix using the histogram intersection kernel implement
    """
    matrix = histogram_intersection_kernel(X_tr, X_tr)
    return matrix.astype("float64")
```

In [44]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 15 (1 mark)

Write a function `test_gram_matrix(X_tr, X_te)` which will compute the test gram matrix using the Histogram Intersection Kernel implemented above.

Inputs

- `X_tr` and `X_te` are 2 dimensional numpy arrays of shape $M \times d$ and $N \times d$ respectively of data type `int64`.

Outputs

- This function should return a 2 dimensional numpy array of data type `float64`.

Data

- There is no specific data for this question. However, you can create your own data `X_tr` and `X_te` satisfying the input criteria.

Marking Criteria

- You will obtain full marks if and only if the kernel matrix calculated by your function exactly matches with the correct one. There is no partial marking for this question.

In [84]:
```
def test_gram_matrix(X_tr, X_te):
    """
    Compute the test gram matrix using the histogram intersection kernel implemente
    """
    matrix = histogram_intersection_kernel(X_te, X_tr)
    return matrix.astype("float64")
```

In [46]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 16 (5 marks)

Let $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ be two sets of corresponding/matching points respectively from two images I_1 and I_2. Further, let (R_1, T_1) and (R_2, T_2) be the camera parameters resepentively for the images I_1 and I_2. Write a function `reconstruct_3d(p1, p2, R1, R2, T1, T2)` to find the 3 dimensional coordinates of the points in p_1 and/or p_2.

Inputs

- p_1 and p_2 are 2 dimensional numpy arrays of shape $N \times 2$ of data type `float32`.
- R_1 and R_2 are 2 dimensional numpy arrays of shape 2×3 of data type `float32`.
- T_1 and T_2 are 1 dimensional numpy arrays of shape $(2,)$ of data type `float32`.

Outputs

- This function should return a numpy array of shape $N \times 3$ of data type `float32`.

Data

- There is not particular data for this question.

Marking Criteria

- You will get full marks if and only if answer returned by the implemented function matches with the true answer. There is no partial marking for this question.

In [85]:
```
def reconstruct_3d(p1, p2, R1, R2, T1, T2):
```

```
    """
    Find the 3-dimensional co-ordinates of the points in p1 and/or p2.
    """
    # Join rotational matrices
    A = np.concatenate((R1, R2), axis=0)

    # Get relative distances of each point from its respective camera
    t1 = np.transpose(p1 - T1)
    t2 = np.transpose(p2 - T2)

    # Join these translational matrices
    b = np.concatenate((t1, t2), axis=0)

    # Calculate the generalized inverse of A
    A_inv = np.linalg.pinv(A)

    # Multiply result against translational matrices
    result = np.matmul(A_inv, b)
    result_t = np.transpose(result)

    return result_t.astype("float32")
```

In [48]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 17 (4 marks)

Write a function `train_cnn(model, train_loader)` to train the following version of the Residual Network (ResNet) model on the EXCV10 (Exeter Computer Vision 10) dataset (available at this link). At the end of the training, this function should save the best weights of the trained CNN at: `data/weights_resnet.pth`. The EXCV10 dataset contains 10000 images from 10 classes which are further split into train (available at `train/` folder; total 8000 images with 800 images/class) and validation (available at `val/` folder; total 2000 images with 200 images/class) sets. For training your model, please feel free to decide your optimal hyperparameters, such as the number of epochs, type of optimisers, learning rate scheduler etc within the function, which can be done to optimise the performance of the model on the validation set.

Inputs

- `model` is an instantiation of ResNet class which can be created as follows: `ResNet(block=BasicBlock, layers=[1, 1, 1], num_classes=num_classes)`. An example of this can be found in the snippet in the following cell.
- `train_loader` is the training data loader. You can create the dataset and data loader for your training following the example in the cell below. Feel free to try other data augmentation and regularization techniques to train a better model.

Outputs

- This function should not necessarily return any output, instead it should save your best model at `data/weights_resnet.pth`.

Data

- You can train your model on the data available at https://empslocal.ex.ac.uk/people/staff/ad735/ECMM426/EXCV10.zip. As EXCV10 dataset is quite large in size, donot upload it with your submission.

Marking Criteria

- You will obtain full marks if the model weights saved at `data/weights_resnet.pth` can be loaded to a new instantiation of the model `ResNet(block=BasicBlock, layers=[1, 1, 1], num_classes=num_classes)`. You will not get any mark if your model is missing or saved in a different location or it cannot be loaded to the aforementioned model instance. Additionally, the quality of your trained model will be examined in the next question.

In [49]:

