Julie DANIEL
Chloé GATTINO

Case Study: Medical Image Segmentation Using U-Net

Introduction :

Medical image segmentation plays a crucial role in modern healthcare, enabling precise identification and analysis from complex medical scans such as MRI images. In this case study, we explore the application of the U-Net architecture, a deep learning model specifically designed for biomedical image segmentation tasks.

We have MRI brain scans classified into four categories: non-tumorous, gliomas, meningiomas, and pituitary adenomas. Our goal is to develop a model capable of predicting the presence of a tumor from an MRI image and accurately identifying its type.

You can see and test our model on our github repository :

https://github.com/ZulieD/Analyse_image/tree/main

Implementation Steps :

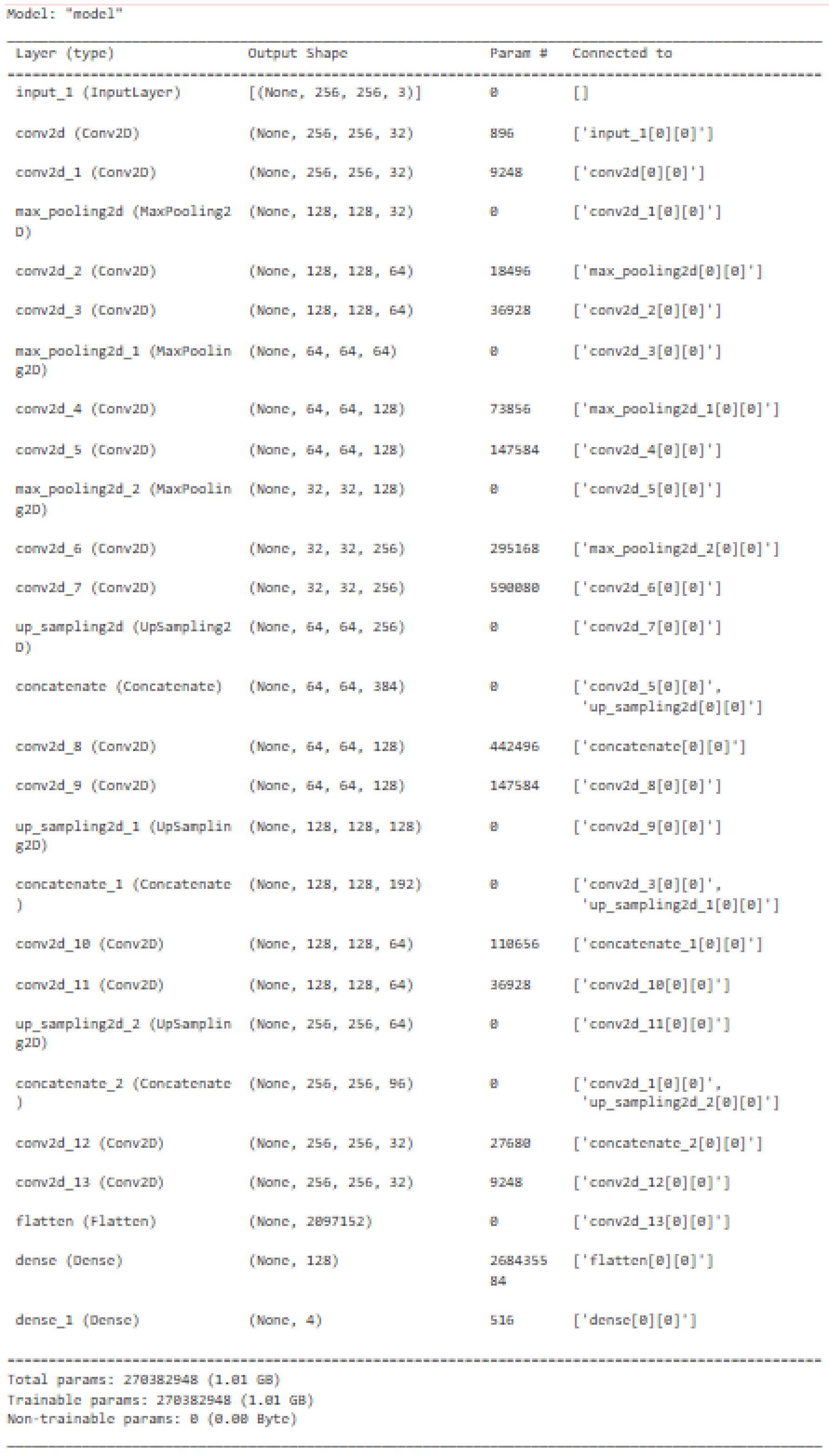
1. Data Preparation:

We began by downloading the MRI data and using the flow_from_directory function. This function enabled us to label the images according to their respective subfolders and divide the dataset into training and validation sets.

