Image of Brain Stroke Lesion segmented with X-Net and Quaternion Neural Network

Master's degree in Artificial Intelligence and Robotics

Course: Neural Network
Professor: Aurelio Uncini

Authors:
Denise Landini – 1938388
Alessandro Lambertini – 1938390

La Sapienza University - Roma
Academic year: 2020-2021

Summary

1 Introduction

The main objective of our study is to better segment the images related to the broke stroke lesion segmentation. This is very interesting, especially in the medical filed. In fact, this project can help the doctors to automatically recognize which part of the brain present a stroke lesion.

We propose a new method based on X-Net architecture and quaternions that are both two new approaches, but the more recent is the quaternions one.

In the first chapter, we are interest in analysing the Depthwise Separable Convolutions because our architecture is based on Depthwise Separable Convolutions and long-range dependencies. So, we discuss the differences between standard convolution and DSC, but we also talk about advantages and disadvantages that we have by using DSC.

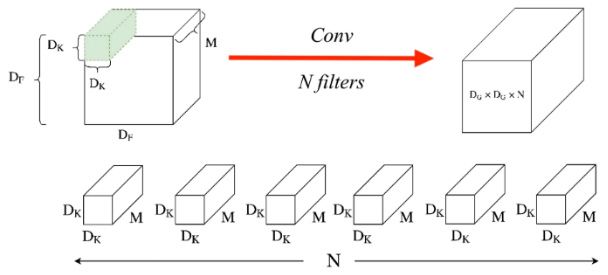
In the second chapter, we define the dataset that we have used in order to train our network. In details, we explain how it is formed.

In the third and fourth chapter, we explain how we choose to implement our code. In fact, we discuss about the methods that we have used to pre-process and to split the data of the dataset.

In the fifth chapter, we have the more interesting part of this report. We analyse in detail how we have built our Neural Network and, in particular we talk about two network that we have choose to use: X-Net and X-Net with quaternions.

In the last chapter, we compare and define the metrics that we will use, and we discuss the results obtained by training our two models. Furthermore, we compare it also with those in the paper. We discuss the motivation of the result and of the experiment that we have done.

1.1 Standard convolution



Mults once = $D_k^2 \times M$
Mults per Kernel = $D_G^2 \times D_K^2 \times M$
Mults N Kernels = $M \times D_G^2 \times D_K^2 \times N$

Figure 1: Standard convolution

1.2 Depthwise Separable Convolutions

Depthwise separable convolutions work with kernels that cannot be factored into two smaller kernels. It deals with the spatial dimensions, but also with the depth dimension (the number of channels).

In depth wise separable convolution, we have that the process of convolution is broken in two different operations:

1) **Depthwise convolutions: Filtering Stage**
2) **Pointwise convolutions: Combination Stage**

Depthwise convolutions (DC)

I compute the number of multiplications.
In this operation, instead of done the convolution for all the M channel (standard CNN's), the convolution is applied to a **single** channel at a time.

So, the kernels/filters will be of size $D_K \times D_K \times \mathbf{1}$.
In input, there are M channels, so M such filters are required.
The output will be of size $D_G \times D_G \times M$.

Mults once: $D_K \times D_K$
Mults 1 channel: $D_G^2 \times D_K^2$
We have that the filters are sliced by $D_G \times D_G$ times across all the M channels.
DC mults: $M \times D_G^2 \times D_K^2$

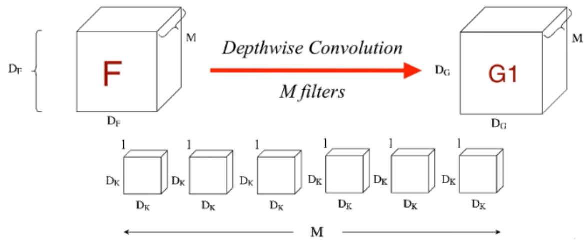


Figure 2: Depthwise convolutions

Pointwise convolution (PC)

It involves performing the linear combination of each of these layers.
In this operation, we apply on the M channels, an operation of 1×1 convolution.
The filter size will be $1 \times 1 \times M$.
We use N such filters.
The output is $D_G \times D_G \times N$.

Mults once: $1 \times M$
Mults 1 kernel: $M \times D_G \times D_G$
PC mults: $M \times D_G^2 \times N$

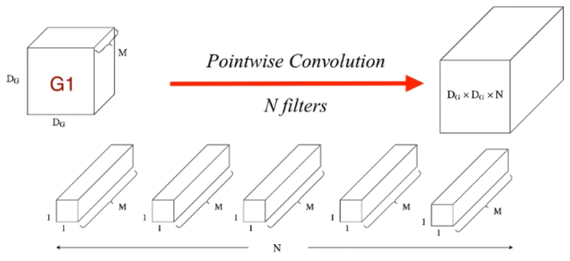


Figure 3: Pointwise convolutions

For overall operation:
Total mults = DC Multiplications + PC multiplications.

Total mults = $M \times D_G^2 \times D_K^2 + M \times D_G^2 \times N$.
By simplification, we obtain:
Total mults = $M \times D_G^2 \times (D_K^2 + N)$

Summary:

	Depthwise convolutions	Pointwise convolution	Standard
Mults once	$D_K \times D_K$	$1 \times M$	$D_K^2 \times M$
Mults 1 channel or 1 kernel	$D_G^2 \times D_K^2$	$M \times D_G \times D_G$	$D_G^2 \times D_K^2 \times M$
Tot. Mults	$M \times D_G^2 \times D_K^2$	$M \times D_G^2 \times N$	$M \times D_G^2 \times D_K^2 \times N$
	Total mults = DC Multiplications + PC multiplications		

Table 1: Difference between DSC and CNN

1.3 Comparison between Standard and Depthwise (in terms of power)

$$\frac{N°\ mults\ in\ Depthwise\ Separable\ convolutions}{N°\ mults\ in\ standard\ convolutions} = \frac{M \times D_g^2 \times (D_k^2 + N)}{M \times D_g^2 \times D_k^2 \times N}$$

We simplify and we obtain:

$$\frac{N°\ mults\ in\ Depthwise\ Separable\ convolutions}{N°\ mults\ in\ standard\ convolutions} = \frac{D_k^2 + N}{D_k^2 \times N} = \frac{1}{N} + \frac{1}{D_k^2}$$

Standard convolution has D_k^2 times more number of multiplications respect to the depth-wise separable convolutions.

