Questions

Question 1 (15 marks)

Write two functions, `compute_gradient_magnitude(gr_image)` and `compute_gradient_direction(gr_image)`, to compute the gradient magnitude and direction of an image using Sobel filters.

Input:

- gr_image is a 2-dimensional numpy array of data with values in $[0, 255]$.

Output:

- The expected outputs are two 2-dimensional numpy arrays of the same shape as the input image.

Data:

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

Question 2 (10 marks)

Write a function, `compute_colour_histogram(image, num_bins)`, that computes the colour histogram of an input image. The function should separately calculate histograms for the R, G, and B channels.

Input:

- `image`: A 3-dimensional numpy array of shape $(height, width, 3)$ with values in $[0, 255]$.
- `num_bins`: The number of bins to use for each channel.

Output:

- Three 1-dimensional numpy arrays, each containing the histogram for the R, G, and B channels.

Data:

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

Question 3 (15 marks)

Write a function, `compute_transform_matrix(points, theta, scale, translation)`, to compute a combined rotation, scaling, and translation matrix for a given set of points. This matrix should perform the following transformations:

1. Counterclockwise rotation by θ degrees around the centre of the points.
2. Uniform scaling by a factor of $scale$.
3. Translation by $[t_x, t_y]$.

Input:

- `points`: A 2-dimensional numpy array of shape $(N, 2)$, where N is the number of points.

- `theta`: A scalar specifying the rotation angle in degrees.
- `scale`: A scalar specifying the uniform scaling factor.
- t_x, t_y: Scalars specifying the translation in x- and y-directions, respectively.

Output:

- A single 3x3 transformation matrix that combines the specified rotation, scaling, and translation operations.

Data:

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

Question 4 (10 marks)

Write a function, `train_cnn()`, to train a Convolutional Neural Network (CNN) using PyTorch. The CNN should classify images from the CIFAR-10 dataset, focusing only on the animal classes. The dataset contains 10 classes, out of which the following 6 are animal categories: bird, cat, deer, dog, frog and horse.

Input:

1. Subset of CIFAR-10 dataset (**Training set**) to include only the specified animal classes. If required, use a subset of Training set (10% or 20%) as your validation set.
2. Define a ResNet model with the following configuration:

   `ResNet(block=BasicBlock, layers=[1, 1, 1], num_classes=6)`
3. Use a data loader (`torch.utils.data.DataLoader`) for batching. You can create the dataset and data loader for your training (See the below example). Feel free to try other data augmentation and regularisation techniques to train a better model.

```
# 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)
```

Output:

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

Data:

- Use `torchvision.datasets.CIFAR10` for dataset loading.

Question 5 (20 marks)

Write a function `test_cnn()` which will return the predicted labels by the model that you trained in the previous question for all the images supplied in test set. The test set will be the **Test set** of the CIFAR-10 dataset (The subset that only contains the specified animal classes).

Input:

- Model is an instantiation of ResNet class which can be created as follows:

  `ResNet(block=BasicBlock, layers=[1, 1, 1], num_classes=6)`

- Use the data loader (`torch.utils.data.DataLoader`) to create the test data loader.

Output:

- 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

Data:

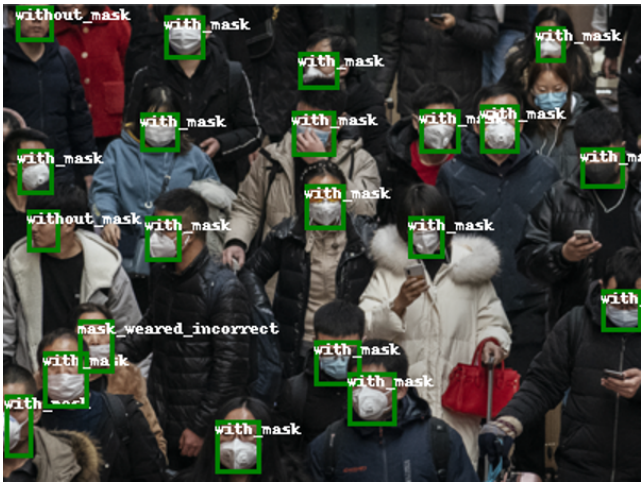
Use the **Test set** of CIFAR-10 dataset on `torchvision.datasets.CIFAR10` (The subset that only contains the specified animal classes: bird, cat, deer, dog, frog and horse.

Marking Criteria:

- Your model will be tested based on average classification accuracy on the test set. 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 one output, the abovementioned evaluation percentages will be calculated from 10 marks instead of 20 marks.

Question 6 (30 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)
```

Input:

- Dataset is an object of the `MaskedFaceTestDataset` class shown in the above code snippet.

Output:

- This function should return a 2-dimensional numpy array of shape $\times 3$ of data type `int64` whose values should respectively indicate the number of all the classes in the dataset.
- 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.
- You should also save and submit your best model at `data/weights_counting.pth`.