Deep Learning Lab Course
Machine Learning and Computer Vision Track
Exercise 4

Artemij Amiranashvili
(send the report to amiranas@cs.uni-freiburg.de)

Due: December 11, 2017

The goal of this exercise is to train a U-net for cell segmentation. (The lecture slides on segmentation can be found at `https://lmb.informatik.uni-freiburg.de/lectures/deep_learning/`). At the end, send us a small report (1 or 2 pages) with the below described plots and the implementation.

1 Cell segmentation data

First install h5py in order to work with the binary dataset:

```
$ pip3 install --user h5py
```

The class "Data" in the main.py file will load the training and testing data. The get_train_image_list_and_label_list function will randomly sample an 300×300 image and the according 116×116 labels out of the training data images. We will use those samples to train the U-net. Since we will use a batchsize of 1, the function will put 1 image and its labels into the according lists.

The get_test_image_list_and_label_list function will output a list with 12 new images (again with 300×300 resolution) and a list with the according labels (again with 116×116 resolution). Those images are not present in the training set. Use them for the validation accuracy of the network.

2 U-net implementation

Implement the U-net with the architecture on slide 11 of the segmentation slides with the following reductions: Decrease the input resolution to 300×300 and use only half of the number of filters for each layer. Those reductions are necessary to fit the network on the available gpu memory of the computer pool.

Do not use any padding in all layers ($\Leftrightarrow$ use 'valid' padding). If everything is implemented correctly the output resolution should be 116×116.

In tensorflow you can get the shape of each tensor using: tensor.get_shape(). Use it to crop the filters to the according dimension during the "Reusing Features" part of the network. You can use tf.concat to combine the cropped features with the new features.

Each convolution layer uses $[3 \times 3]$ filters with stride 1 and the output convolution uses a $[1 \times 1]$ filter with stride 1. Each max pooling or transposed convolution layer uses a pooling or filter size of $[2 \times 2]$ and a stride of 2. We use a ReLu nonlinearity after each convolutional layer except for output layer.

3 Training of the Autoencoder

The last layer should have the shape of $[? \times 116 \times 116 \times 2]$. The first dimension is the batchsize, followed by the output resolution. The last dimension is the number of classes that we try to segment. In this exercise only two classes are present: "cells" and "background". To train the network we treat each pixel individually, like in the auto-encoder. We apply a softmax along the last dimension for every pixel. The loss is the crossentropy loss along the last dimension. We basically solve a 2-class classification problem for each pixel. The total loss is the sum over all crossentropy losses. We divide the total loss by the total number of pixels.

Use a batch size of 1 and the Adam optimizer with the following parameters:

```
tf.train.AdamOptimizer(0.0001, 0.95, 0.99)
```

Train the U-net for 40000 iterations. Plot the according learning curves with the training and validation accuracy. Also plot several segmentation examples of the network using the validation data.

Segmentation accuracy

Most of the pixels in the images have the label "background". Therefore the pixel-average class prediction is not a good measure for the segmentation accuracy (predicting "background" everywhere would already result in a quite good accuracy). To avoid this problem use the "intersection-over-union" measure for accuracy, which is defined by:

$$\text{acc} = \frac{\text{number\_of\_correct\_cell\_pixel\_predictions}}{\text{total\_number\_of\_cell\_pixels} + \text{number\_of\_incorrect\_cell\_pixel\_predictions}}$$

4 Running experiments

4.1 Logging in to the pool computers remotely

1. Connect to the main server:
   ```
   $ ssh username@login.informatik.uni-freiburg.de
   ```
2. Do not run anything on the main server. Connect to one of the pool computers from there instead:
   ```
   $ ssh tfpoolXX
   ```
3. Replace XX with a 2 digit number of the computer you want to connect to. Every computer in the pool has a sticker on it with its name and number. Before running an experiment make sure no one else is using the selected computer by running `$ who`. Also make sure your tensorflow program is running on the gpu and not the cpu.

4.2 Continue running experiments after you log out

1. Start a screen session: (here with name ”test”)
   ```
   $ screen -S test
   ```
2. Start your python program. Thereafter detached from the screen session by entering: ctrl + a + d
3. The program will continue running in the background after you leave the screen. If you log out the program will still continue running (as long as no one turns off the pc). You can check your currently active screens on the current computer with:
   ```
   $ screen -ls
   ```
   You can reenter the ”test” screen session with:
   ```
   $ screen -r test
   ```
   In order to close a screen session run `$ exit` inside the screen session.
4. Make sure you write down on which computer you started a screen session and keep track of it to avoid ”lost” screen sessions.