1.4 Comparison between Standard and Depthwise (in terms of n. of parameters)

	Standard
Param 1 kernel	$D_k^2 \times M$
Param N kernels	$D_k^2 \times M \times N$

Table 2: N. parameters in CNN

	Depth-wise convolutions	**Point-wise convolution**
Param 1 kernel	D_k^2	M
Param M or N kernels	$M \times D_k^2$	$M \times N$
	Total: $M \times (D_k^2 + N)$	

Table 3: N. parameters in DSC

$$\frac{N°\ parameters\ in\ Depthwise\ Separable\ convolutions}{N°\ parameters\ in\ standard\ convolutions} = \frac{M \times (D_k^2 + N)}{D_k^2 \times M \times N} = \frac{1}{N} + \frac{1}{D_k^2}$$

1.5 Advantages and disadvantages of DSC

Advantages

If used properly, it manages to enhance efficiency without significantly reducing effectiveness, which makes it a quite popular choice.
Depth-wise Separable convolutions reduce computation time and number of parameters when compared to standard convolution.

Disadvantages

Because it reduces the number of parameters in a convolution, if your network is already small, you might end up with too few parameters and your network might fail to properly learn during training.

The role of the 1 x 1 Kernel

n kernel 1 x 1 x m in which:
n = number of output channels
m = number of input channels
can be used outside of separable convolutions.

Goal of kernel 1 x 1: increase or reduce the depth of an image. If I have the convolution that has too many or too little channels, a 1 x 1 kernel can help balance it out.

The main purpose of a 1 x 1 kernel is to apply non-linearity.
After every layer of a NN, we can apply an activation layer (ReLu, Softmax, etc…).
In order to increase the number of non-linear layers without significantly increasing the number of parameters and computations, we can apply a 1 x 1 kernel and add an activation layer after it.
This helps give the network an added layer of depth.

2 Dataset

2.1 Dataset 1: Brain CT images with Intracranial Haemorrhage Masks

We use the open-source dataset which is composed by 82 patients.
For each patient there are two folders: bone folder and brain folder.
For the studies that we have done, the only folder that is important is brain folder which contains approximately 30 image slices per patient, named with a number (i.e., "1.jpg").
In this folder there are some images taken in a different part of the brain.
If the image taken into account is an image that illustrate a part of the brain that has a stroke lesion, the following image remark only the problem encountered that is called with the same name of the previous image following by "_HGE_Seg.jpg".

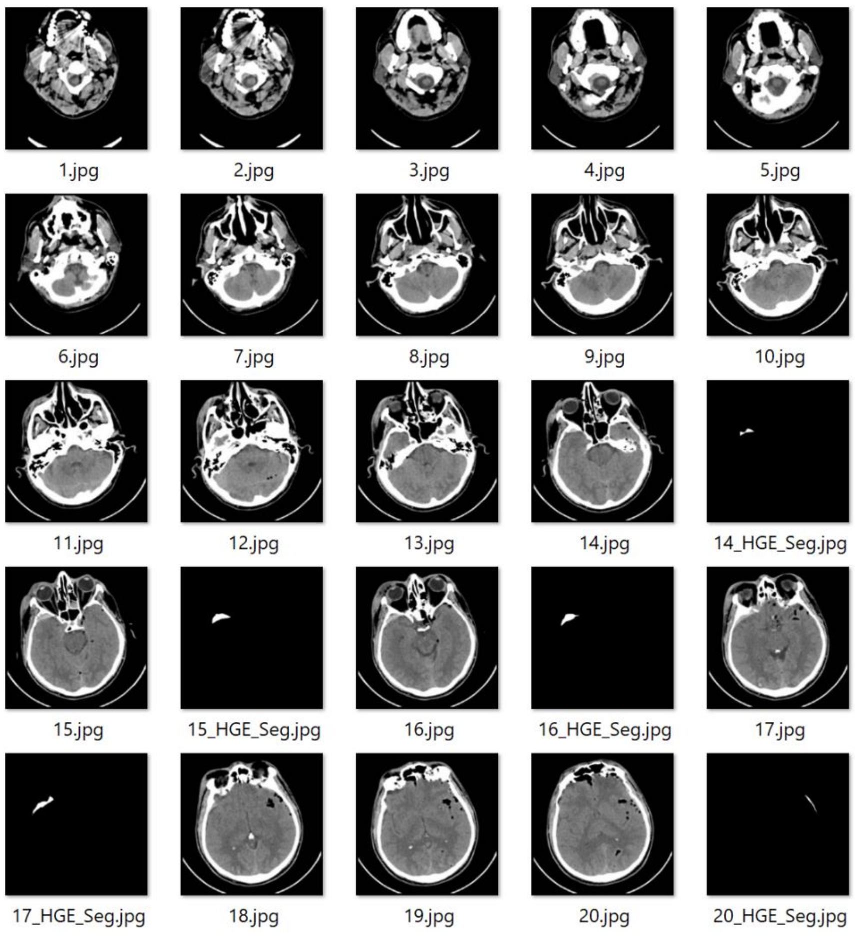


Figure 4: Example of dataset

3 Pre-processing of the data

In this chapter we explain the different ways that we have used in order to take the input data.
We use two different ways because the data in the dataset are insert in a different way.

3.1 Pre-processing for Dataset 1

This is the first step that we have followed. For this, we have created a list and we have sorted it. As we mentioned above, we have considered only the brain folder and not the bone folder because we are not interested in.

