Image Segmentation

Semantic Segmentation Exercises

Introduction to Deep Learning in Computer Vision

October 2025

(*Based on exercise from DLCV 2019 by ASM Shihavuddin and Ricardo Cruz*)

The data used for this exercise stems from the ISBI cell tracking challenge http://celltrackingchallenge.net/. The data loader below assumes that you are working on the HPC machines with the data located at

`/dtu/datasets1/02516/phc_data`

If this is not the case, you will have to adapt the data loader accordingly.

These are microscopy images taken over time of different cells that move around in the image plane; the images you will be working with have been processed to contain only a single cell each.

Have a look at the phc_data folder. It is structured like this:

```
images/
    img_00000.jpg
    img_00001.jpg
        ...
labels
    label_00000.png
    label_00001.png
        ...
```

We provide you with a class similar to the Hotdog_NotHotdog data loader that will allow you to load the dataset from /dtu/datasets1/02514/

Exercise 1: Train an auto-encoder.

The files that you will use are:

- `train.py`: 1) Imports the dataset, loss, network architecture, 2) trains the model, and 3) saves the trained model.
- `predict.py`: 1) Loads a saved model, 2) loads the test set, and 3) generates and saves the segmentation masks from the test set.
- `model/EncDecModel.py`: Auto-encoder architecture.
- `dataset/PhCDataset.py`: Dataloader.

Loss

As important as building the architecture is defining **the optimizer** and **the loss function**. The **loss function** is what we are trying to minimize. Many can be used. A popular one for binary segmentation is *binary cross-entropy* which is given by

$$\mathcal{L}_{BCE}(y, \hat{y}) = -\sum_i \left[y_i \log \sigma(\hat{y}_i) + (1 - y_i) \log(1 - \sigma(\hat{y}_i))\right].$$

where y is the desired output and $\hat{y}$ is the output of the model. σ is the *logistic* function (also called *sigmoid*), which converts a real number $\mathbb{R}$ into a probability $[0, 1]$.

However, this loss suffers from numerical stability problems. Most importantly, $\lim_{x\to 0} \log(x) = \infty$ which leads to optimization unstability. This loss can be simplified into the following loss, which is equivalent and is not as prone to numerical unstability:

$$\mathcal{L}_{BCE} = \hat{y} - y\hat{y} + \log\left(1 + \exp(-\hat{y})\right).$$

Question. Can you still see situations in which this loss might be unstable? How would you change it?

Exercise 2: Implement Dice loss, Focal loss and Total Variation regularizer.

The files that you will use are:

- `lib/losses.py (DiceLoss)`: Implement Dice loss.
- `lib/losses.py (FocalLoss)`: Implement Focal loss.
- `lib/losses.py (BCELoss_TotalVariation)`: Implement Total Variation.

Dice loss

Given two masks X and Y, a common metric to measure the distance between these two masks is given by:

$$D(X,Y) = \frac{2|X \cap Y|}{|X| + |Y|}$$

This function is not differentiable and the loss function **must** always be differentiable for gradient descent to work. But we can approximate it using:

$$\mathcal{L}_D(X,Y) = 1 - \frac{mean(2X \cdot Y + 1)}{mean(X+Y)+1},$$

where $X \cdot Y$ is element-wise product of your image of pixel-wise confidences

Focal loss

First, remember how binary cross entropy looks like:

$$\mathcal{L}_{BCE}(y,\hat{y}) = -\sum_i \left[y_i \log \sigma(\hat{y}_i) + (1-y_i)\log(1-\sigma(\hat{y}_i))\right].$$

The problem with this loss is that it tends to benefit the **majority** class (usually the background) relative to the **minority** class (usually the foreground). Therefore, usually people apply weights to each class:

$$\mathcal{L}_{wBCE}(y,\hat{y}) = -\sum_i \alpha_i \left[y_i \log \sigma(\hat{y}_i) + (1-y_i)\log(1-\sigma(\hat{y}_i))\right].$$

Traditionally, the weight α_i is defined as the inverse frequency of the class of that pixel i, so that observations of the minority class weight more relative to the majority class.

Another recent addition has been the **focal loss** which weights each pixel by the confidence we have in the prediction of that pixel.

$$\mathcal{L}_{focal}(y,\hat{y}) = -\sum_i \left[(1-\sigma(\hat{y}_i))^\gamma \, y_i \log \sigma(\hat{y}_i) + (1-y_i)\log(1-\sigma(\hat{y}_i))\right].$$

A good value for γ is generally 2.

* Lin, Tsung-Yi, et al. “Focal loss” Proceedings of the IEEE international conference on computer vision. 2017.

