Case study: Brain tumor detection

Olivier Canévet

December 22, 2023

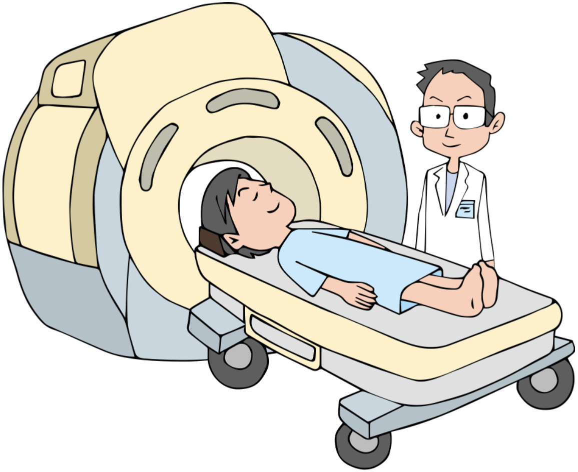
Case study: brain tumor detection

This part is a use case on brain tumor detection in magnetic resonance imaging (MRI) images.

The objective is to address all the steps which you can encounter in a machine learning project, from data collection, to evaluation, including model design and training.

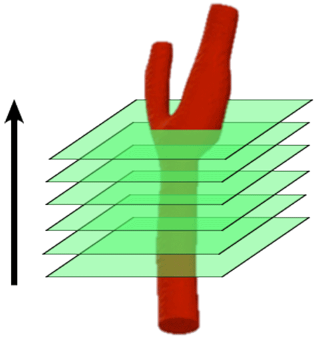
Magnetic Resonance Imaging (MRI)

MRI is used in radiology for medical diagnosis and follow-up on diseases. The patient lies on a motorised bed which is moved inside the scanner.



Scanning the body

The MRI scans the body by slices:



The resulting ”volume” consists of a set of 2d images.

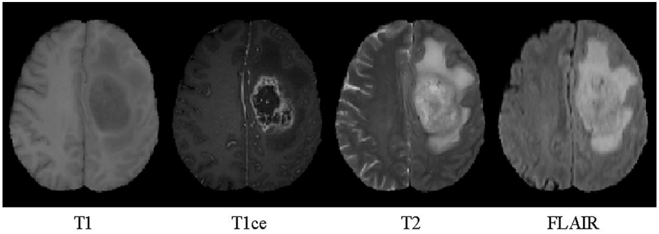
MRI images

An MRI image usually consists of 4 channels (or modalities):

- T1-weighted MRI (T1),
- T1-weighted MRI with gadolinium contrast enhancement (T1-Gd), and
- T2-weighted MRI (T2),
- Fluid Attenuated Inversion Recovery (FLAIR).

As a comparison, a usually RGB image consists of 3 modalities: red, green, and blue.

Here is an example of the 4 modalities:



The project

You work on a project to detect brain tumors in MRI images. You have a background in machine learning and computer vision and your task will be to apply machine learning techniques.

The research consortium consists of:

- your team in an academic institution or a company,
- a hospital. They will provide you with some data, but not now, as it takes time to collect and annotate the data.

Question

How would you start the project?

Literature

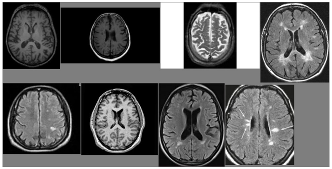
- Meet with the doctors in the hospital to understand what they want, what data they will provide, what performance they want to achieve, what metrics they want to use, etc.
- Read papers on the subject: conferences, arXiv, books, etc.
- Look for existing datasets, pre-trained models, devices, etc.
- Look for available code and softwares: GitHub, PapersWithCode, etc.
- Also important: check the license whether it is available for commercial use, academic research, etc.

Datasets

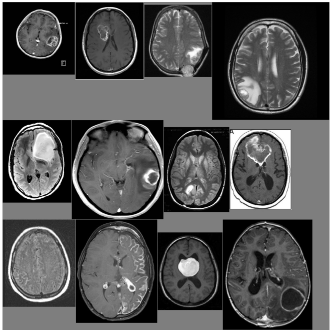
The hospital will provide you with some data. But not yet.

In the meantime, you look for existing datasets: as you did some literature search, you decide to use datasets used in the publications. You first come accross a small brain MRI image dataset which contains 370 images (185 with tumor, and 185 without tumor).