Then we have created two lists:

- images_i: take the input of the network;
- images_o: give the output of the network.

The images_o contains:

- for the images that represents the part of the brain that has no problem we have as output a black image;
- for the images that represents the part of the brain that has a stroke lesion, we return as output the image that illustrates the problem.

After this, we resize the images.

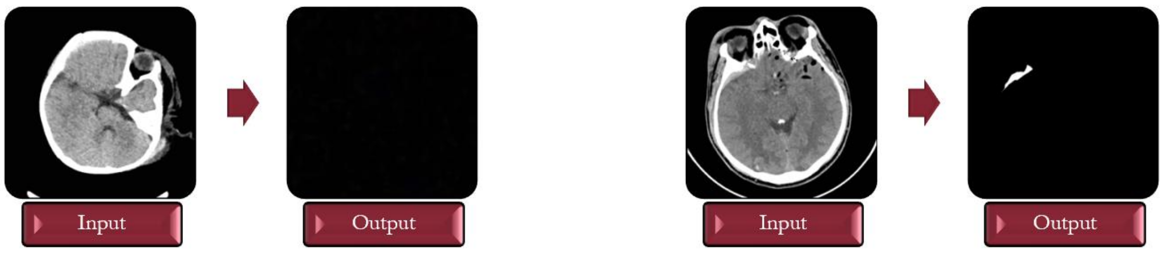


Figure 5: Dataset 1: Brain CT with Intracranial Haemorrhage Masks

4 Split of the data

This is the second step of our works.
We think that when we split the data is not correct to separate completely randomly the images that we have. So, we have splitted the 82 patients randomly, 65 pateintes (80%) for the train and 17 patients (20%) for the test.

5 Models

In this chapter we talk about two models that we have used
These models are very similar to each other, the only difference is that the second uses quaternions.
We take care to illustrate the details of the architecture and we will analyse each component in detail.
These two models are:

- X-Net;
- X-Net with quaternions.

5.1 X-Net

For image segmentation task, usually, we use SegNet, U-Net, 2D Dense-UNet, but we have two types of problem:

1) Limitation due to the heavy network parameters and many automatic segmenting methods do not consider the different sizes and locations of lesion.
2) They cannot capture long-range dependencies veritably, because they use some methods to collect information from a few surrounding pixels.

In order to have a more effective method for our objectives, we build an encoder-decoder architecture, that is called X-Net based on X-block and FSM as we can see in Figure 6.
It also adopts the skip connections.

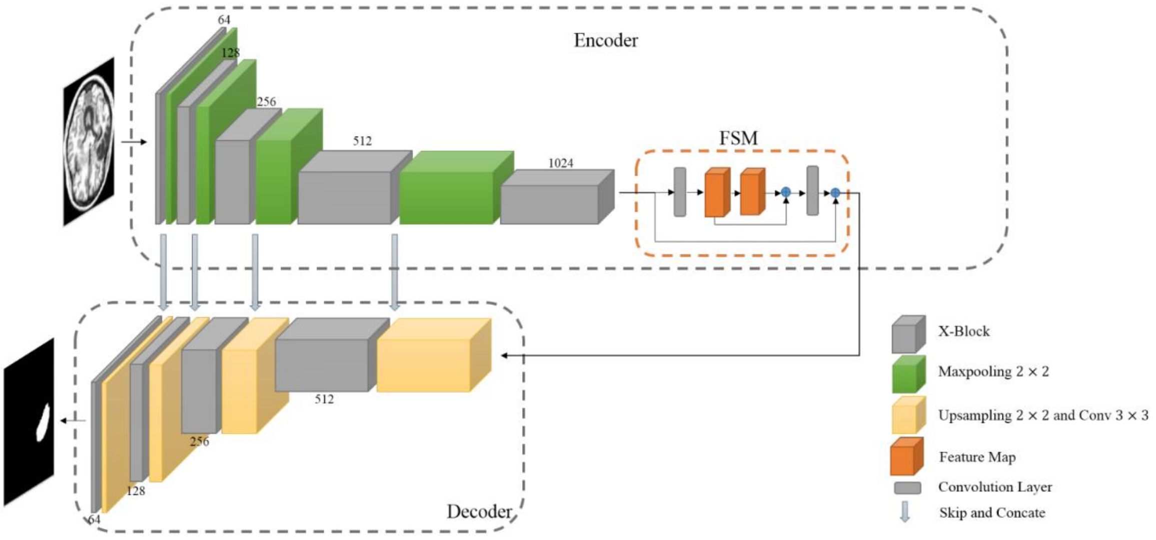


Figure 6: Encoder-decoder architecture

For each convolutional layer, both in the encoder and decoder, we use:

- Batch Normalization: that is a technique that allow to improve the velocity, performance and stability of a Neural Network. We use this in order to normalize the input.
- Rectified Linear Unit (ReLU) as activation function. We use this to introduce nonlinearity in the network and also because the computation of the derivative is fast and its value is zero only if the input is less than zero.

5.1.1 Implementation X-Net

We have implemented X-Net as in Figure 6.
This model is implemented in Keras. As optimizer, we use the Adam method and the learning rate is fixed at 0.001. The loss function is given by the Dice loss and Cross Entropy. The batch size for training set is set to 8. We set the maximum number of epochs to 100. But, we use Early Stopping to prevent overfitting.

5.1.2 Structure of encoder

The input of the encoder is an image. The size of image that we have used is $224 \times 192 \times 1$.
The encoder is composed by X-blocks and Maxpooling layers that are connected in cascade to produce high-dimension feature maps.
We use Maxpooling because is a method of reducing the size of an image by splitting it into blocks and keeps only the one with the highest value. Doing this, we reduce the problem of overfitting and only the area with grater activation are maintained.