```
# ResNet model
from ca_utils import ResNet, BasicBlock
model = ResNet(block=BasicBlock, layers=[1, 1, 1], num_classes=1000)  # change num_

# Dataset
from torchvision import transforms, datasets

# Vanilla image transform
image_transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize(

# Dataset
import torchvision
train_data = torchvision.datasets.ImageFolder('train/', transform=image_transform)

# Data loader
from torch.utils.data import DataLoader
train_loader = DataLoader(train_data, batch_size=64, shuffle=True, num_workers=4, 
```

In [86]:

```
from torch.optim.lr_scheduler import ExponentialLR, MultiStepLR
from torch.utils.data import DataLoader
from tqdm.notebook import tqdm

import torch
import torch.nn.functional as F
import torch.optim as optim
import torchvision


class AverageMeter(object):
    """Computes and stores the average and current value"""

    def __init__(self):
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count
```

```
def train_cnn(model, train_loader):
    # Configuration
    device = "cpu"

    # Map to device
    model = model.to(device)

    # Make the parameters trainable
    for param in model.parameters():
        param.requires_grad = True
    # Optimiser
    learning_rate = 0.001
    momentum = 0.99
    optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=momentum)

    # Learning rate schedulers
    scheduler1 = ExponentialLR(optimizer, gamma=0.9)
    scheduler2 = MultiStepLR(optimizer, milestones=[10, 20, 30, 40, 50], gamma=0.1

    # Meter
    loss = AverageMeter()

    # Training loop
    num_epoch = 40
    for epoch in range(1, num_epoch + 1):
        model.train()
        tk0 = tqdm(train_loader, total=int(len(train_loader)))
        for batch_idx, (data, target) in enumerate(tk0):
            # Transfer the model to the required device
            data, target = data.to(device), target.to(device)

            # Compute the forward pass
            output = model(data)

            # Compute the loss function
            loss_this = F.cross_entropy(output, target)

            # Initialise the optimiser
            optimizer.zero_grad()

            # Compute the backward pass
            loss_this.backward()

            # Update the parameters
            optimizer.step()

            # Update the loss meter
            loss.update(loss_this.item(), target.shape[0])

        # Print the loss and scores
        print("Train: Average loss: {:.4f}\n".format(loss.avg))

        # Save the model
        torch.save(model.state_dict(), f"data/weights_resnet.pth")
        # test_cnn(model, test_loader)

        # Step the schedulers
        scheduler1.step()
        scheduler2.step()


# model = ResNet(block=BasicBlock, layers=[1, 1, 1], num_classes=10)
# image_transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize
#
```