Without a tumor



With a tumor



(image displayed in gray levels, but there are 4 channels)

Dataset split

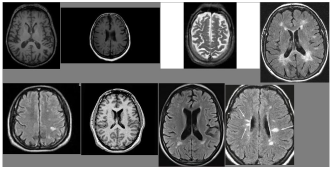
How would you split the dataset?

We can keep 20 to 40 images of both class for testing, so 300 training images and 70 test images.

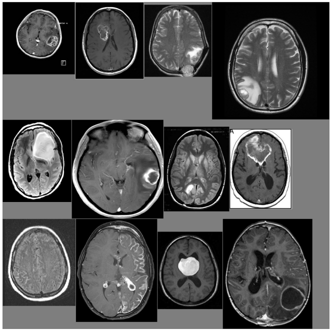
Question

What potential issues can you anticipate with this dataset?

Without a tumor



With a tumor

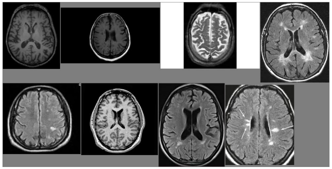


Pre-processing

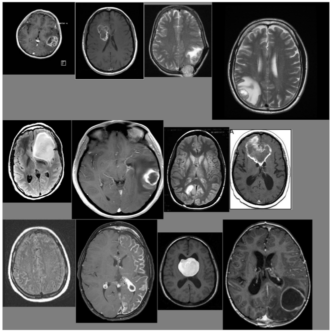
- The images are not of the same size,
- The background color is not the same,
- The brain is not located at the same position,
- The brain is sometimes rotated,
- etc.

This will require some pre-processing,

Without a tumor



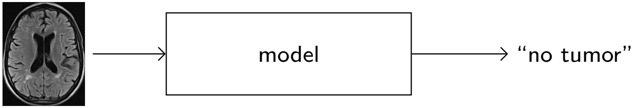
With a tumor



Classification task

You decide to start with the simple task of classification: “tumor” vs. “no tumor” on some given slice images (i.e. not the full volume as input, only the images)

You are going to train a neural network which takes as input one image (a slice of the full MRI) and will output the probability of a tumor being present.



We assume that you have pre-processed the data: all the images are of the same size, and the brain is properly centered in the images.

Minibatch size

What would be the PyTorch tensor size for a minibatch of 10 images of MRI images of height 192 and width 128?

Since an MRI image consists of 4 channels, with the following tensor convention $N \times C \times H \times W$ for a minibatch of N images of height H and width W, the PyTorch tensor size for a minibatch of 10 images of MRI images of height 192 and width 128 would be of size $10 \times 4 \times 192 \times 128$.

- T1-weighted MRI (T1),
- T1-weighted MRI with gadolinium contrast enhancement (T1-Gd), and
- T2-weighted MRI (T2),
- Fluid Attenuated Inversion Recovery (FLAIR).

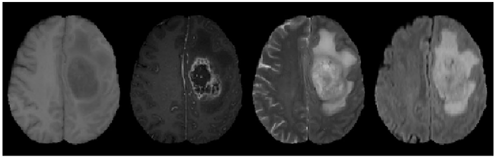


Image size

You found a Python library which loads the MRI images as NumPy arrays. You load one image, and print its shape:

```
>>> image.shape
(192, 128, 4)
>>> type(image)
<class 'numpy.ndarray'>
```

What issue can you notice with the size $(192, 128, 4)$ when training with PyTorch?

Note that the channel dimension is at the end, but PyTorch requires the channel dimension to be before the height and the width.

How would you use PyTorch function to make it a tensor of size $4 \times 192 \times 128$ (use `permute` and `from_numpy`).

```
x = torch.from_numpy(image).permute(2, 0, 1)
```

And then

```
>>> x.shape
torch.Size([4, 192, 128])
>>> type(x)
<class 'torch.Tensor'>
```

Adapting a network for RGB images

You pre-process the data so that all the images are of size $4 \times 224 \times 224$

You start by training a "usual" (i.e. not custom) model such as AlexNet or ResNet available in PyTorch.

The default architecture of AlexNet (or ResNet):

- takes as input an RGB image, and
- outputs a vector of size 1000.