So, the encoder architecture is composed by:

- X-block with 64 filters
- Maxpooling 2×2
- X-block with 128 filters
- Maxpooling 2×2
- X-block with 256 filters
- Maxpooling 2×2
- X-block with 512 filters
- Maxpooling 2×2
- X-block with 1024 filters

The output of this part is the input for the Feature Similarity Module (FSM).

5.1.3 FSM: feature similarity module

FSM (Feature Similarity Module) is a non-local operation.
It is used to capture long-range spatial information, which contributes to the segmentation of lesions with different scales and shapes.
We consider FSM as a network model that can be linked with other fully convolutional neural networks.

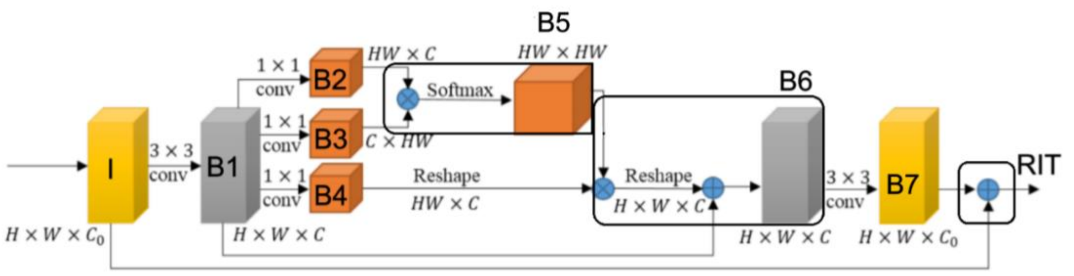


Figure 7: FSM: Feature Similarity Module

FSM is part of the Encoder of our neural network and in particular, allow us to connect the encoder to the decoder, as we can see in the Figure 6.

The input is given by the output of the fifth X-block of the Encoder.

The input is a feature map $X_0 \in R^{H \times W \times C_0}$. In our case we have $R^{14 \times 12 \times 128}$.
At first, we apply the 3×3 convolution and we obtain as output B1: $X \in R^{H \times W \times C},\ C < C_0$,
where $C = \frac{C_0}{8}$. This convolution is used to filter out the irrelevant feature from the input I, because of this we obtain a feature map with depth $C < C_0$.
Then, we have three convolutions 1×1 over B1, after which we obtain B2, B3, B4.
We do the dot product between B2 and B3. Then, we apply the softmax function and we obtain B5.
After this, we do the dot product between B4 and B5 and we obtain B6.
B6 has a wide range of contextual view and aggregates the long-range context.
Then, with B6 we do the convolution 1×1 and we apply the function add between B1 and B6.
We apply a 3×3 convolution over B6 and we obtain B7.
Finally, we apply the function add between the input and B7, and we obtain the output of FSM. We do this in order to avoid overfitting.

The output of FSM is the input for the Decoder.

5.1.4 Structure of decoder

The decoder is composed by X-blocks, Upsampling and convolution layers in order to recover the spatial resolution. So, the decoder architecture is composed by:

- Upsampling 2×2 and Convolution 3×3
- X-block with 512 filters
- Upsampling 2×2 and Convolution 3×3
- X-block with 256 filters
- Upsampling 2×2 and Convolution 3×3
- X-block with 128 filters
- Upsampling 2×2 and Convolution 3×3
- X-block with 64 filters.
- Convolution 1×1 with "sigmoid" as activation function.

The output is the image.

5.1.5 X-Net: details

For the reasons mentioned above, we build X-Net framework in which:

- we add a depthwise separable convolution layer in order to reduce the number of trainable parameters (that is less than what we have in the other methods) and it ensures the strength of feature extraction and representation;
- the long-range dependencies are effectively explored for brain stroke lesion segmentation.

So, U-Net is replaced by DSC (Depthwise Separable Convolution) in order to reduce the convolution kernel parameters.
Specifically, the DSC is a convolution that works independently over each channel of the input feature map.

At each X-block we have:

- Input feature map: $I \in R^{H \times W \times C_i}$, where C_i is the number of input channels.
- Two paths:
  1) First path:
     - The residual connection consists in 1×1 convolution layer in order to guarantee that the number of channels in output is equal to C_O.
  2) Second path:
     - Three depthwise separable convolution layers in cascade with kernel size 3×3.
     - Convolution of size 1×1.
- Output feature map: $O \in R^{H \times W \times C_O}$, where C_O is the number of output channels.
  It is obtained by summing the output of the first and the second path.

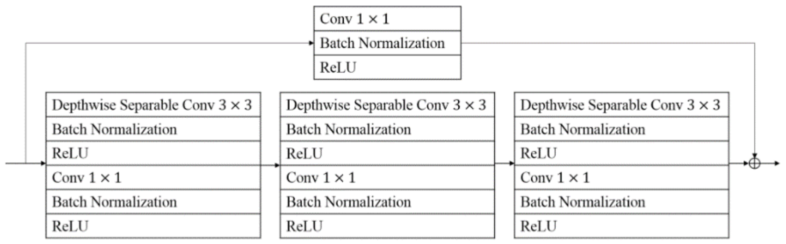


Figure 8: X-block

5.2 X-Net with quaternions Neural Network (QNN)

We have some difference that characterize the use of the real-valued NN instead of quaternions NN.

At first, we want to find a representation for the multidimensional data in order to decode the relations between the properties of observable entities.
Furthermore, in real-valued NN, since we have that the estimation of the parameters may fail to associate all the appropriate relations to all the pixels that compose a picture, we conclude that, in the latent space, the latent relation between RGB components of a pixel may not be suitably encoded.

Quaternions are fourth dimensional and the aim objective is to process and build entities that are composed by four elements (at maximum).
Quaternions use the Hamilton product. With this equation, the quaternion neural networks (QNNs) are able to capture the internal latent relations within the features of a quaternion.

NN are not able to reconstruct the spatial relation within 3D coordinates and within colour pixels, but QNN can do this because during the Hamilton product, the quaternion-weight components are shared through multiple quaternion-input parts and it creates relations within the elements.