Regularization

The more of our prior knowledge we can feed the neural network, the better. One way to introduce such background knowledge is by regularization. Regularization are extra terms added to the loss function.

Centered: if you know the segmentation is always centered, you can punish it if it is not.

$$\mathcal{L}_{centered}(y, \hat{y}) = 1 - \sigma(\hat{y}_{w/2,h/2})$$

Sparsity: if there is a lot of sparsity (small patches of segmentation), you can add an L1 term:

$$\mathcal{L}_{sparcity} = \sigma(\hat{y})$$

Contiguity: to avoid many zig-zags in the semantic frontier, total variation can be used:

$$\mathcal{L}_{contiguity} = \sum_{i,j} |\sigma(\hat{y}_{i+1,j}) - \sigma(\hat{y}_{i,j})| + \sum_{i,j} |\sigma(\hat{y}_{i,j+1}) - \sigma(\hat{y}_{i,j})|$$

* Ferreira, P. M., Marques, F., Cardoso, J. S., & Rebelo, A. (2018). ”Physiological Inspired Deep Neural Networks for Emotion Recognition”. IEEE Access, 6, 53930-53943.

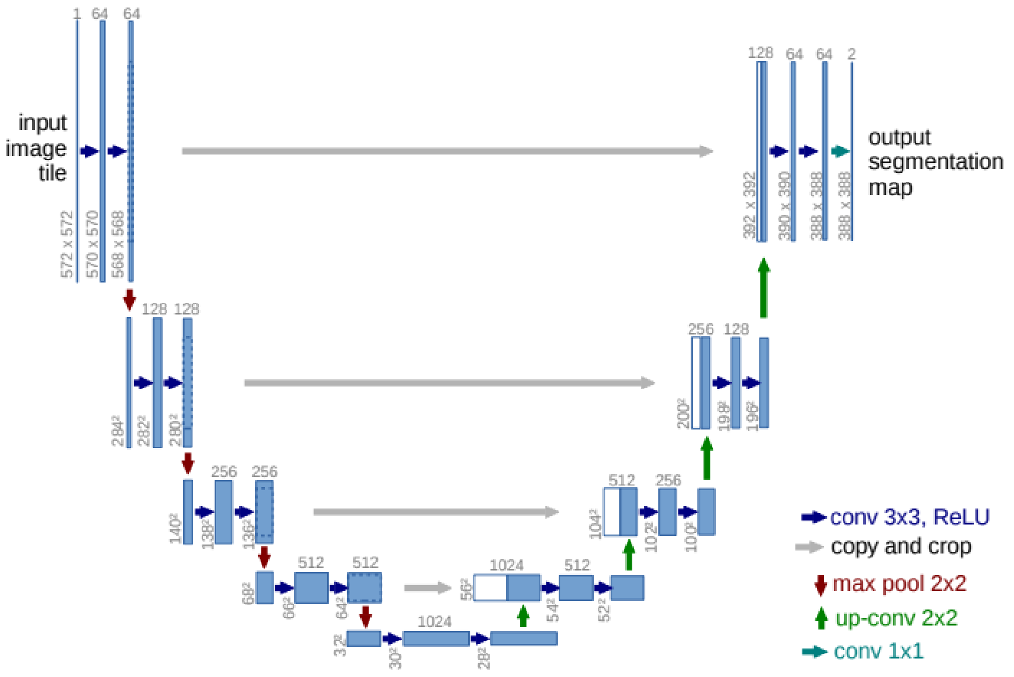
Exercise 3: Implement UNet. Copy the EncDec model and rename it to UNet. Then introduce the skip connections using torch.concat in the forward() function. In the end, you will also need to adjust the input size of each convolution in the constructor.

The files that you will use are:

- `lib/model/UNetModel.py (UNet)`: Implement UNet.

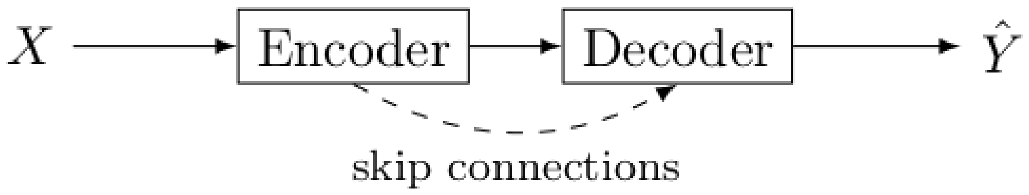
UNet

A U-Net is a neural network architecture that receives an image and outputs an image. It was originally conceived for semantic segmentation (as we will use it), but it is so successfully that it has since been used in other contexts. Given a medical image, it outputs a grayscale image representing the probability of each pixel being the region of interest.