2. Model Definition:

We implemented our U-Net model using TensorFlow. The model begins with convolutional layers in the encoding phase to progressively extract features from input images. Convolutional layers with ReLU activation are used to detect significant patterns in the image, followed by max pooling layers that reduce spatial dimensions while preserving essential information. This process is repeated across multiple blocks, each increasing the number of filters to capture increasingly complex features in our MRI images. Once encoding is complete, decoding begins with transposed convolution operations to reverse the max pooling process and gradually reconstruct the images. This process is repeated in each decoder block, gradually reducing the number of filters. Finally, the output layer employs a softmax-activated convolution to generate a vector of real scores and convert them into probabilities. Softmax allows the direct comparison of probabilities among different classes. The class with the highest probability is considered the model's prediction.

To properly evaluate this model, we defined the "CategoricalCrossentropy" loss function used in machine learning and neural networks, particularly for classification tasks with multiple classes. We then used the "accuracy" metric to assess the model's performance by comparing predictions against ground truth labels.

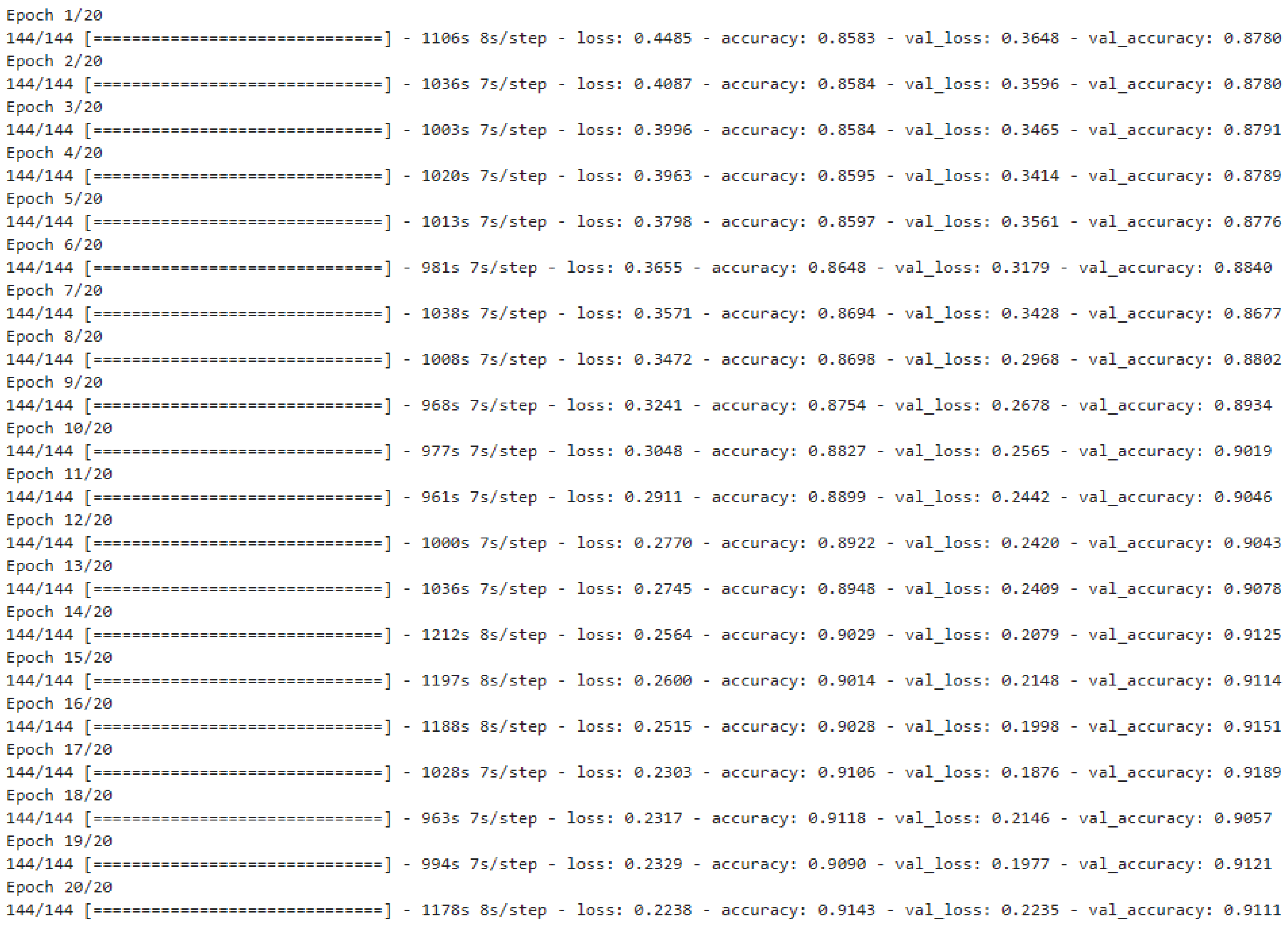


```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]        0         []

 conv2d (Conv2D)             (None, 256, 256, 32)         896       ['input_1[0][0]']

 conv2d_1 (Conv2D)           (None, 256, 256, 32)         9248      ['conv2d[0][0]']