In Figure 9, we can see:

- The multiple weights that decode the latent relations within a feature are considered at the same level as for learning global relations between different features;
- The quaternion weight w, during the Hamilton product, decodes the internal relations in only one quaternion Q_{out}.

The bigger NNs allow us to have better performance, while the QNNs are able to obtain results that are comparable or better for the same work, but they use four time less parameters than which are used in NNs. So, QNNs has a fourfold saving memory with respect to NNs.

In the end, the hyper-complex number are better than real numbers in order to made efficient model in multidimensional space, because of the natural multidimensional representation of quaternions and their ability to reduce the number of parameters.

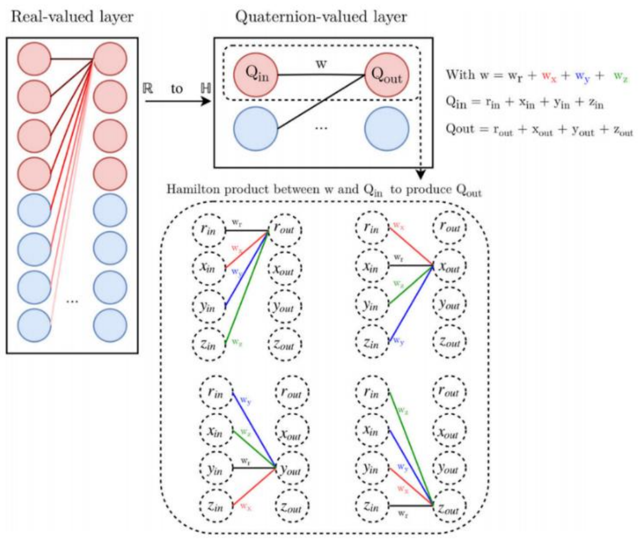

Real-valued layer
Quaternion-valued layer
ℝ to ℍ
Q_{in}
w
Q_{out}
With $w = w_r + w_x + w_y + w_z$
$Q_{in} = r_{in} + x_{in} + y_{in} + z_{in}$
$Qout = r_{out} + x_{out} + y_{out} + z_{out}$
Hamilton product between w and Q_{in} to produce Q_{out}

Figure 9: Real-valued layer compared to quaternion-valued layer.

The quaternion activation functions are adapted to train QNNs with the standard backpropagation algorithm and in particular there are two classes of quaternion-valued activation function:

1) **Fully quaternion-valued functions**: are simply the extensions of real-valued functions to the hyper-complex domain. (i.e., sigmoid, hyperbolic tangent).
   Issue: because of the big number of singularities, they need to have a careful training phase.
   They achieve better result.
2) **Split functions** (more used): we apply a convolution real-valued function to the quaternion in order to reduce the singularities.
   Issue: each quaternion-valued layer learns an internal relation scheme independently of the other layers.
   They are worse than the fully quaternion-valued functions, even if there are many applications and a simple training phase.

5.2.1 Implementation X-Net with Quaternions

The piece of code that we have taken from external resources are reported in the Notebook.
The black and white image were represented in the quaternion domain in the three complex channels and the real channel was set to zero. We have done this choice because we had images in black and white and not with colors. If we have had the color images, we would have put the primary color component for each channel.
To implement this network, we have used PyTorch.
First, we have adapted the Depthwise Separable Convolution class in order to work with quaternions, but then the implementation of the X-Net is the same that we have done for the first model.
So, we have the same structure of the X-block, but also the same structure of the Encoder, FSM and Decoder.
The crucial change is that in all this implementation we have added the forward function and we have modified the dimension of all the element in the network to fit the quaternion.

6 Tools used and Results obtained

In this chapter we start talking about the metrics that we have decided to use.
Then we discuss the results obtained with the dataset that we have used, evaluated by applying the X-Net model (Model 1) and X-Net with quaternions (Model 2).

At the end we have done two types of comparisons:

- We have compared the results obtained with X-Net by using our dataset, with those in the paper;
- We have compared the results obtained with X-Net model and X-Net with quaternions.

6.1 Metrics used

We decide to use the metrics written in the Table 4. In particular, we use:

- **Dice score**:

$$\frac{2 \cdot Area\ of\ Overlap}{total\ pixels\ combined}$$

- **Loss function:**

$$BinaryCrossEntropy + (1 - Dice)$$

6.2 Results obtained with dataset 1 evaluating with Model 1

To show the results, we have decided to use the loss function and the dice. Furthermore, we have done the same train of the data by using early stopping, that is a regularization method used to avoid overfitting, and without using it. Let us start to compare and to discuss the different results that we have obtained by doing these experiments.

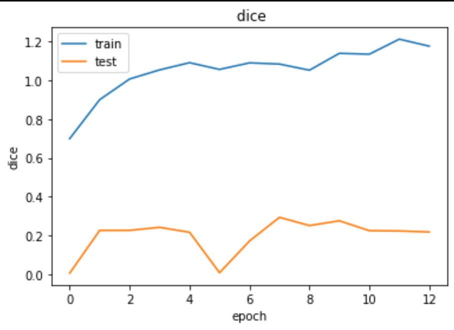
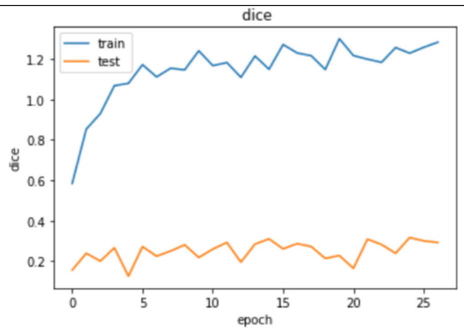
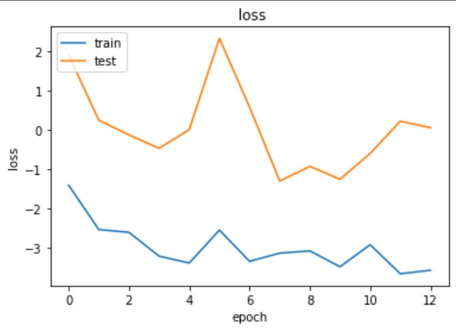
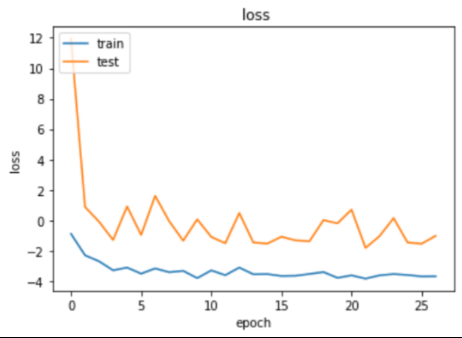
	Early Stopping	Without early stopping
	DATASET 1	
Dice		
Loss		