Like the encoder-decoder network, this architecture is composed by two parts: Half of the network applies convolutions and pooling layers to successively reduce the input image (*encoder*). The other half successively increases the size of the image (*decoder*). In the end, we have an output image with the same size as the input image, but the number of channels might be different (e.g. the output might be grayscale and the input can be RGB).

Unlike the encoder-decoder network, U-Net adds skip-connections:



Notice that the activation map along the encoder decoder network are symmetric. That is, the size of the activation map in the 1st layer is equal to the size of the last layer, the size of the 2nd layer's activations is equal to the penultimate activations, etc. The authors took advantage of this fact to introduce skip-connections to further improve the performance of the networks. What these skip-connections do is to connect each layer not only to the previous layer, but also to its twin layer. This allows gradients to travel more freely (avoiding things like vanishing gradients) and also helps avoid checkboard artifacts which sometimes plagues this type of networks.

This can be done during ‘fpass‘ by concatenating (‘torch.cat‘) the encoding output to the respective decoding input. For example, ‘torch.cat([d0, e2], 1)‘.

* Ronneberger, Olaf, Philipp Fischer, & Thomas Brox. ”U-Net: Convolutional networks for biomedical image segmentation” International Conference on Medical image computing and computer-assisted intervention. Springer, Cham, 2015.

Exercise 4: Implement another version of UNet. Replace max-pooling by convolutions with stride=2, and replace upsampling by transpose-convolutions also with stride=2. .

The files that you will use are:

- `lib/model/UNetModel.py (UNet2)`: Implement UNet.

Downsampling and Upsampling with Convolutions

So far, we have been using **Max-Pooling** for the downsampling and **nearest-neighbor Upsampling** for the upsampling.

Down-sampling:

```
conv = nn.Conv2d(3, 64, 3, padding=1)
pool = nn.MaxPool2d(3, 2, padding=1)
```

Up-Sampling

```
upsample = nn.Upsample(32)
conv = nn.Conv2d(64, 64, 3, padding=1)
```

As been discussed last week, downsample can also be done with stride=2, and upsample with stride=1/2. This generally works better than using max-pooling and upsample.

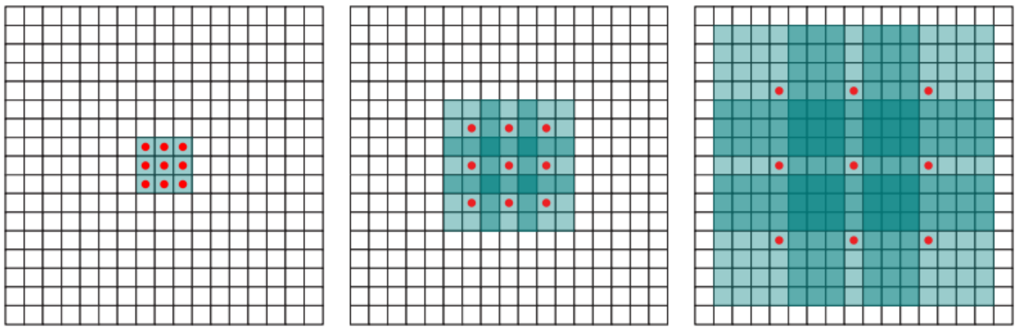
Exercise 5: Implement a neural network with dilated convolutions

The files that you will use are:

- `lib/model/DilatedNetModel.py`: Implement DilatedNet.

Dilated convolutions

We have been using encoder-decoder architectures, which are the most popular. But other architectures exist. For example, we could have used so-called **dilated convolutions** as an alternative to downsampling via pooling or striding



In the first layer, we use dilation=1, the second layer uses dilation=2, and so on, until a good part of the receptive field is reached.

* Yu, Fisher, and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions" arXiv preprint arXiv:1511.07122 (2015).

Exercise 6: Train a model to tackle multi-class segmentation

Multi-Class Segmentation

1. Use the CMP-Facade dataset that you find at

```
/dtu/datasets1/02516/CMP\_facade\_DB\_base
```

2. Load the dataset. There are 12 classes, therefore, each segmentation will have the shape $12 \times H \times W$.

3. Copy over one of the previous models and adapt for multi-class segmentation. The most important change is the to *loss function*. Feel free to use this pytorch loss which implements softmax-crossentropy.

Further thoughts

1. Especially when there is little data, **transfer-learning** can also be used for segmentation. A popular approach is to replace the encoder part of the U-Net by the convolutional part of VGG-16.

2. **Data augmentation** can also be used to improve segmentation quality. Special care must be taken because (i) some augmentation is symmetric and must be applied in pairs like rotation, and (ii) some augmentation is asymmetric like brightness.

3. We have covered here *semantic* segmentation. What could we have done differently for *instance* segmentation?