 max_pooling2d (MaxPooling2  (None, 128, 128, 32)         0         ['conv2d_1[0][0]']
 D)

 conv2d_2 (Conv2D)           (None, 128, 128, 64)         18496     ['max_pooling2d[0][0]']

 conv2d_3 (Conv2D)           (None, 128, 128, 64)         36928     ['conv2d_2[0][0]']

 max_pooling2d_1 (MaxPoolin  (None, 64, 64, 64)           0         ['conv2d_3[0][0]']
 g2D)

 conv2d_4 (Conv2D)           (None, 64, 64, 128)          73856     ['max_pooling2d_1[0][0]']

 conv2d_5 (Conv2D)           (None, 64, 64, 128)          147584    ['conv2d_4[0][0]']

 max_pooling2d_2 (MaxPoolin  (None, 32, 32, 128)          0         ['conv2d_5[0][0]']
 g2D)

 conv2d_6 (Conv2D)           (None, 32, 32, 256)          295168    ['max_pooling2d_2[0][0]']

 conv2d_7 (Conv2D)           (None, 32, 32, 256)          590080    ['conv2d_6[0][0]']

 up_sampling2d (UpSampling2  (None, 64, 64, 256)          0         ['conv2d_7[0][0]']
 D)

 concatenate (Concatenate)   (None, 64, 64, 384)          0         ['conv2d_5[0][0]',
                                                                     'up_sampling2d[0][0]']

 conv2d_8 (Conv2D)           (None, 64, 64, 128)          442496    ['concatenate[0][0]']

 conv2d_9 (Conv2D)           (None, 64, 64, 128)          147584    ['conv2d_8[0][0]']

 up_sampling2d_1 (UpSamplin  (None, 128, 128, 128)        0         ['conv2d_9[0][0]']
 g2D)

 concatenate_1 (Concatenate  (None, 128, 128, 192)        0         ['conv2d_3[0][0]',
 )                                                                   'up_sampling2d_1[0][0]']

 conv2d_10 (Conv2D)          (None, 128, 128, 64)         110656    ['concatenate_1[0][0]']