What should you change in the first and last layers of AlexNet or ResNet to fit your needs?

Model architecture for classification

For the input layer,

- Turn the first `nn.Conv2d(3,64)` into `nn.Conv2d(4,64)` to input the 4-channel MRI images.

For the output, you have 2 choices:

- either you use an output layer with 2 outputs and treat the problem as a 2-class multiclass classification problem. The output layer would be `nn.Linear(..., 2)` and you can use the `nn.CrossEntropyLoss`.
- or you treat the problem a binary problem and the model output only one value `nn.Linear(..., 1)` followed by a `nn.Sigmoid` and then the loss is `nn.BCELoss()`.

Pre-training

Can you / should you use a pre-trained model?

We cannot use a RGB model “as is” (then input and output layers need to be updated), but why not trying to load the pre-trained weights on ImageNet for the layers.

In any case, one can try to train from scratch, or to fine-tune an existing model (first and last layer modified), and see what is best.

Accuracy

You have successfully trained the model on the training data (at least, the code does not crash...), and you compute the accuracy on the test data (the 70 images, 35 of each class). You get 52.8% accuracy on the test set (37/70 images as correctly classified).

Are you happy?

No, because since the classes are balanced, the random guess would be 50% (like flipping a coin).

Accuracy on another test set

You come across a new dataset for the same task: there are 90 images of class "no tumor" and 10 images for class "tumor". Your network has an accuracy of 90%: in this case, you compute the accuracy with the following formula:

$$\text{accuracy} = \frac{\text{Nb. correctly classified}}{\text{Nb. of samples}}$$

Are you happy?

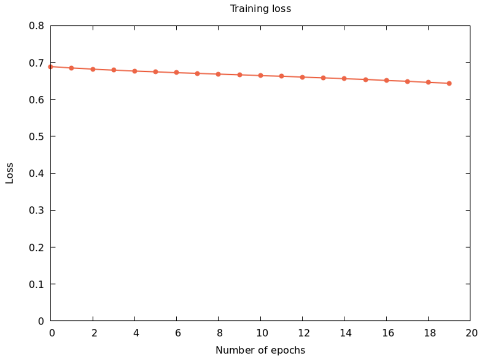
No, because a constant classifier outputting 0 would reach the same accuracy.

You need to compute the balance accuracy:

$$\frac{1}{C}\sum_{y=1}^{C}\hat{P}(f(X) = Y \mid Y = y) = \frac{1}{C}\sum_{c}\frac{\text{Nb. correctly classified of class } c}{\text{Nb. of samples of class } c}$$

Training loss

Since the accuracy is not satisfactory, you decide to plot the training loss to see if the training actually goes well. Here is what you get:



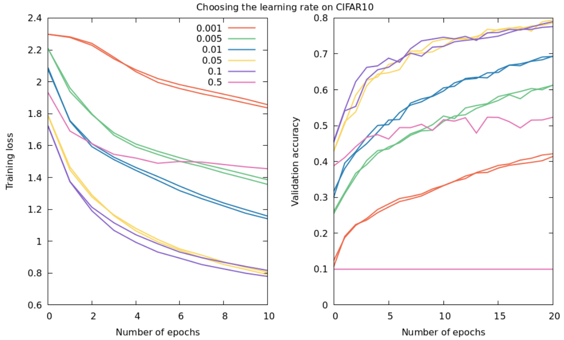
What can you say and what could you do to improve the situation?

- The network is not training (flat loss)
- Starting value at log(2) because there are 2 classes (good point)
- Increase the learning rate

Choosing the initial learning rate

How would you choose the learning rate?

One can test several learning rates for a few epochs and choose the one which reduces the loss more at the beginning. For instance on CIFAR10 (10 classes):

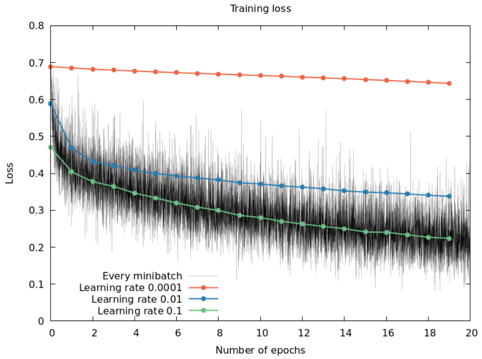


Here, we could start with an initial learning rate of 0.1. Then when you train with more epochs, the learning rate can be reduced when the loss reaches a plateau.

