Due Date: February 27th, 23:59

Instructions

- *This assignment is involved – please start well ahead of time.*
- *For all questions, show your work!*
- *Submit your report (PDF) and your code electronically via the course Gradescope page.*
- *For open-ended experiments (i.e., experiments that do not have associated test-cases), you do not need to submit code – a report will suffice.*
- *TA for this assignment* ***Pierre-Louis Benveniste****.*

Summary:

Coding instructions You will be required to use PyTorch to complete all questions. Moreover, this assignment **requires running the models on GPU** (otherwise it will take an incredibly long time); if you don't have access to your own resources (e.g. your own machine, a cluster), please use Google Colab. During development, you can use the unit tests defined in `test.py` to help you. Remember, these are only basic tests, and more tests will be run on Gradescope.

Submission You are expected to submit `mlp.py`, `utils.py`, `mobileNet.py` and `unet.py` to gradescope for autograding, as well as a PDF report for experimental and open-ended problems.

Medical image classification

Problem 1

Implementing an MLP (10 pts) In this problem, you will implement an MLP. An MLP consists of multiple linear layers each followed by a non-linear function. You can find the code template in `mlp.py`.

1. You are expected to complete `class Linear` in the file `mlp.py`. The class must have two attributes: `weight` and `bias`. Complete the forward function accordingly.
2. Now you can move on to the `class MLP`. In `_build_layers`, you are expected to build the hidden layers as well as the output (classification) layer. In `_initialize_linear_layer`, you are expected to fill the bias with zeros, and use the Glorot & Bengio (2010) normal initialization for the weights.
3. Complete `activation_fn` so that it can process inputs according to different `self.activation`. Complete the forward function according to docstring.

Problem 2

In this second problem, we will focus more deeply on convolutions. The mathematical definition of a convolution, where f is the input, g the kernel and s the feature map, is the following:

$$s(t) = (f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\, d\tau$$

In this problem, we will first focus on applying convolutions on 2D images. Given an image I and a kernel K, the value of the feature map at coordinates (i, j) is:

$$S(i,j) = (I * K)_{i,j} = \sum_m \sum_n I_{m,n} K_{i-m,j-n}$$

As explained in the book *Deep Learning* page 328, many neural network libraries implement a related function called the cross-correlation, which is the same as convolution but without flipping the kernel. We will use this definition in this problem.

Figure 1: Image of a multipla for experiments on convolutions. Source: Motor1. https://es.motor1.com/news/731567/fiat-multipla-matriculado-nuevo-2024/

$$S(i,j) = (K * I)_{i,j} = \sum_m \sum_n I_{i+m,j+n} K_{m,n}$$

For the following questions, you can use an image of your choice with enough textures to be interesting when applying convolutions. Otherwise, you can use the image in Figure 1.

1. (5pts : 2pts code + 3 pts report) Complete the function `discrete_2d_convolution` in the file `utils.py`. Compare the speed performance of this function with the speed of the function `scipy.signal.convolve2d`. Explain the difference in computation speed.
2. (5 pts) Propose a kernel to blur the image. Apply the blurring kernel to the image and show the resulting image. Explain in your own words why it performs blurring.
3. (5 pts) Propose and apply kernels used for the identification of horizontal and vertical edges. Show the results of the two kernels used here.

Now, we will implement the **MobileNet** model. MobileNet was developed by a team of researchers at Google in 2017. They aimed to design an efficient and light-weight CNN for mobile devices. MobileNet uses depthwise separable convolutions to significantly reduce the number of parameters compared to other networks with regular convolutions and the same depth in the nets.

A depthwise separable convolution is a form of factorized convolutions which factorize a standard convolution into a depthwise convolution and a 1×1 convolution called a pointwise convolution. For MobileNets the depthwise convolution applies a single filter to each input channel. The pointwise convolution then applies a 1×1 convolution to combine the outputs of the depthwise convolution (source: https://arxiv.org/pdf/1704.04861). Figure 2 shows an illustration of a depthwise separable convolution.

Figure 3 shows the Depthwise Separable convolutions blocks used in the MobileNet architecture. Figure 4 shows the full architecture of the MobileNet.

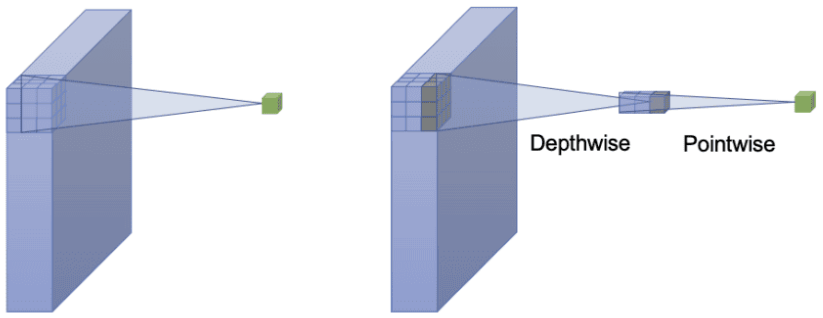

Figure 2: Left: standard convolution. Right: depthwise seperable convolution. Source: https://viso.ai/deep-learning/mobilenet-efficient-deep-learning-for-mobile-vision/

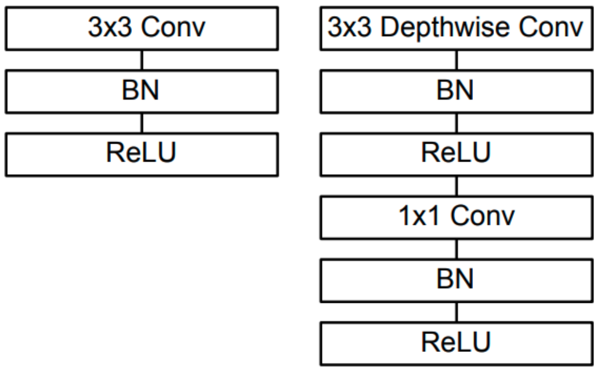

Figure 3: Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU. Source: https://arxiv.org/pdf/1704.04861

4. (5 pts) Based on Figure 3, implement the `DepthwiseSeparableBlock` class in `mobileNet.py`.
5. (10 pts) Based on Figure 4, implement the `MobileNet` class in `mobileNet.py`. To simplify the problem, we will not take into account and incorporate the α and ρ parameters. In the "Filter Shape" column, you should adapt the model so that the FC/s1 layer adapts to the number of classes passed as input to the model.

In this part of the assignment, we will work on the classification of medical images. The dataset used here is **PathMNIST**. PathMNIST is a set of 107,180 images based on a prior study for predicting survival from colorectal cancer histology slides. The dataset is comprised of 9 types of tissues, resulting in a multi-class classification task. Each images were resized to 3×28×28. More details here: https://medmnist.com/.

6. (10 pts) Using the code in `main_classification.py` and without changing the hyperparameters, compare the performance of your `MLP` network and `MobileNet`. Plot the training curves and conclude.

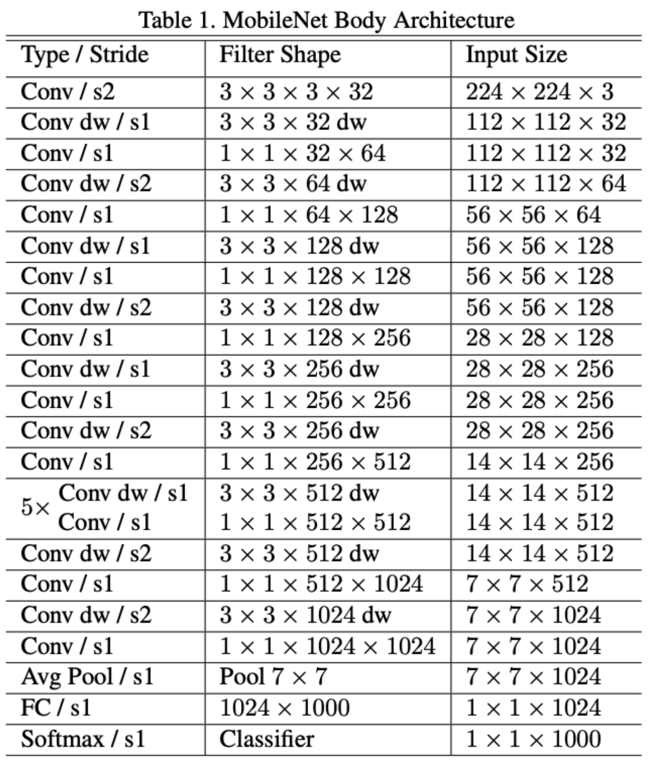

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
5× Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 4: MobileNet Body Architecture. The input sizes in the right column do not represent the example studied in this problem.

Image segmentation

Problem 3

Implement a UNet In this problem, you are going to implement a UNet model for semantic segmentation.

In semantic segmentation, the predicted segmentations are often evaluated using the Dice score. Given two sets, X and Y, where $|X|$ and $|Y|$ are the cardinalities of the two sets (i.e. the number of elements in each set), it is defined as:

$$DiceScore = \frac{2|X \cap Y|}{|X| + |Y|}$$

To optimize the model, Dice Loss is frequently employed, which is mathematically defined as:

$$DiceLoss = 1 - DiceScore$$

In practice, a hybrid objective function known as DiceCELoss, a combination of Cross-Entropy and Dice Loss, is often preferred to leverage the strengths of both. For the purpose of this assignment, we define DiceCELoss as:

$$DiceCELoss = 0.5DiceLoss + 0.5CE$$

1. (10 pts) Implement the `DecoderBlock` and the `UNet` in `unet.py`. Please provide your solution in `unet.py`. Use the parameters of the architecture provided in Figure 5. In this problem, the double convolution uses padding ($padding = 1$) so that the output has the same size as the input.

2. (5 pts) Implement the `DiceLoss` and `DiceCELoss` in `utils.py`.

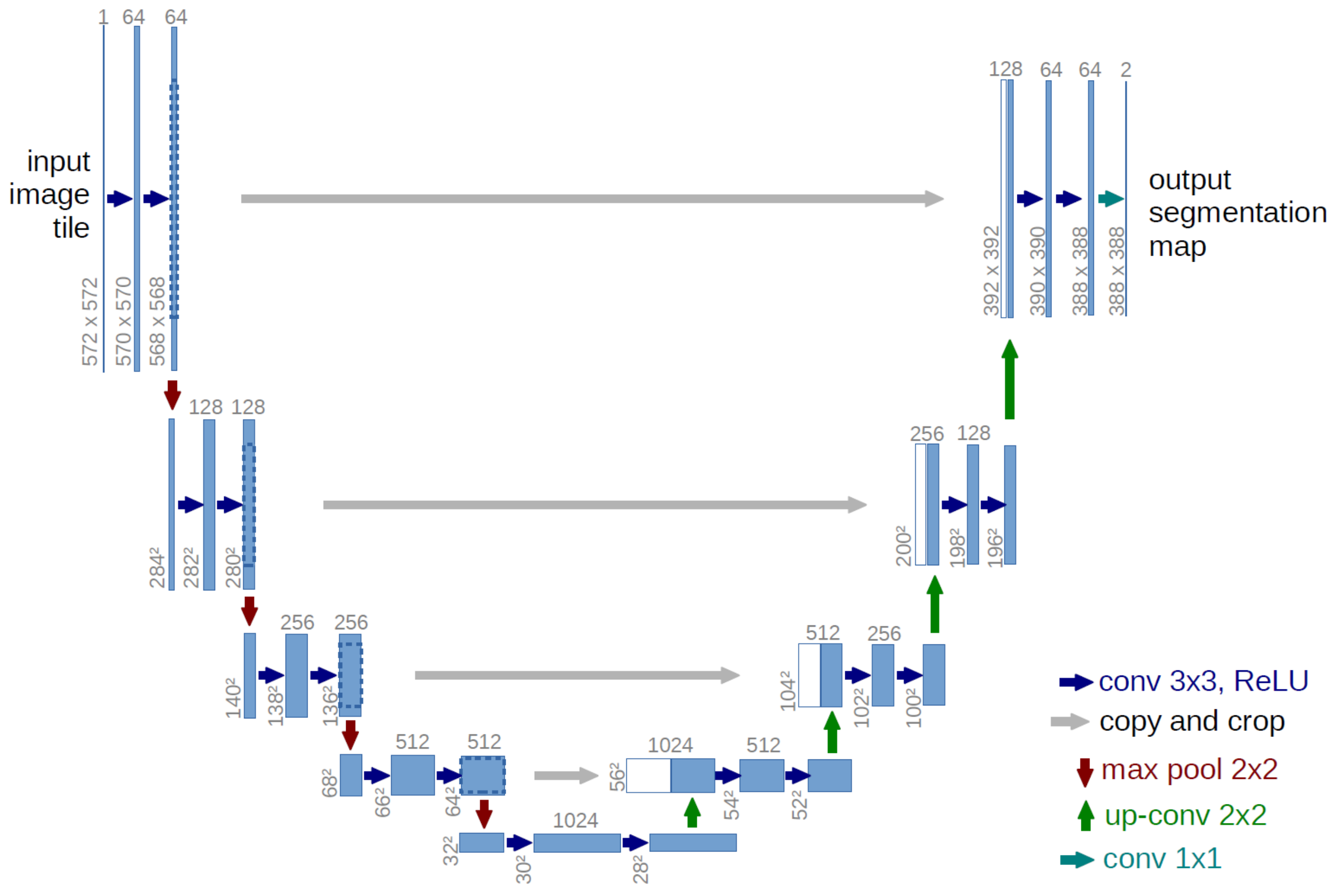

Figure 5: UNet architecture. Source: 10.1007/978-3-319-24574-4_28. The input sizes on this figure do not reflect the sizes of the input used in this problem. More details on the architecture can be found in the paper.

Problem 4

In this problem, you are going to work on the segmentation of retina blood vessel. The dataset can be downloaded here: https://drive.google.com/drive/folders/1__5iCDNvLJ809LTZe7CvQjK_IXNBEa8L?usp=sharing. You can use the file `main_segmentation.py` to get started. Please submit a PDF report to answer the following questions.

1. (10 pts) Build the UNet model without the skip connections. You can choose how to build it (no code submission is required for this question). Compare the performances with the nnUNet with skip connections and conclude.
2. (5 pts) Investigate the effect of learning rate with the Adam optimizer. Perform experiments with learning rates of 0.1, 0.01, 0.001, 0.0001, 0.00001. Provide the figures and explain your findings.
3. (10 pts) Investigate 4 different data augmentation strategies. Display examples of data augmentations used. Conclude which augmentations work best and why.
4. (10 pts) Use a pretrained model and finetune it on the training set. Compare its performance with the same UNet trained from scratch. Conclude on the observations. You can use the following pretrained model from the `segmentation-models-pytorch` library: `smp.Unet(encoder_name="resnet18", encoder_weights="imagenet", ...)`.