Lab2 report

Lucie Sénéclauze Bertolo

March 25, 2025

1 Introduction

In this report, I present my implementation steps and model result for the implementation of 2 famous deep networks : UNet and ResNet34. The implementation has been made on the oxford-IIIT-PET dataset in order to perform binary segmentation.

2 Implementation details

2.1 Code structure and overflow review

My codebase follows the suggested directory structure but I took the liberty to add "plot/" file to store any interesting plot so that someone running my code will see where my plots are coming from.

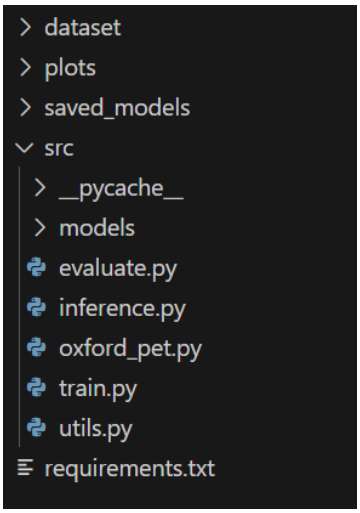


Figure 1: Directory structure

The train and inference scripts contains functions that can be called using a command line read by a parser. The evaluate script contains different visualisation functions that can be called by the user. The dataset definition is made in the oxford_pet.py script and the dataset is separated in 3 : training, validation and testing. Training and validation are used in the train.py script, respectively to train the model and to check its performances. Testing is used in evaluate and infer to see various results on the model accuracy. Everything has been implemented with Pytorch and follows a classic Pytorch pipeline.

2.2 Model architecture

I have implemented both a UNet structure and a ResNet34_unet structure

2.2.1 UNet

The UNet structure takes its name from it U shape : it is made of an encoder and a decoder. Furthermore, it has a "net-like" structure with intermediate output from the encoder being fed as intermediate input for the decoder. This serve as a "help" for the decoder to better reconstruct the data. The data fed to the network will increase its feature number while reducing its dimension when going through the encoder and then it will go back to its original dimensions in the decoder. Depending on the documentation, some merge the intermediate data from the encoder with the one from the decoder using copy and crop and some other use a concatenation. I chose to use a concatenation in order to retrieve the maximum amount of information possible.

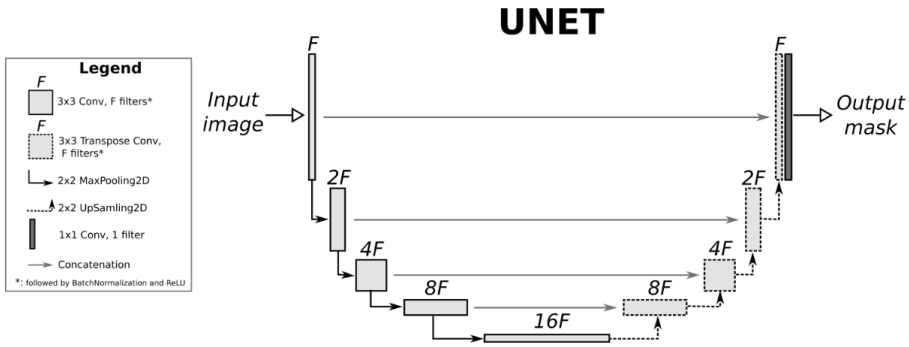


Figure 2: unet strucuture

Source: https://nchlis.github.io/2019_10_30/page.html

2.2.2 ResNet34-UNet

ResNet34 is a deep residual network that follows this structure. I implemented it by doing a merge of ResNet and UNet : resnet as the encoder and unet as the decoder.