Note that $\log(10) \simeq 2.3$ and that random guess here is 10%.

Training

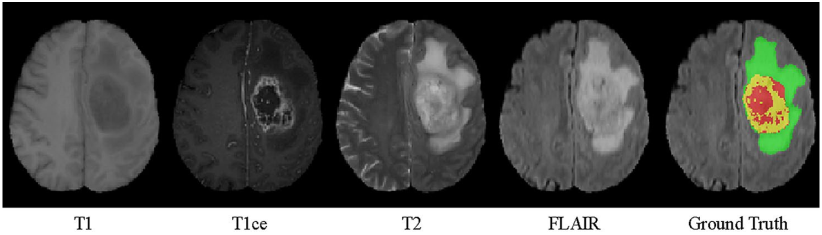
You change the learning rate and you get the following



Segmentation task

After the classification task, you decide to switch to the segmentation task to precisely locate the tumor, rather than simply classify as "tumor" vs. non tumor.

You find a dataset named BraTS[1] (for brain tumor segmentation dataset).



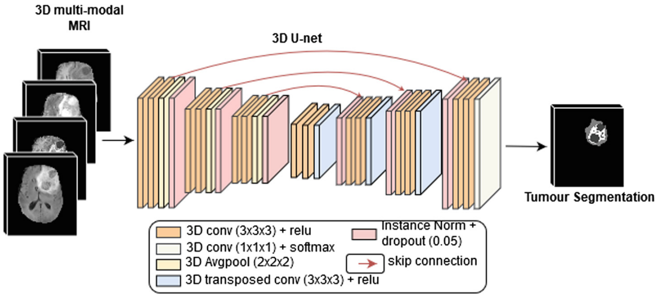
The meaning of the colors for the ground truth is:

- red: the necrotic and non-enhancing tumor core
- yellow: the enhancing tumor core
- green: peritumoral edema

[1] `https://www.med.upenn.edu/cbica/brats2020/data.html`

Architectures for segmentation

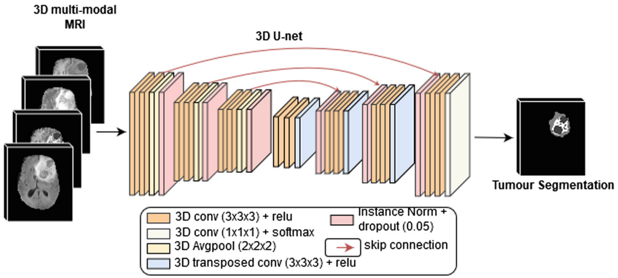
In your literature search, you come across the following architectures for brain tumor segmentation. The first paper[2] you find implement the following architecture:



What is the input of the model? What architecture(s) / block(s) / element(s) do you recognize?

[2] https://link.springer.com/chapter/10.1007/978-3-030-11726-9_23

Architectures for segmentation

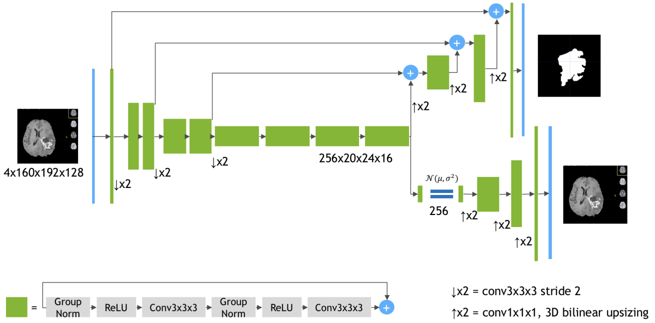


This U-net takes as input a 3d image: the multiple 4d-slices of the MRI

- 3d convolutions and ReLU layers,
- pooling layers to reduce the size of the activation maps,
- 3d transposed convolutions to increase the size of the activation maps,
- skip connections (making a U-net shape)
- the output is a volume, probably of the same height and width as the input image, but with 3 channels (one per class), maybe a background class as well counting as “none of the other classes”.