```
# train_data = torchvision.datasets.ImageFolder("train/", transform=image_transforn
# train_loader = DataLoader(train_data, batch_size=64, shuffle=True, num_workers=1,
# test_data = torchvision.datasets.ImageFolder("val/", transform=image_transform)
# test_loader = DataLoader(test_data, batch_size=64, shuffle=False, num_workers=1,
#
# train_cnn(model, train_loader)
```

In [51]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 18 (12 marks)

Write a function `test_cnn(model, test_loader)` which will return the predicted labels by the `model` that you trained in the previous question for all the images supplied in the `test_loader` object. The test set will contain 3000 images (300 images/class) from the same distribution as of the EXCV10 dataset.

Inputs

- `model` is an instantiation of ResNet class which can created as follows `ResNet(block=BasicBlock, layers=[1, 1, 1], num_classes=num_classes)`. An example of this can be found in the cell below.
- `test_loader` is the data loader containing test data. The test data loader can be created following the example in the cell below. We will only use vanilla transformation to the test dataset.

Outputs

- This function should return a 1 dimensional numpy array of data type `int64` containing the predicted labels of the images in the `test_loader` object.

Data

- You can test your model on the `val` set of the data available at https://empslocal.ex.ac.uk/people/staff/ad735/ECMM426/EXCV10.zip. As EXCV10 dataset is quite large in size, please donot upload it with your submission.

Marking Criteria

- Your model will be tested based on average classification accuracy on a test set of 3000 images (300 images/class). You will obtain 50% marks if the obtained accuracy of your model on the test set is greater than or equal to 50%, 60% marks if your model obtains 55% accuracy or more, 70% marks if your model gets 60% accuracy or more, 80% marks if your model acquires 65% accuracy or more, 90% marks if your model wins 70% accuracy or more, and full marks if your model secures 75% accuracy or more. You will not obtain any mark if your model can not achieve 50% accuracy.

In [52]:
```
# Dataset
from PIL import Image
from torchvision import transforms, datasets
class EXCV10TestImageFolder(datasets.ImageFolder):
    def __init__(self, *args, **kwargs):
        super(EXCV10TestImageFolder, self).__init__(*args, **kwargs)
```

```
    def __getitem__(self, index):
        img_path = self.imgs[index][0]
        pic = Image.open(img_path).convert("RGB")
        if self.transform is not None:
            img = self.transform(pic)
        return img

# Vanilla image transform
image_transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize(

# Dataset
test_data = EXCV10TestImageFolder('val/', transform=image_transform)

# Data loader
from torch.utils.data import DataLoader
test_loader = DataLoader(test_data, batch_size=64, shuffle=False, num_workers=4, p
```

In [87]:
```
def test_cnn(model, test_loader):
    # config
    device = "cpu"
    # meters
    loss = AverageMeter()
    acc = AverageMeter()
    correct = 0
    # switch to test mode
    model.eval()
    results = []
    for data in test_loader:
        data = data.to(device)
        # since we dont need to backpropagate loss in testing,
        # we dont keep the gradient
        with torch.no_grad():
            # compute the forward pass
            # it can also be achieved by model.forward(data)
            output = model(data)
        # get the index of the max log-probability
        pred = output.argmax(dim=1, keepdim=True)
        for p in pred:
            results.append(p[0])
    return np.array(results)
```

In [54]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [55]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [56]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [57]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [58]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [59]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Question 19 (16 marks)

Write a function `count_masks(dataset)` which will count the number of faces correctly wearing mask (`with_mask` class), without mask (`without_mask` class) and incorrectly wearing mask (`mask_weared_incorrect` class) in the list of images `dataset` which is an instantiation of the `MaskedFaceTestDataset` class shown below. (**Hint**: You are expected to implement a 3 class (4 class with background) masked face detector which can detect the aforementioned categories of objects in a given image. However, you are absolutely free to be more innovative and come out with different solutions for this problem.)

Mask

Inputs

- `dataset` is an object of the `MaskedFaceTestDataset` class shown in the cell below.

Outputs

- This function should return a 2 dimensional numpy array of shape $N \times 3$ of data type `int64` whose values should respectively indicate the number of faces wearing mask, without mask and incorrectly wearing mask.

Data

- You can train and test your model on the data available at https://empslocal.ex.ac.uk/people/staff/ad735/ECMM426/MaskedFace.zip. This dataset contains some images and corresponding annotations (locations together with category information) of masked faces, which are split into `train` and `val` subsets. You can train your model on `train` set and decide your hyperparameters on the `val` sets. As MaskedFace dataset is quite large in size, please donot upload it with your submission.

Marking Criteria

- The evaluation will be done based on Mean Absolute Percentage Error (MAPE) which is defined as follows:

$$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - P_t}{\max(A_t, 1)}\right| \times 100$$

  where A_t is the true number and P_t is the predicted number of the corresponding class t in an image. For each image in `dataset`, MAPE will be computed, which will be averaged over all the images in `dataset`. You will obtain 50% marks if the obtained average MAPE of your model on the test set is lower than or equal to 30%, 62.5% marks if your model obtains 25% MAPE or less, 75% marks if your model gets 20% MAPE or less, 87.5% marks if your model acquires 15% MAPE or less, and full marks if your model secures 10% MAPE or less. You will not obtain any mark if your model can not achieve 30% MAPE.

In [60]:

```
# Dataset
import os, glob
from PIL import Image
from torch.utils.data import Dataset
class MaskedFaceTestDataset(Dataset):
    def __init__(self, root, transform=None):
        super(MaskedFaceTestDataset, self).__init__()
        self.imgs = sorted(glob.glob(os.path.join(root, '*.png')))
        self.transform = transform
```

```
    def __getitem__(self, index):
        img_path = self.imgs[index]
        img = Image.open(img_path).convert("RGB")
        if self.transform is not None:
            img = self.transform(img)
        return img

    def __len__(self):
        return len(self.imgs)
```

In [88]:
```
# Count masked faces
def count_masks(test_dataset):
    # YOUR CODE HERE
    raise NotImplementedError()
```

In [62]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [63]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [64]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [65]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [66]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

In [67]:
```
# This cell is reserved for the unit tests. Please leave this cell as it is.
```

Checkpoints

Checkpoints are very **IMPORTANT** for this course assessment. This step will ensure that you have implemented all the required functions and their expected outputs are structurally correct i.e. the outputs are consistent from shape, datatype and dimensionality perspective. However, passing these checkpoints will not ensure your implementations or answers are correct, which will be further checked via hidden unit tests after the submission. Please run the following two cells sequentially to run the checkpoints.

Please note that the execution of the second cell **should not take more than one minute**, which is actually the last checkpoint.

Initially, when none of the above functions is implemented, executing the following two cells should produce the following output:

Initial Log

Once you have all the required functions correctly implemented, executing the following two cells should produce the following output:

Final Log

In [68]:
```
# This cell will run all the answer cells
%reset -f
from IPython.display import Javascript
display(Javascript("Jupyter.notebook.execute_cells([2, 6, 9, 12, 15, 18, 28, 31, 3
```

In [90]:
```
# This cell will run the initial tests for questions
import os
import cv2
import time
import torch
import numpy as np
from termcolor import colored
from torch.utils.data import TensorDataset, Dataset, DataLoader
from ca_utils import load_interest_points

start_time = time.time()

# Test data
dummy_1 = np.random.randint(0, 255, size=(750, 750, 3), dtype="uint8")
dummy_2 = np.random.randint(0, 255, size=(750, 750), dtype="uint8")
k = np.random.randint(0, 2, size=(3, 3), dtype="uint8")
shapes = cv2.cvtColor(cv2.imread('data/shapes.png'), cv2.COLOR_BGR2RGB)
points = np.load('data/points.npy')
notre_dame_1 = cv2.cvtColor(cv2.imread('data/notre_dame_1.jpg'), cv2.COLOR_BGR2RGB
notre_dame_2 = cv2.cvtColor(cv2.imread('data/notre_dame_2.jpg'), cv2.COLOR_BGR2RGB
x1, y1, x2, y2 = load_interest_points('data/notre_dame_1_to_notre_dame_2.pkl')
N = 10000
nC = 100
X = np.random.randint(notre_dame_1.shape[1], size=(N, 1))
Y = np.random.randint(notre_dame_1.shape[0], size=(N, 1))
locations = np.concatenate((X, Y), axis=1)
clusters = np.random.randint(nC, size=N)
U = np.random.randint(50, size=(10, 50))
V = np.random.randint(50, size=(20, 50))
class CheckPointDataset(Dataset):
    def __init__(self, data):
        self.data = data
    def __getitem__(self, item):
        return self.data[item]
    def __len__(self):
        return len(self.data)
test_data = CheckPointDataset(torch.rand(8, 3, 224, 224))
test_loader = DataLoader(test_data, batch_size=2)
p1 = np.random.randint(0, 226, (100, 2), dtype="uint8")
p2 = np.random.randint(0, 226, (100, 2), dtype="uint8")
R1 = np.array([[0.9903, 0.0000, -0.1392], [0.0242, 0.9848, 0.1720]], dtype=np.floa
R2 = np.array([[1.0000, 0.0000, 0.00000], [0.0000, 0.9848, 0.1736]], dtype=np.floa
T1 = np.array([500, 160], dtype=np.float32)
T2 = np.array([500, 160], dtype=np.float32)