Table 4: Dice and loss with and without Early Stopping

We can see from the plot in Table 4 that if we use early stopping, the train stops after 13 epochs.
In fact, we can notice that from the epochs 0 the train set always increase approximately with a logarithmic trend, reaching dice = 1.1. Instead, the test set, has a minimum in the fifth epoch. In fact, at epoch zero start to increase, then decrease and after the fifth epoch start to increase reaching dice = 0.2.

For what concern the loss function, we have a maximum in the fifth epoch for the test set. At start we have a parable with the concavity downwards, but after the peak, it starts to decrease and at the end it increases the value reaching loss = 0. So, we have a maximum in loss = 2 (epoch = 5) and minimum in loss = -1 (epoch = 7 and epoch = 9).
For the train set we have a trend that start at loss function = -1.2 and finish at loss = -3.7.

In general, we can see in both plots, that if we had continued the train, we would certainly have encountered overfitting because we have the test and train set that they begin to drift further and further away.

Without early stopping, we have decided to train the model for 25 epochs.
We can see that for dice function we have the train set that start at a low dice = 0.6 and reach the value dice = 1.3. So, we have an increasing of the value, that is very positive.
For the test set, we have a lot of peaks, but in general we have a trend that is always around dice = 0.2.

For the loss function, we have an exponential decrease for the test set, that start from loss = 1.2 and achieve the value loss = 0. But, after 3 epochs it starts to have a lot of peaks and from there on it always assumes a value around zero.
The train set has a trend that is linear with value loss that tends to -3.8.

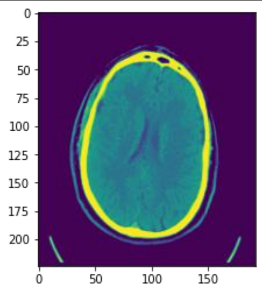
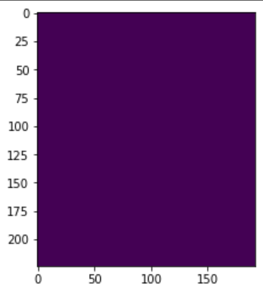
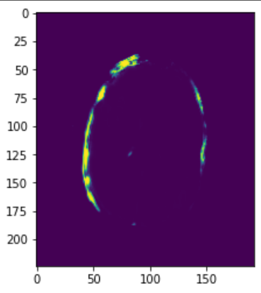
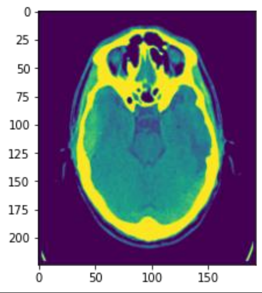
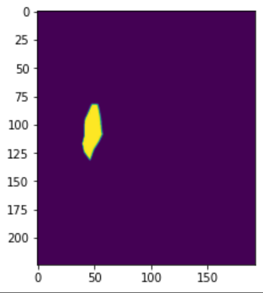
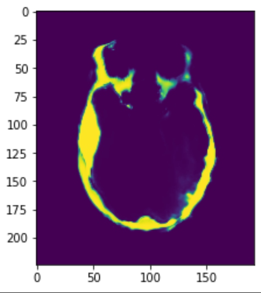
	Original input	**Original output**	**Our output**
Image that has not Broke Lesion			
Image that has Broke Lesion			

Table 5: Example of random samples using X-Net (with FSM)

In table 5, we can observe the results on two random test samples.
We can see that the network wrongly classified the bone as a lesion, but in the second image, in which we have a Lesion, the model correctly segments the lesion. So, if we cut the bone from the end image, we will have a perfectly segmentation of the lesion. We can cut the bone because we have a folder in which are store the images.

6.3 Comparison: X-Net with FSM and X-Net without FSM

We think that is interesting to compare the performance of our network with FSM and without it because with this comparison we understand the effectiveness of FSM. We train the Model using early stopping.

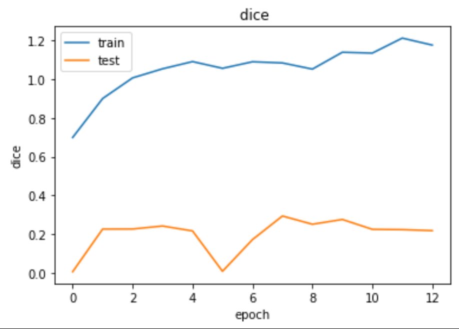
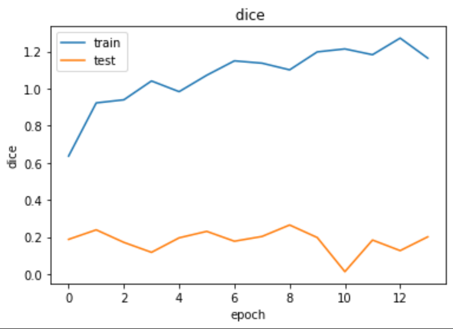
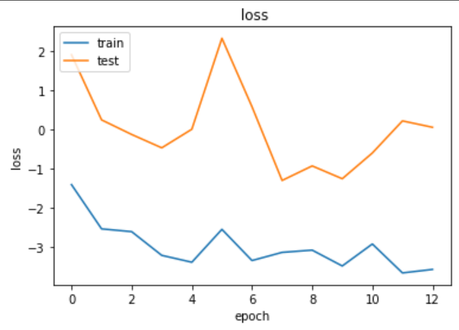
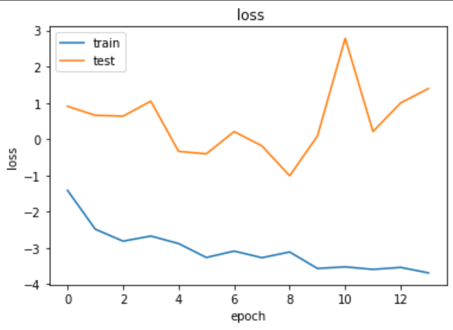
	X-Net with FSM	X-Net without FSM
Dice		
Loss		