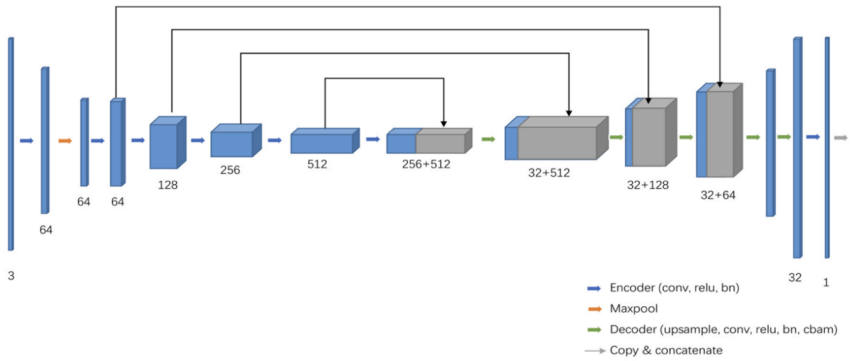


Figure 3: ResNet34_UNet structure

Source: https://www.researchgate.net/publication/359463249_Deep_learning-based_pelvic_levator_hiatus_segmentation_from_ultrasound_images

2.3 Training pipeline

I trained my model using Pytorch's Adam optimizer because it combines the good properties of Adadelta and RMSprop and generally performs great. For the loss function, I used the cross entropy loss as it is particularly well suited for binary segmentation. I could have used the dice loss but

using the dice score was already imposed and I found interesting to be able to monitor the evolution of my model's performance over 2 different metrics (ie the loss and the dice score).

2.4 Model result

I have stored my weights in the saved_models directory. However, note that if you rerun the program using the exectution steps in section 5, it will overwrite my weights which may change the final displayed performance.
I have also accidently overwritten my final weights 10 minutes before the deadline when I wanted to check if everything was working alright. Hence, the displayed dice score may not be the same as the one I claim here to have because I lost my better performing model. However if you re run the training process it, you should get back my "good" dice score.

3 Data preprocessing

I did not spend a lot of time of the project working on the dataset which, as I will explain in section 4.1.3 may have been a mistake. However, I did do some modifications do the code I was given.
First, I only used the simplified oxford pet class because the unsimplified one was simply too big for me to compute and the simplified had all the images be in the same dimensions. I diminished the size of the images from the suggested 256x256 to 128x128 in order to simplify the computation. I also diminished the size of the test dataset as the computation were unnecessarily long.