 conv2d_11 (Conv2D)          (None, 128, 128, 64)         36928     ['conv2d_10[0][0]']

 up_sampling2d_2 (UpSamplin  (None, 256, 256, 64)         0         ['conv2d_11[0][0]']
 g2D)

 concatenate_2 (Concatenate  (None, 256, 256, 96)         0         ['conv2d_1[0][0]',
 )                                                                   'up_sampling2d_2[0][0]']

 conv2d_12 (Conv2D)          (None, 256, 256, 32)         27680     ['concatenate_2[0][0]']

 conv2d_13 (Conv2D)          (None, 256, 256, 32)         9248      ['conv2d_12[0][0]']

 flatten (Flatten)           (None, 2097152)              0         ['conv2d_13[0][0]']

 dense (Dense)               (None, 128)                  2684355   ['flatten[0][0]']
                                                          84

 dense_1 (Dense)             (None, 4)                    516       ['dense[0][0]']

==================================================================================================
Total params: 270382948 (1.01 GB)
Trainable params: 270382948 (1.01 GB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

3. Training:

We trained the U-net model on the previously prepared training data (X_train and y_train) using the fit method, then we provided the validation data (X_val and y_val) to evaluate the performance of the model after each epoch of training. We have the number of epochs (epochs=20) to specify how many complete cycles the dataset will be trained for. Then we have the batch size which is 16 (batch_size=16), which means that the model will update its weights after processing 16 training samples.

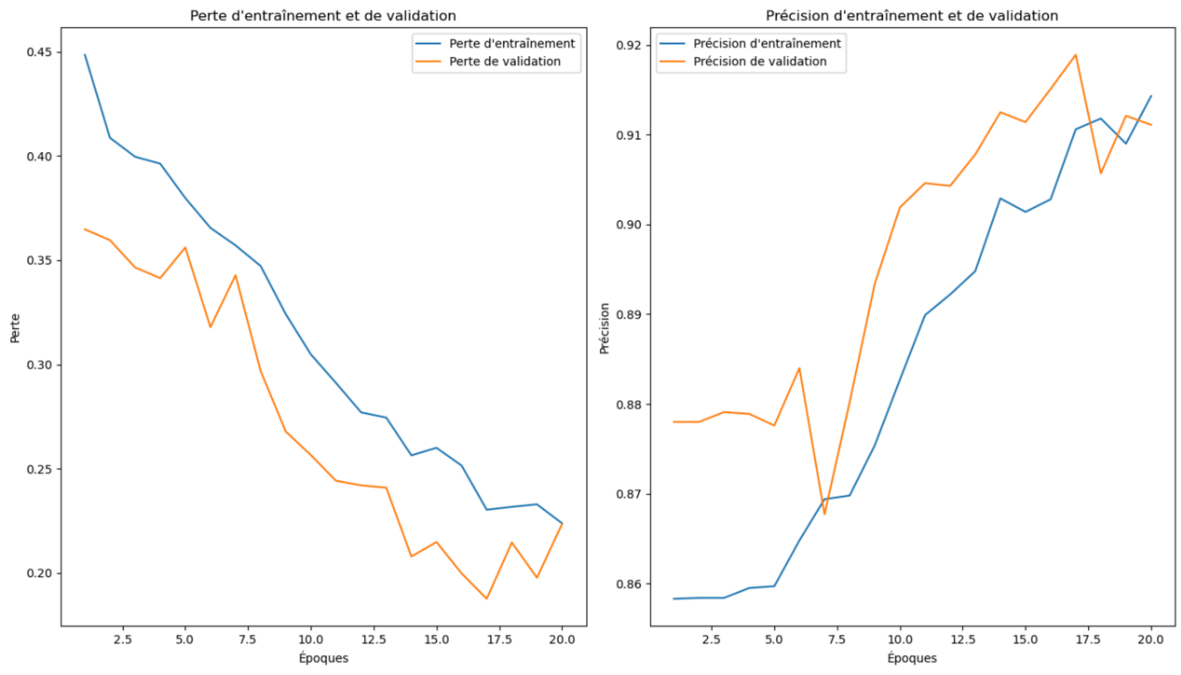
We have obtained the following results :