Table 6: Dice and loss for X-Net with FSM and without it

We can see from Table 6 that the result that we have obtained are very similar, and it is a little better the model with FSM. But it is possible that this result is the consequence of the small dimension of our dataset. We think that if the dataset is bigger, the power of FSM is much visible.

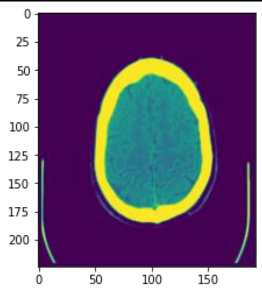
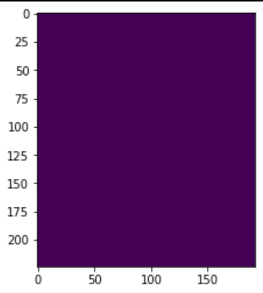
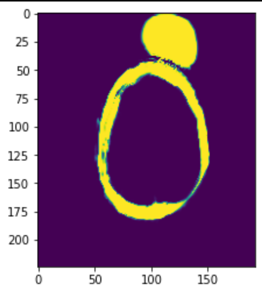
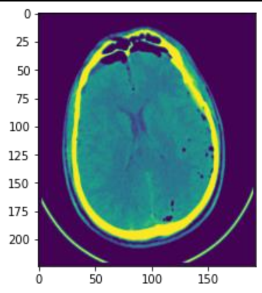
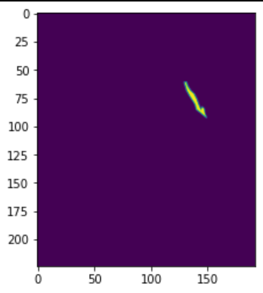
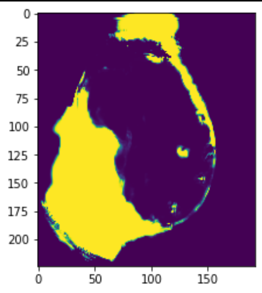
	Original input	Original output	Our output
Image that has not Broke Lesion			
Image that has Broke Lesion			

Table 7: Example of random samples using X-Net without FSM

As we can see from Table 5 and Table 7, in this case we see a big difference. In fact, if we use X-Net without FSM, even if we cut the bone from the end image, we will not have a perfectly classification of the lesion because the problem is that we have a big area that are marked in our output and not only those of the bones. So, we can conclude that the work of FSM is very important.

6.4 Results obtained with dataset 1 evaluation with Model 2

As we can see in table 8, we have used the loss function given by:

$$BinaryCrossEntropy + (1 - Dice)$$

For Dice metric, we have not the good results, but we can see that it tends to increase, that is positive. We can have some different reasons to explain this trend.
Firs, it is possible that the number of epochs that we have chosen is not sufficient for a quaternion network. The trend can be better or worse if we train it again and again.
Secondly, it is probably that our dataset is not big enough to be trained with QNN.
Finally, this structure of the network might not adapt to quaternions or the choices that we have done to represent the images in the quaternion domain is not adapt.
In any case, if we observe carefully, we can say that the train set is increasing.

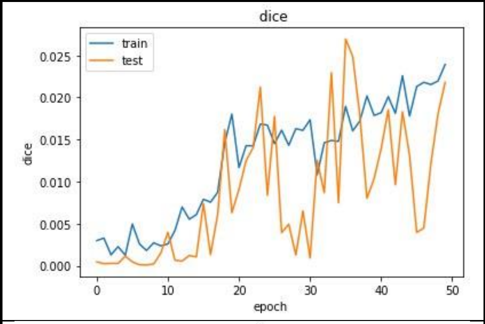
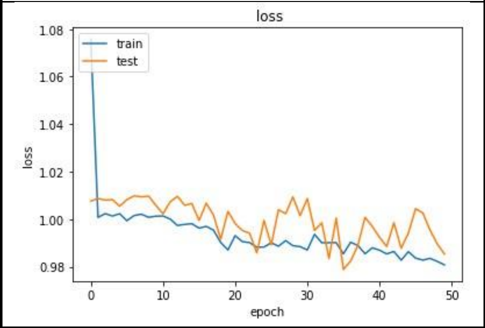
Dice	
Loss	

Table 8:Dice and loss for X-Net with quaternions

For loss function, we have a good result, but it can be better. In fact, it tends to decrease. So, if we train the QNN many times it might achieve a better loss.

In table 9, we have used the loss function given by:

$$1 - Dice$$

As we can see in Table 9, the results that we have obtained by training X-Net with quaternion is not so good like in the Table 8.
In any case, if we observe carefully, we can say that the train set is increasing the dice and decreasing the loss, that is good.
Instead, the trend of the test set has the same behaviour of the train set, but it presents a lot of peaks. For this reason, we can conclude that if we train more our QNN, we can obtain better results. So, we think that the first hypothesis is right.

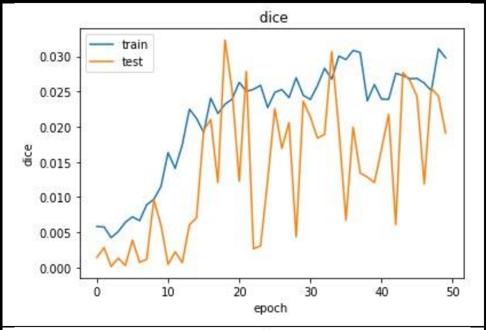
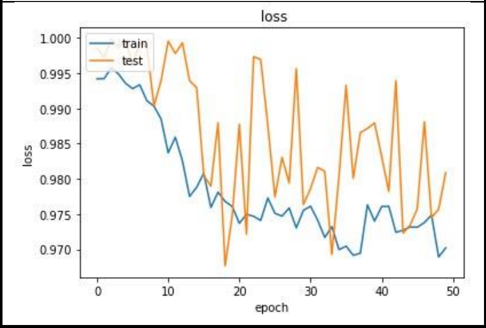
Dice	
Loss	

Table 9: Dice and loss for X-Net with quaternions

With the quaternions the network tends more to return black images. For this we have tried to put only the dice as a loss function to try to push the network not to return black images. We have a different behaviour because with the dice every time we use the colors, while with the loss function it does not.

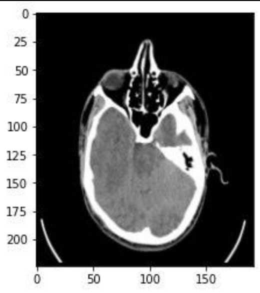
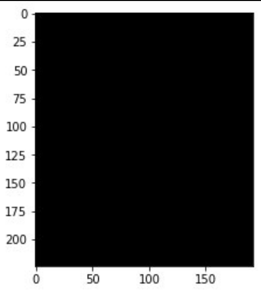
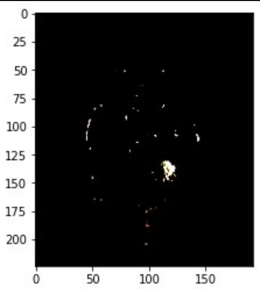
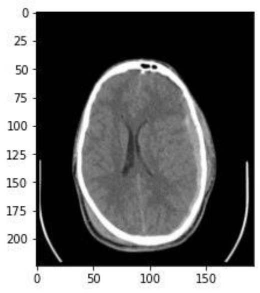
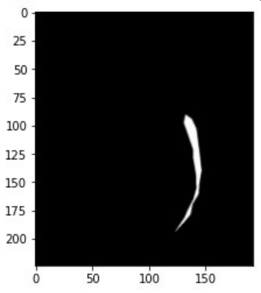
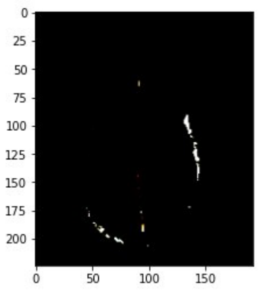
	Original input	**Original output**	**Our output**
Image that has not Broke Lesion			
Image that has Broke Lesion			

Table 10:Example of random samples using X-Net with quaternions

As we can see in Table 10, the result that we have obtained in the image that has broke lesion is good, because it highlights the problem that the patient has.
But for the image that has not the broke lesion, the result is not good, because we have a white point so marked.

6.5 Comparisons between our results and paper results

Model	Dice	#Parameters
ResUNet	0.4702	33.2M
DeepLab v3+	0.4609	41.3M
2D Dense-UNet	0.4741	50.0M
PSPNet	0.3571	48.1M
SegNet	0.2767	29.5M
U-Net	0.4606	34.5M
X-Net (paper)	0.4867	15.1M
X-Net (ours)	0.2188	15.1M

Table 11: Comparison between our results and paper results

In the paper, the authors have used ATLAS dataset that contains 43281 images, instead our dataset contains 2501 images.

So, if we compare the dimensions of the two datasets (ATLAS and ours) and given only the 6% of the images that have ATLAS dataset, we have obtained the final dice = 0.2188 that is good compared to 0.4867 (paper).

6.6 Comparisons between Model 1 and Model 2

Model	Dice	#Parameters
Model 1	0.2188	15.1M
Model 2	0.0230	26.5M

Table 12: Comparisons between Model 1 and Model 2

As we can see in Table 12, we have that the number of parameters in Model 2 is almost double respect to which of the Model 1. This is a problem because usually the number of parameters should decrease in a quaternions network.
Instead, for Dice value, we have that Model 1 is better. We have a so low dice value for Model 2 for the reasons that we have explained above.

7 Conclusion

In building this project we have encountered some obstacles, in particular with the Dataset. In fact, we were intent on use the ATLAS dataset that is the same that has used our paper of reference; but for some reasons we don't use this.

We think that in the future, we can use the original dataset ATLAS and do more comparisons. Furthermore, we can also think to extend this project, for example by using the transformations (i.e., wavelet) in order to use the quaternions better and not in the simplest approach. Otherwise, we can also extend it by using a multimodal approach in order to analyse similar images linked to the same measure.

In any case, we are satisfied of our work.

8 References

1. Kehan Qi, Hao Yang, Cheng Li, Zaiyi Liu, Meiyun Wang, Qiegen Liu, Shanshan Wang:
   X-Net: Brain Stroke Lesion Segmentation Based on Depthwise Separable Convolution and Long-range Dependencies. (2019).
2. Titouan Parcollet, Mohamed Morchid, Georges Linarès: A survey of quaternion neural networks (2019).
3. François Chollet, Googl Inc.: Xception: Deep Learning with Depthwise Separable Convolutions (2017).
4. Keras.io
5. https://www.kaggle.com/vbookshelf/computed-tomography-ct-images [for dataset 1]
6. Sook-Lei Liew, Julia M. Anglin, […] Alison Stroud
   A large, open source dataset of stroke anatomical brain images and manual lesion segmentations [for dataset 2: ATLAS]