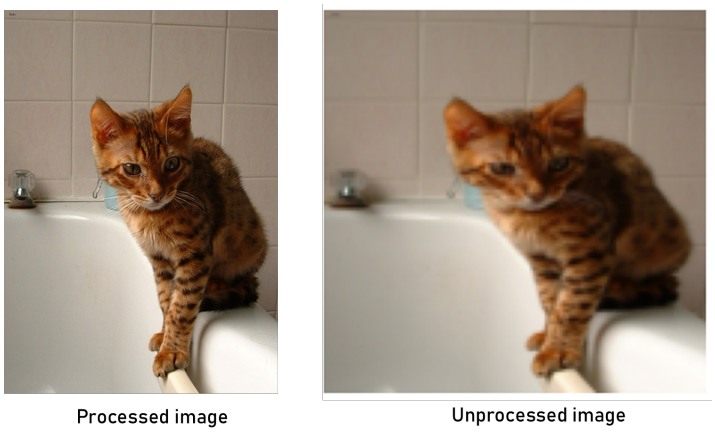


Figure 4: Image preprocessing

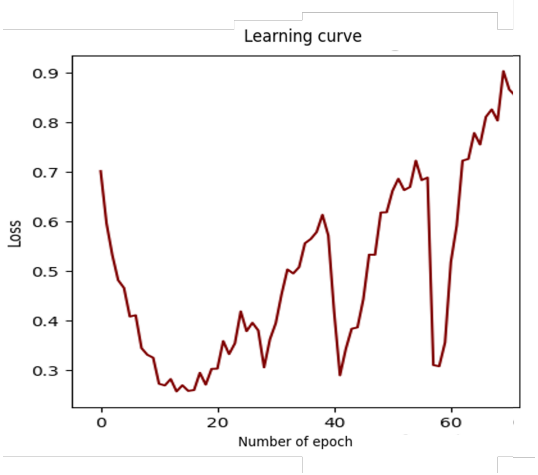
4 Experimentation results

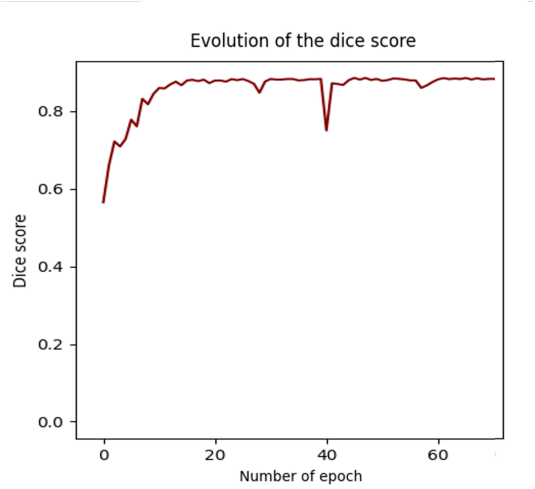
4.1 UNet

4.1.1 Choosing the hyperparameters

The first step of the training was to find the best hyperparameters. To choose the number of batch, I simply selected the highest one that was available taking into account my available memory space. For the number of epoch, I trained my model on a very high number of epoch and visualised the evolution of the dice score and loss over the epoch to choose the saved model with the best performance.

As we can see on the figure, the dice score doesn't change much after around 20 epoch while the generalisation loss increases. As such, I chose 20 for the number of epoch as a tradeoff between the generalisation loss and the dice score. Finally for the learning rate, I didn't really have time to explore it's impact and I chose 0.001 as it is a classical value





4.1.2 The results

Below you can see my prediction results for a randomly chosen image. I chose to display the evolution of the model's prediction across the epoch to see that it is indeed learning.

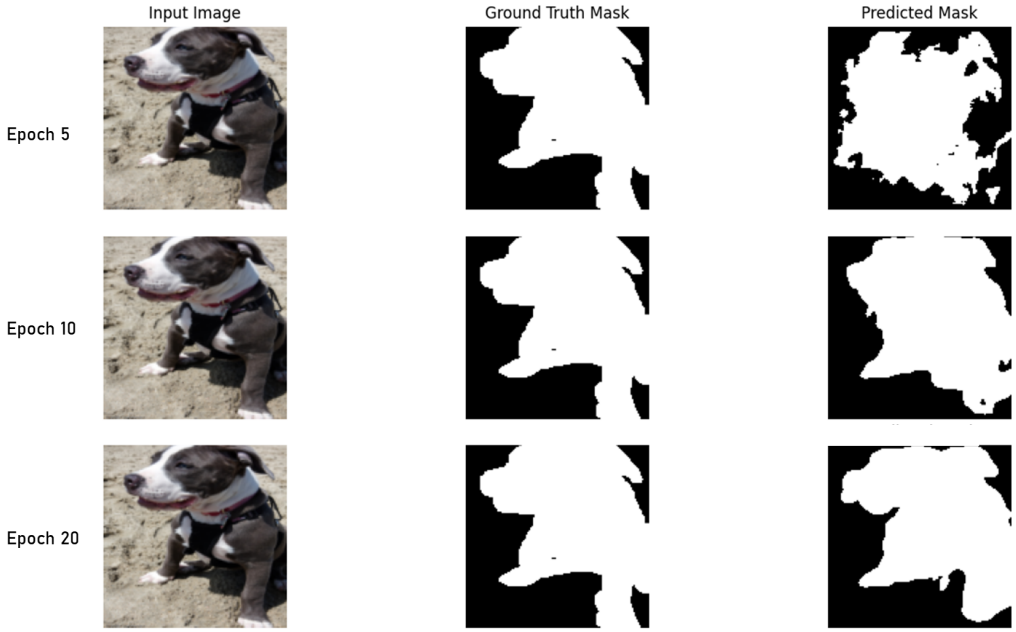


Figure 5: Evolution of the model

4.1.3 Possibleways to improve

I sadly did not reach the expected value of 0.9 for the dice score, and I could only get up to around 0.88. Here is a list of the ameliorations I would have put in my model in order to make my overall performance better, had I had the time :

- As I mentionned in section 4.1.1, I didn't really explore the impact that the choice of the learning rate could have. I could have trained my model for different learning rates, using otherwise the exact same parameters, and chosen the one that yield me the best performance.
- I could also have played a bit pore with other hyperparameters such as the choice of the loss or the choice of the optimizer.