```
Epoch 1/20
144/144 [==============================] - 1106s 8s/step - loss: 0.4485 - accuracy: 0.8583 - val_loss: 0.3648 - val_accuracy: 0.8780
Epoch 2/20
144/144 [==============================] - 1036s 7s/step - loss: 0.4087 - accuracy: 0.8584 - val_loss: 0.3596 - val_accuracy: 0.8780
Epoch 3/20
144/144 [==============================] - 1003s 7s/step - loss: 0.3996 - accuracy: 0.8584 - val_loss: 0.3465 - val_accuracy: 0.8791
Epoch 4/20
144/144 [==============================] - 1020s 7s/step - loss: 0.3963 - accuracy: 0.8595 - val_loss: 0.3414 - val_accuracy: 0.8789
Epoch 5/20
144/144 [==============================] - 1013s 7s/step - loss: 0.3798 - accuracy: 0.8597 - val_loss: 0.3561 - val_accuracy: 0.8776
Epoch 6/20
144/144 [==============================] - 981s 7s/step - loss: 0.3655 - accuracy: 0.8648 - val_loss: 0.3179 - val_accuracy: 0.8840
Epoch 7/20
144/144 [==============================] - 1038s 7s/step - loss: 0.3571 - accuracy: 0.8694 - val_loss: 0.3428 - val_accuracy: 0.8677
Epoch 8/20
144/144 [==============================] - 1008s 7s/step - loss: 0.3472 - accuracy: 0.8698 - val_loss: 0.2968 - val_accuracy: 0.8802
Epoch 9/20
144/144 [==============================] - 968s 7s/step - loss: 0.3241 - accuracy: 0.8754 - val_loss: 0.2678 - val_accuracy: 0.8934
Epoch 10/20
144/144 [==============================] - 977s 7s/step - loss: 0.3048 - accuracy: 0.8827 - val_loss: 0.2565 - val_accuracy: 0.9019
Epoch 11/20
144/144 [==============================] - 961s 7s/step - loss: 0.2911 - accuracy: 0.8899 - val_loss: 0.2442 - val_accuracy: 0.9046
Epoch 12/20
144/144 [==============================] - 1000s 7s/step - loss: 0.2770 - accuracy: 0.8922 - val_loss: 0.2420 - val_accuracy: 0.9043
Epoch 13/20
144/144 [==============================] - 1036s 7s/step - loss: 0.2745 - accuracy: 0.8948 - val_loss: 0.2409 - val_accuracy: 0.9078
Epoch 14/20
144/144 [==============================] - 1212s 8s/step - loss: 0.2564 - accuracy: 0.9029 - val_loss: 0.2079 - val_accuracy: 0.9125
Epoch 15/20
144/144 [==============================] - 1197s 8s/step - loss: 0.2600 - accuracy: 0.9014 - val_loss: 0.2148 - val_accuracy: 0.9114
Epoch 16/20
144/144 [==============================] - 1188s 8s/step - loss: 0.2515 - accuracy: 0.9028 - val_loss: 0.1998 - val_accuracy: 0.9151
Epoch 17/20
144/144 [==============================] - 1028s 7s/step - loss: 0.2303 - accuracy: 0.9106 - val_loss: 0.1876 - val_accuracy: 0.9189
Epoch 18/20
144/144 [==============================] - 963s 7s/step - loss: 0.2317 - accuracy: 0.9118 - val_loss: 0.2146 - val_accuracy: 0.9057
Epoch 19/20
144/144 [==============================] - 994s 7s/step - loss: 0.2329 - accuracy: 0.9090 - val_loss: 0.1977 - val_accuracy: 0.9121
Epoch 20/20
144/144 [==============================] - 1178s 8s/step - loss: 0.2238 - accuracy: 0.9143 - val_loss: 0.2235 - val_accuracy: 0.9111
```

4. Evaluation:

To evaluate our model, we generated a graph that illustrates the training and validation loss as well as accuracy across 20 epochs. The left side of the graph visualizes the loss, while the right side depicts the accuracy.



We observe a consistent decrease in loss and an increase in accuracy across epochs. This trend indicates that our model performs effectively, achieving an accuracy of 91% and a loss of 0.2. These metrics are indicative of the model's ability to correctly classify the data and minimize errors, showcasing its robust performance over the training process.

5. Deployment and Testing:

To finish, we test our model on the test data (X_test and y_test) using the Keras evaluate method. This method returns the accuracy on the test data. This phase is crucial for validating the robustness and generalization of the model on data it hasn't seen during training.

We obtained an accuracy on the test data of 75%.

Conclusion

To conclude, the objective of this project was to develop a model for brain tumor segmentation from MRI images using the U-Net architecture. This automation aims to assist radiologists in diagnosing and monitoring brain tumors more effectively. The final accuracy achieved is 75%, with a progressive decrease in training and validation loss, as illustrated in the graphs. Although the 75% accuracy is promising, it indicates that there is still room for improvement. The accuracy could be enhanced by developing a more complex U-Net model with additional layers, or by using a Res-Net model, which could improve performance. Optimizing hyperparameters such as the learning rate and batch size could also yield better results.