ECMM426: Computer Vision
Coursework

University's Policy: Referencing, and academic conduct
You are required to cite the work of others used in your solution and include a list of references, and must avoid plagiarism, collusion and any academic misconduct behaviours (link).

Submission: The project will be submitted in a ZIP file, on the 6th of March 2024, before noon. As usual, late penalties will apply.

Assessment: The project will be worth **60%** of your module mark.

Important note: The submitted ZIP file must contain **the code** as well as **the saved outputs**. It must also include **a report outlining the output filenames for every question as well as any other outputs** that's required in the questions (For instance, accuracy percentage for Question 5 and MAPE score for Question 7)

Question 1 (14 marks)
Write two functions *compute_gradient_magnitude(gr_im, kx, ky)* and *compute_gradient_direction(gr_im, kx, ky)* to compute the magnitude and direction of gradient of the grey 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*.

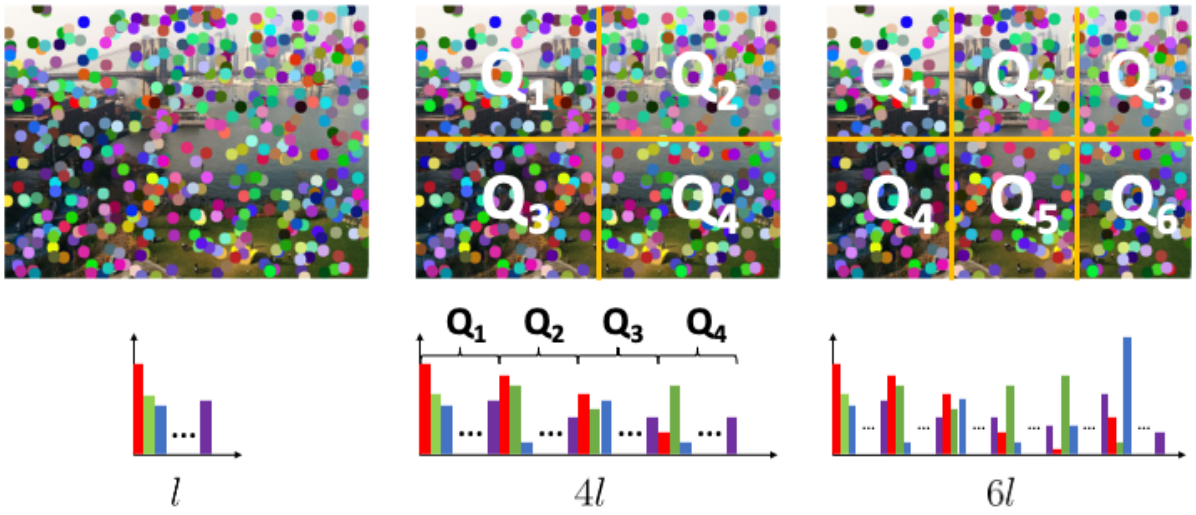
Outputs
- The expected outputs are two 2-dimensional numpy array of the same shape as of *gr_im* and of data type *float64*.

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

Marking Criteria
- The outputs should exactly match with the correct gradient magnitude and gradient direction of a given grey image *gr_im* to obtain the full marks. There is no partial marking for this question.

Question 2 (20 marks)
Write a function *generate_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 quantised 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. You have to showcase three test cases to call the above function in order to calculate the bag-of-visual-words spatial histograms on the image *im* showcasing coarse and fine divisions.



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 quantised cluster id.
- *division* is a list of integers of length 2.

Outputs

- This function should return a 1-dimensional numpy array of data type *int64.* The results for all three test cases should be saved and submitted along with the information in the report on the division numbers and the corresponding output filenames.

Data

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

Marking Criteria

- 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.

Question 3 (10 marks)

Write a function *compute_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 with an angle θ (theta in the definition) in the anticlockwise direction about the centre 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 2-dimensional 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 partial mark for this question.

Question 4 (10 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 below snippet.

- *train_loader* is the training data loader. You can create the dataset and data loader for your training following the example in the snippet below. Feel free to try other data augmentation and regularisation techniques to train a better model.

```python
# ResNet model
from ca_utils import ResNet, BasicBlock
model = ResNet(block=BasicBlock, layers=[1, 1, 1], num_classes=1000)  # change num_classes if needed, this is an example

# Dataset
from torchvision import transforms, datasets

# Vanilla image transform
image_transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# 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, pin_memory=True)
```

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, do not 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.

Question 5 (15 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).*
- *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.
- This function should also return the classification accuracy as a percentage

```python
# 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((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# 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, pin_memory=True)
```

Data

- You can test your model on *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, do not 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 cannot achieve 50% accuracy.
- In the case you only provide the first output and not the classification accuracy, the abovementioned evaluation percentages will be calculated from 7.5 marks instead of 15 marks.

Question 6 (6 marks)

Write *compute_confusion_matrix(ture,predictions)* to compute the confusion matrix for the output labels generated in Question 5.

Inputs

- *true* is a 1-dimensional array of true labels for the test data in Question 5
- *predictions* is a 1-dimensional array of outputs from Question 5

Data

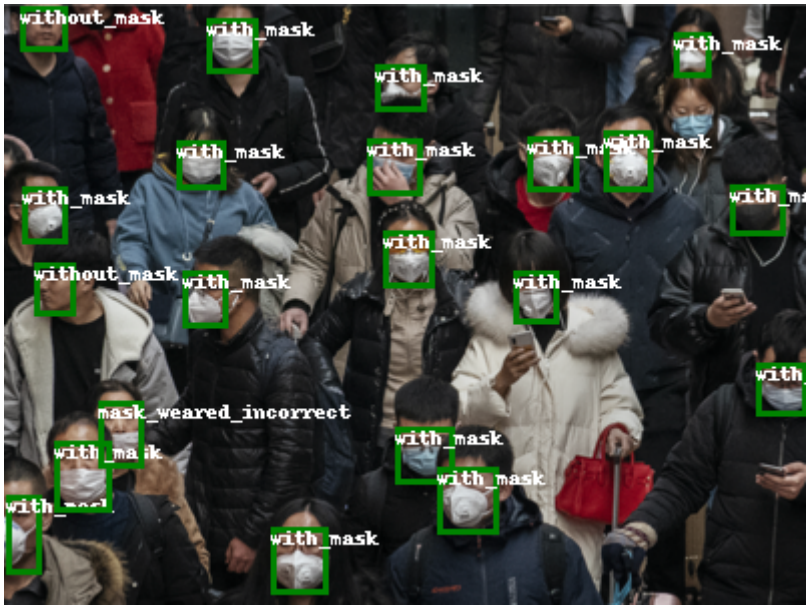
- You can use the output and true labels of Question 5 as your data

Marking Criteria

- The output should exactly match the correct confusion matrix resulting from the output in Question 5 to obtain the full marks. There is no partial marking for this question.

Question 7 (25 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.)




```python
# 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)
```

Inputs

- *dataset* is an object of the *MaskedFaceTestDataset* class shown in the above code snippet.

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.

- This function should also return the Mean Absolute Percentage Error (MAPE) score using the below formula.

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

  where A_t is the true number and P_t is the predicted number of the corresponding class t in an image. MAPE should be computed for each image in dataset, which will be averaged over all the images in dataset.

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 do not upload it with your submission.

Marking Criteria

- 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 cannot achieve 30% MAPE.
- In the case, you only provide the first output and not the MAPE score, your abovementioned evaluation percentages will be calculated from 12.5 marks instead of 25 marks.