# Q1 initial test
try:
    output_1 = add_gaussian_noise(dummy_1, 0.0, 0.0)
    if isinstance(output_1, np.ndarray) and output_1.shape == (750, 750, 3) and ou
        print(colored("Q1. The 'add_gaussian_noise' function has passed the initia
    else:
        print(colored("Q1. The 'add_gaussian_noise' function cannot pass the initi
except (NotImplementedError, NameError):
    print(colored("Q1. The 'add_gaussian_noise' function cannot be found.", "red")

# Q2 initial test
try:
    output_2 = add_speckle_noise(dummy_1, 0.0, 0.0)
    if isinstance(output_2, np.ndarray) and output_2.shape == (750, 750, 3) and ou
        print(colored("Q2. The 'add_speckle_noise' function has passed the initial
    else:
        print(colored("Q2. The 'add_speckle_noise' function cannot pass the initia
except (NotImplementedError, NameError):
```

```
    print(colored("Q2. The 'add_speckle_noise' function cannot be found.", "red"))

# Q3 initial test
try:
    output_3 = cal_image_hist(dummy_2)
    if isinstance(output_3, np.ndarray) and output_3.ndim == 1 and output_3.shape[(
        print(colored("Q3. The 'cal_image_hist' function has passed the initial tes
    else:
        print(colored("Q3. The 'cal_image_hist' function cannot pass the initial te
except (NotImplementedError, NameError):
    print(colored("Q3. The 'cal_image_hist' function cannot be found.", "red"))

# Q4 initial test
try:
    output_4 = compute_gradient_magnitude(dummy_2, k, k)
    if isinstance(output_4, np.ndarray) and output_4.shape == (750, 750) and outpu
        print(colored("Q4. The 'compute_gradient_magnitude' function has passed the
    else:
        print(colored("Q4. The 'compute_gradient_magnitude' function cannot pass th
except (NotImplementedError, NameError):
    print(colored("Q4. The 'compute_gradient_magnitude' function cannot be found.",

# Q5 initial test
try:
    output_5 = compute_gradient_direction(dummy_2, k, k)
    if isinstance(output_5, np.ndarray) and output_5.shape == (750, 750) and outpu
        print(colored("Q5. The 'compute_gradient_direction' function has passed the
    else:
        print(colored("Q5. The 'compute_gradient_direction' function cannot pass th
except (NotImplementedError, NameError):
    print(colored("Q5. The 'compute_gradient_direction' function cannot be found.",

# Q6 initial test
try:
    output_6 = detect_harris_corner(shapes)
    if isinstance(output_6, np.ndarray) and output_6[0].shape[0] > 1 and output_6[(
        print(colored("Q6. The 'detect_harris_corner' function has passed the init:
    else:
        print(colored("Q6. The 'detect_harris_corner' function cannot pass the init
except (NotImplementedError, NameError):
    print(colored("Q6. The 'detect_harris_corner' function cannot be found.", "red