Architectures for segmentation

Another paper[3] proposes this architecture:

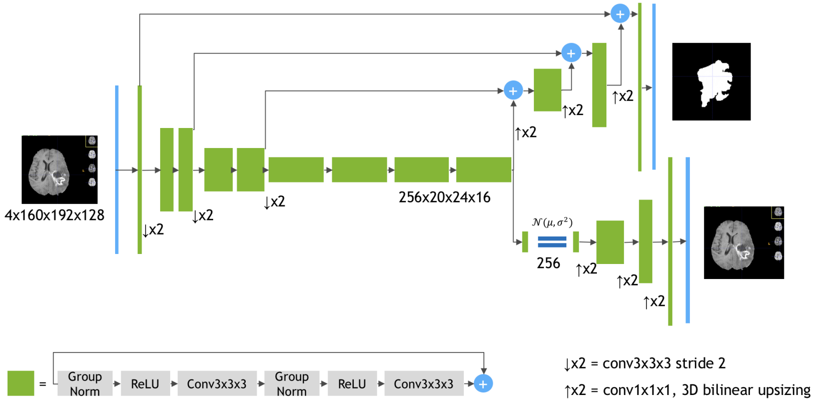


4x160x192x128
↓x2
↓x2
↓x2
256x20x24x16
↑x2
↑x2
↑x2
$\mathcal{N}(\mu,\sigma^2)$
256
↑x2
↑x2
↑x2
= Group Norm → ReLU → Conv3x3x3 → Group Norm → ReLU → Conv3x3x3 → +
↓x2 = conv3x3x3 stride 2
↑x2 = conv1x1x1, 3D bilinear upsizing

What architecture(s) / block(s) / element(s) do you recognize? What causes the size of the activation maps to be reduced? What does the $4 \times 160 \times 192 \times 128$ mean?

[3] https://arxiv.org/pdf/1810.11654.pdf

Architectures for segmentation


4x160x192x128
↓x2
↓x2
↓x2
256x20x24x16
↑x2
↑x2
↑x2
$\mathcal{N}(\mu,\sigma^2)$
256
↑x2
↑x2
↑x2
= Group Norm → ReLU → Conv3x3x3 → Group Norm → ReLU → Conv3x3x3 → +
↓x2 = conv3x3x3 stride 2
↑x2 = conv1x1x1, 3D bilinear upsizing

- The network takes as input 160 frames of size $4 \times 192 \times 128$. 4 is for the T1, T2, T1ce, and FLAIR channels. 160 is the number of slices in the MRI image. The input is a volume.
- The downsampling is not done with pooling, but with a convolutional layer with a stride of 2,
- The top part is a U-Net for segmentation,
- The bottom part is a variational autoencoder forcing the signal to follow a $\mathcal{N}(\mu,\sigma)$.

Delivering the software to the hospital

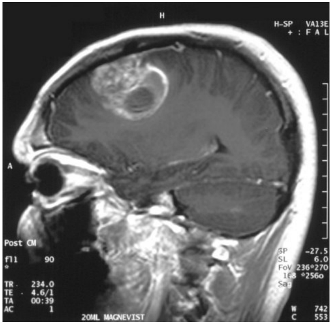
Once you get good results on your segmentation dataset, you send your model to the hospital so that they can apply it on their own image. They are surprised that it does not work as well as what you have told them.

What do you do?

- Check they are using the code properly
- Debug with them
- Check the data
- Check the data (again)
- etc.

Looking at the data

You ask them to send you an image on which the model is failing. Here is the image:



What is wrong?

The input is not the same at all (not the same view point). Your model was not trained on such data.

They now apply the model on the correct view. They still observe bad performance. What could be wrong now?

Distribution mismatch

The MRI machine may not produce the same images (different lightning, difference image mean, different quality, etc.)

You may need to collect and annotate data from the machine of the hospital and fine-tune the model on their data, or enrich your current training set with their data.

> *"It turns out," Ng said, "that when we collect data from Stanford Hospital, then we train and test on data from the same hospital, indeed, we can publish papers showing [the algorithms] are comparable to human radiologists in spotting certain conditions."*
>
> *But, he said, "It turns out [that when] you take that same model, that same AI system, to an older hospital down the street, with an older machine, and the technician uses a slightly different imaging protocol, that data drifts to cause the performance of AI system to degrade significantly. In contrast, any human radiologist can walk down the street to the older hospital and do just fine."*

https://spectrum.ieee.org/andrew-ng-xrays-the-ai-hype