Mini-projet segmentation 2024

# Preparation

In order to prepare for the project, I started with a playlist on YouTube for “Image Segmentation using U-Net” and in the following part, I will mention the most relevant ideas.

U-Net is a convolutional neural network that was developed for biomedical image segmentation. The network is based on a fully convolutional neural network whose architecture was modified and extended to work with fewer training images and to yield more precise segmentation.

The network consists of a contracting path and an expansive path, the origin of the U-shaped architecture. The contracting path is a typical convolutional network that consists of repeated application of convolutions, each followed by a rectified linear unit (ReLU) and a max pooling operation. During the contraction, the spatial information is reduced while feature information is increased. The expansive pathway combines the feature and spatial information through a sequence of up-convolutions and concatenations with high-resolution features from the contracting path.

The parameters of this NN can be modified based on our needs. Such as padding, Maxpool, Conv, upsample, Final conv. heh

The above schema is a representation of an image that goes in the CNN through a series of convolutions followed by Maxpooling operations that will appear as is in our code later on.

# Model (code in segmentationProject.ipynb)

The U-net model has a simple structure. It's a repetition of basic building blocks convolutions, ReLu, max pooling for the downsampling/encoding path, upsampling, convolutions, and ReLu for the upsampling/decoding path.

*Hence it is not complicated to implement. But how deep to go in the downsampling ? how many blocks to build?*

**With a colleague** in class after implementing the model and each of us training his model with different layers in the neural network. We found that the model going from dimension 64 by 64 to 1024 by 1024 was performing way better than the initial model going from 64 by 64 to 256 by 256.

So here is the U-net model in detail that I choose to go with:

* For the input layer, we have a shape of (128,128,1) for grayscale images as a default input but we can change it later when calling the function.
* Encoder (Contracting Path): Consists of convolutional blocks with two 3x3 Convolutional layers followed by a 2x2 MaxPooling layer. Each convolutional layer is followed by ReLU activation and uses 'same' padding. The number of filters starts from 64 and doubles at each down-sampling step. (so we get 64 🡪 128 🡪 256 🡪 512 🡪 1024 ). After each convolutional block, there's a MaxPooling layer to reduce spatial dimensions.

Dropout is applied after the last convolutional block of the encoder with a rate of 0.5.

The dropout is used to shuffle our input data to train our model.

* Bottleneck: The bottleneck consists of two 3x3 Convolutional layers with ReLU activation and 'same' padding. Dropout is applied after the bottleneck with a rate of 0.5.
* Decoder (Expansive Path) : Each block in the decoder involves up-sampling followed by two 3x3 Convolutional layers.
* Output Layer:

Consists of a 1x1 Convolutional layer followed by a sigmoid activation function.

Produces the final segmentation mask.

Output shape: (128, 128, 1)

This is a typical U-net architecture and one advantage of this model is that it can accurately segment complex images.

# Data and size:

After having designed the model, we should have a function that controls the size of the images to test our model quickly. So, I chose the (128,128) size making sure that the input images are also in grayscale so the size would be: (128,128,1). Applying that to the images during the preprocessing phase is crucial.

Obviously, we created a function to import all our data from the directory: load\_data().

# Training the model:

Now we can split the data and train our model. But before that, an important step was the normalization of the images and labels, without that (the results were very blurry and inaccurate: also comparing by testing with my colleague).

We train\_test\_split() the data then resize\_data()

We create the model’s instance and then train it with the right input data size. To train it we must test different batch\_size and epochs. Again, first we chose 10 epochs and 15 as a batch size but with multiple testing, we found out that a better option was:

batch\_size=8: The training data will be divided into batches of 8 samples. The model updates its weight after each batch.  
epochs=20: The entire training dataset will be passed through the network 20 times.

Result of the training:  
A screenshot of a computer

Description automatically generated

Results analysis:  
- The model starts with a relatively high training and validation accuracy, indicating it has a good initial performance on both datasets.   
- The final metrics indicate a well-trained model with high accuracy and low loss on both training and validation sets.

The model with all the previous parameters is performing well.

# Visualization of the results:

We can perform a prediction with our model after training it using our function: show\_predictions(model, dataset, num=1):

We will choose out of the dataset randomly some images with labels (by default 1) get the prediction of the mask by calling model.predict().

Plotting everything we get:   
A white spots on a black background

Description automatically generated

A white spots on a black background

Description automatically generated

A white spots on a black background

Description automatically generated

# Deeper comparison for results:

We add a code that is used for visualizing the predictions of our model on a test dataset, specifically focusing on comparing predicted masks with true masks.

The goal here is to visualize how well the model's predicted masks match the true masks by overlaying them on the input images. Providing a visual comparison and helping to assess the model's performance in a clearer way. For that:

we can use in the plotting alpha=0.5 that sets the transparency of the masks to make the image underneath visible.

For the input image with predicted mask: superpose using a color map (cmap='jet').

For the input image with the true mask superpose using a grayscale color map (cmap='gray').

The input image with both the predicted mask and the true mask superpose for comparison.

Results by comparing predictions:

A blurry image of a blue and white spot

Description automatically generated

A blurry image of white dots

Description automatically generated

A blurry image of white dots

Description automatically generated

# An attempt to use GPU:

I have a powerful GPU that can be used to run and train the model. So, I spent a lot of time trying to implement a test code: as you can see in the file: gpu\_test.py I had to download drivers and then download CUDA and cuDNN on my Operating System I could find the GPU working but thru python and TensorFlow it didn’t work because of many incompatibilities.

So I tried again knowing that: (<https://www.tensorflow.org/install/pip>)  
Caution: TensorFlow 2.10 was the last TensorFlow release that supported GPU on native-Windows. Starting with TensorFlow 2.11, you will need to install TensorFlow in WSL2, or install tensorflow or tensorflow-cpu and, optionally, try the TensorFlow-DirectML-Plugin

So running this command on cmd:  
python -c "import tensorflow as tf; print(tf.config.list\_physical\_devices('GPU'))"

Would return and empty array meaning that GPU is not detected by python.

Steps:

* Install python version between 2.9 – 2.11 (add it to the env variables)
* install tensorflow version < 2.11
* check tensorflow compatibility with cuda and cuDNN:  
  Version Python version Compiler Build tools cuDNN CUDA  
  tensorflow-2.10.0 3.7-3.10 GCC 9.3.1 Bazel 5.1.1 8.1 11.2
* after downloading cuda and cuDNN
* we have problems with python version

conclusion this method didn’t work.

Even on Google Collab I tried to execute the code with the full dimension but the session printed out: your session crashed after using all the available RAM.

That’s why we will stick with dimension reduction.