# Q7 initial test
try:
    output_7 = compute_homogeneous_rotation_matrix(points, 30)
    if isinstance(output_7, np.ndarray) and output_7.ndim == 2 and output_7.dtype =
        print(colored("Q7. The 'compute_homogeneous_rotation_matrix' function has p
    else:
        print(colored("Q7. The 'compute_homogeneous_rotation_matrix' function canno
except (NotImplementedError, NameError):
    print(colored("Q7. The 'compute_homogeneous_rotation_matrix' function cannot be

# Q8 initial test
try:
    output_8 = compute_sift(notre_dame_1, x1, y1)
    if isinstance(output_8, np.ndarray) and output_8.ndim == 2 and output_8.dtype =
        print(colored("Q8. The 'compute_sift' function has passed the initial test
    else:
        print(colored("Q8. The 'compute_sift' function cannot pass the initial test
except (NotImplementedError, NameError):
    print(colored("Q8. The 'compute_sift' function cannot be found.", "red"))

# Q9 initial test
try:
```

```
    output_9 = compute_sift(notre_dame_2, x2, y2)
    output_10 = match_features(output_8, output_9, x1, y1, x2, y2)
    if isinstance(output_10, tuple) and isinstance(output_10[0], np.ndarray) and ou
        print(colored("Q9. The 'match_features' function has passed the initial tes
    else:
        print(colored("Q9. The 'match_features' function cannot pass the initial te
except (NotImplementedError, NameError):
    print(colored("Q9. The 'match_features' function cannot be found.", "red"))

# Q10 initial test
try:
    output_11 = find_affine_transform(x1, y1, x2, y2)
    if isinstance(output_11, np.ndarray) and output_11.shape == (3, 3) and output_1
        print(colored("Q10. The 'find_affine_transform' function has passed the ini
    else:
        print(colored("Q10. The 'find_affine_transform' function cannot pass the in
except (NotImplementedError, NameError):
    print(colored("Q10. The 'find_affine_transform' function cannot be found.", "re

# Q11 initial test
try:
    output_12 = make_bovw_spatial_histogram(notre_dame_1, locations, clusters, [2,
    if isinstance(output_12, np.ndarray) and output_12.ndim == 1 and output_12.shap
        print(colored("Q11. The 'make_bovw_spatial_histogram' function has passed t
    else:
        print(colored("Q11. The 'make_bovw_spatial_histogram' function cannot pass
except (NotImplementedError, NameError):
    print(colored("Q11. The 'make_bovw_spatial_histogram' function cannot be found

# Q12 initial test
try:
    output_13 = histogram_intersection_kernel(U, V)
    if isinstance(output_13, np.ndarray) and output_13.ndim == 2 and output_13.dtyp
        print(colored("Q12. The 'histogram_intersection_kernel' function has passed
    else:
        print(colored("Q12. The 'histogram_intersection_kernel' function cannot pas
except (NotImplementedError, NameError):
    print(colored("Q12. The 'histogram_intersection_kernel' function cannot be foun

# Q13 initial test
try:
    output_14 = generalized_histogram_intersection_kernel(U, V, 0.6)
    if isinstance(output_14, np.ndarray) and output_14.ndim == 2 and output_14.dtyp
        print(colored("Q13. The 'generalized_histogram_intersection_kernel' functio
    else:
        print(colored("Q13. The 'generalized_histogram_intersection_kernel' functio
except (NotImplementedError, NameError):
    print(colored("Q13. The 'generalized_histogram_intersection_kernel' function ca

# Q14 initial test
try:
    output_15 = train_gram_matrix(U, V)
    if isinstance(output_15, np.ndarray) and output_15.ndim == 2 and output_15.dtyp
        print(colored("Q14. The 'train_gram_matrix' function has passed the initial
    else:
        print(colored("Q14. The 'train_gram_matrix' function cannot pass the initia
except (NotImplementedError, NameError):
    print(colored("Q14. The 'train_gram_matrix' function cannot be found.", "red")

# Q15 initial test
try:
    output_16 = test_gram_matrix(U, V)
    if isinstance(output_16, np.ndarray) and output_16.ndim == 2 and output_16.dtyp
        print(colored("Q15. The 'test_gram_matrix' function has passed the initial
```