- I think that working on the data would have been especially helpful here. The first way could have been to not decrease the image size as much as I did so that my model would learn to be more precise. While it wouldn't necessary have a positive effect on the dice score, I think it would be useful if my model was to have an application in real life.
- Another thing I could have done with the data would have been data augmentation. Since the images we have to treat here are pictures of pets, it wouldn't be relevant I think to do "extreme" transformation of the data like turning it upside down but I think a mirroring the images or cropping them could have been interesting. Doing data augmentation is useful because it easily double the size of the dataset which is rather small (only 7398 images). However the drawback is that it would also double the computing time and I didn't have the resources to train a dataset that big.

Figure 6: Possible ways to do data augmentation

- Finally, I could have tried to alter the model, maybe adding more layers to optain a more complicated representation in the latent space. That could also have affected my model's performances.

4.2 ResNet34_UNet

For the ResNet34, I encountered more problems and I couldn't manage to get it to work.
As for the hyperparameters choice and possible ways to improve the performances, they are the same as for the UNet model.

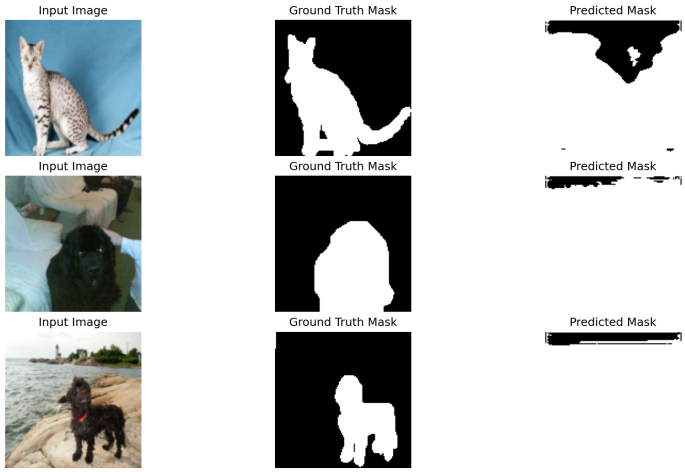


Figure 7: Bad ResNet34_UNet results

5 Execution steps

5.1 Training step

Here is the command line that I ran for both models :
python src/train.py –data_path ”dataset/oxford-iiit-pet/” –epochs 20 –batch_size 32 –learning-rate 0.001 –model ”resnet34_unet”
python src/train.py –data_path ”dataset/oxford-iiit-pet/” –epochs 20 –batch_size 32 –learning-rate 0.001 –model ”unet”

However, the batch size can be increased if you wish to run it on your device which may be more powerfull than mine.

5.2 Inference

For the inference, the command lines are python src/inference.py –model ”resnet34_unet” –data_path ”dataset/oxford-iiit-pet/” –batch_size 32
python src/inference.py –model ”unet” –data_path ”dataset/oxford-iiit-pet/” –batch_size 32

5.3 Evaluate

For the evaluate part, I did not use a parser and instead when running the evaluate.py script you will be met with the possibillity to try different result evaluation methods.

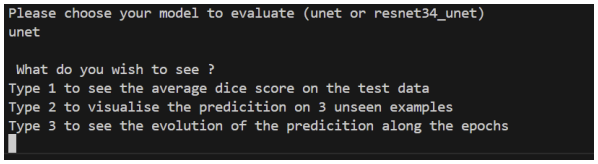


Figure 8: The evaluate interface