ECE-GY 6123 Image and Video Processing, Fall 2023
Computer Assignment 4: CNN Segmentation
Due 11/17/2023 at 11:59 PM

Please submit your solutions, including Python code and outputs (when relevant), **as a single PDF file to GradeScope**.

In this assignment, you will implement and train a convolutional neural network (CNN) for binary segmentation of pedestrian images (see Figure 1). Your implementation should be done using `PyTorch`, for which you can follow the installation instructions here.

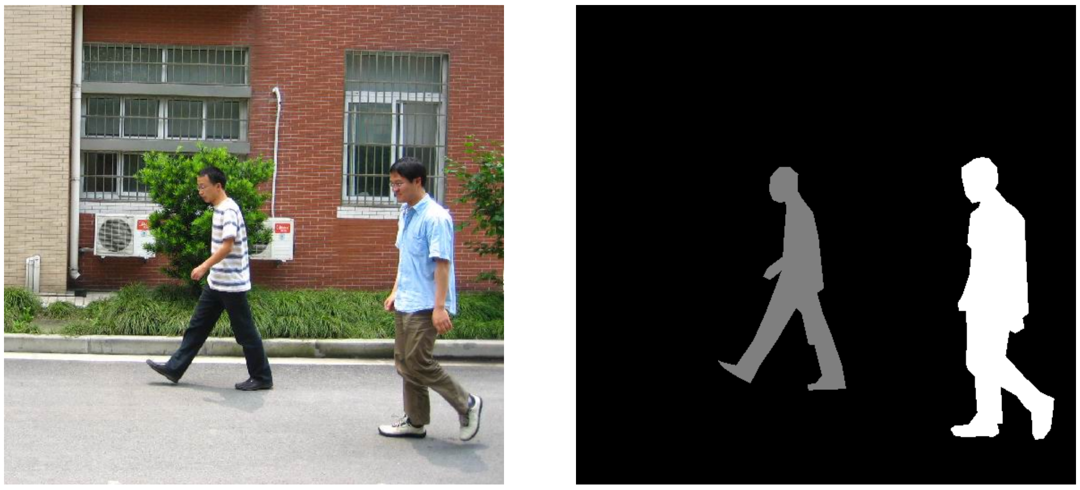

Figure 1: An input-target sample from the Fudan Pedestrian Segmentation dataset [1]. Left: example image. Right: corresponding segmentation mask.

Problem description: Your goal is to implement and train a CNN on the Fudan Pedestrian segmentation dataset (FudanPed) [1] for binary segmentation of pedestrians. Your trained network should be able to produce a probability mask for the scene, with which you can threshold at $p = 0.5$ during inference to decide if a pedestrian is at the pixel location or not.

The FudanPed [1] dataset contains 170 samples of RGB image - segmentation mask pairs. An example is given in Figure 1. Note that the provided segmentation mask assigns a different value to each instance of a pedestrian in the image (notice in Fig. 1 a gray mask and white mask). For your task, you need to binarize the mask by setting all values greater-than or equal-to 1 to 1. This converts the original dataset from the instance-segmentation task to the binary segmentation task.

The DICE score of two sets A and B is defined as,

$$\text{DICE}(A, B) = \frac{2|A \cap B|}{|A| + |B|}.$$

It measures the overlap between A and B, normalized by the size of each set individually[1]. Use the DICE score to evaluate the binarized output of your network against the ground-truth segmentation mask. To train a network, you need a "soft" loss function, i.e. one that allows gradients to pass through by not clipping values. A common loss function is the so-called "soft-dice" loss,

$$\mathcal{L}_{\text{SoftDice}}(\boldsymbol{p}, \boldsymbol{q}) = 1 - \frac{2\sum_i \boldsymbol{p}_i \boldsymbol{q}_i + \epsilon}{\sum_i \boldsymbol{p}_i + \boldsymbol{q}_i + \epsilon},$$

where $\boldsymbol{p}, \boldsymbol{q}$ are the un-binarized output mask of your neural network and the ground-truth binary segmentation mask, and $\epsilon \approx 1$ is used for numerical stability. Note that this loss is symmetric. Other losses, such as binary cross entropy, are also commonly used.

Requirements: The following items are expected to be completed in your code and commented on in your report.

(a) Cut the FudanPed dataset into an 80-10-10 train-val-test split.

(b) Apply data augmentation to your dataset during training and show an example of your data augmentation in your report.

(c) Implement and train a CNN for binary segmentation on your train split. Describe your network architecture[2], loss function, and any training hyper-parameters. You may implement any architecture you'd like, but **the implementation must be your own code.**

(d) Report training loss, validation loss, and validation DICE curves. Comment on any overfitting or underfitting observed.

(e) Report the average dice score over your test-set. **You should be able to achieve a score of around** 0.7 **or better**.

(f) Show at least 3 example segmentations (i.e. show the RGB image, mask, and RGB image $\times$ mask for 3 samples) from your training data and 3 from your testing data. Comment on the generalization capabilities of your trained network.

(g) Show at least 1 example segmentation on an input image **not from the FudanPed dataset**. Again, comment on the generalization capabilities of your network with respect to this "out-of-distribution" image.

Additionally, all code should be attached to the end of your report (unless submitting a Jupyter Notebok).

[1]This is not the same as the intersection over union (IOU) score (think about the set union formula).
[2]a reference or figure is fine

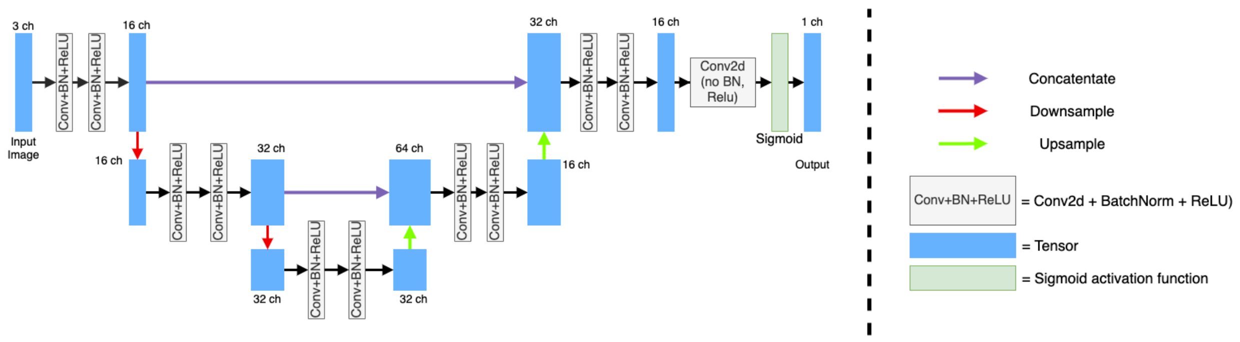

Figure 2: UNet architecture [2]. Note that concatenation is along the channel dimension.

Suggestions:

- It is strongly suggested that you implement the UNet architecture [2], as described in Figure 2. You can implement downsampling via 2×2 Max-pooling and upsampling by bilinear upsampling.
- Experiment with your hyperparamters scientifically! For example, increase batch size or decrease the your initial learning rate to combat noisy gradients. A good starting point is,
  - `NUM_EPOCHS = 40`, `batchsize = 8`, `learning_rate=0.001`
- Schedule your learning rate to decrease over time.
- Remember: do not binarize your output mask during training.
- Rescale your input images to a smaller size to speed up training (ex. 64x64, 96x96, or 128x128).
- Use a sigmoid activation at the output of your network.
- Save a checkpoint of your best performing network (as determined by val-loss) during training.
- Refer to the following PyTorch Tutorial for creating custom datasets/dataloaders. You can also refer to the PyTorch tutorial material on BrightSpace.

References

[1] G. S. I.-f. S. Liming Wang, Jianbo Shi, *Fudan Pedestrian Segmentation Dataset*, 2007. [Online]. Available: https://www.kaggle.com/datasets/jiweiliu/pennfudanped

[2] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional networks for biomedical image segmentation,” *Medical Image Computing and Computer-Assisted Intervention*, p. 234–241, 2015. [Online]. Available: https://arxiv.org/abs/1505.04597