```
    else:
        print(colored("Q15. The 'test_gram_matrix' function cannot pass the initial
except (NotImplementedError, NameError):
    print(colored("Q15. The 'test_gram_matrix' function cannot be found.", "red"))

# Q16 initial test
try:
    output_17 = reconstruct_3d(p1, p2, R1, R2, T1, T2)
    if isinstance(output_17, np.ndarray) and output_17.shape == (p1.shape[0], 3) an
        print(colored("Q16. The 'reconstruct_3d' function has passed the initial te
    else:
        print(colored("Q16. The 'reconstruct_3d' function cannot pass the initial t
except (NotImplementedError, NameError):
    print(colored("Q16. The 'reconstruct_3d' function cannot be found.", "red"))

# Q17 initial test
flag1 = os.path.isfile("data/weights_resnet.pth")
try:
    from ca_utils import ResNet, BasicBlock
    model = ResNet(block=BasicBlock, layers=[1, 1, 1], num_classes=10)
    cp = torch.load("data/weights_resnet.pth", map_location=torch.device("cpu"))
    model.load_state_dict(cp)
    flag2 = True
except (FileNotFoundError):
    flag2 = False
if flag1 and flag2:
    print(colored("Q17. The 'train_cnn' function has passed the initial test.", "gr
else:
    print(colored("Q17. The 'train_cnn' function cannot pass the initial test.", "r

# Q18 initial test
try:
    output_18 = test_cnn(model, test_loader)
    flag3 = True
except (NotImplementedError, NameError):
    flag3 = False
if flag3 and isinstance(output_18, np.ndarray) and output_18.ndim == 1 and output_1
    print(colored("Q18. The 'test_cnn' function has passed the initial test.", "gre
else:
    print(colored("Q18. The 'test_cnn' function cannot pass the initial test.", "re

# Q19 initial test
try:
    output_19 = count_masks(test_data)
    if isinstance(output_19, np.ndarray) and output_19.shape == (len(test_data), 3
        print(colored("Q19. The 'count_masks' function has passed the initial test
    else:
        print(colored("Q19. The 'count_masks' function cannot pass the initial test
except (NotImplementedError, NameError):
    print(colored("Q19. The 'count_masks' function cannot be found.", "red"))

# Execution time should be less than 1 minute
tot_time = time.time() - start_time
if tot_time > 60:
    print(colored("Execution took {} which is higher than the time limit and should
else:
    print(colored("Execution took {} which met the time criteria.".format(time.str
```

```
Q1. The 'add_gaussian_noise' function has passed the initial test.
Q2. The 'add_speckle_noise' function has passed the initial test.
Q3. The 'cal_image_hist' function has passed the initial test.
Q4. The 'compute_gradient_magnitude' function has passed the initial test.
Q5. The 'compute_gradient_direction' function has passed the initial test.
Q6. The 'detect_harris_corner' function has passed the initial test.
Q7. The 'compute_homogeneous_rotation_matrix' function has passed the initial tes
t.
Q8. The 'compute_sift' function cannot be found.
Q9. The 'match_features' function cannot be found.
Q10. The 'find_affine_transform' function has passed the initial test.
Q11. The 'make_bovw_spatial_histogram' function has passed the initial test.
Q12. The 'histogram_intersection_kernel' function has passed the initial test.
Q13. The 'generalized_histogram_intersection_kernel' function has passed the initi
al test.
Q14. The 'train_gram_matrix' function has passed the initial test.
Q15. The 'test_gram_matrix' function has passed the initial test.
Q16. The 'reconstruct_3d' function has passed the initial test.
Q17. The 'train_cnn' function has passed the initial test.
Q18. The 'test_cnn' function has passed the initial test.
Q19. The 'count_masks' function cannot be found.
Execution took 00:00:00 which met the time criteria.
```

In [ ]:

In [ ]: