Medical Image Segmentation and Application

# Project Report

**Brain Tissue Segmentation on IBSR 18 dataset**

Abdullah Thabit
Pierpaolo Vendittelli
Prem Prasad


Medical Imaging and Applications (MAIA)
Universitat de Girona
7-1-2020

# Contents

# 1. Introduction

From the past few decades, the fast development of noninvasive brain imaging technologies has opened new doorways in analysing and studying the brain function and anatomy. The advances in brain MR imaging have provided large amount of data with an increasingly high level of quality, and the analysis of these large and complex MRI datasets has become a tedious and complex task for clinicians. This manual analysis is often time-consuming and prone to errors due to various inter- or intra-operator variability studies. These difficulties in brain MRI data analysis required inventions in computerized methods to improve disease diagnosis and testing. Nowadays, computerized methods for MR image segmentation, registration, and visualization have been extensively used to assist doctors in qualitative diagnosis.

Brain MRI segmentation is an essential task in many clinical applications because it influences the outcome of the entire analysis. This is because different processing steps rely on accurate segmentation of anatomical regions. For example, MRI segmentation is commonly used for measuring and visualizing different brain structures, for delineating lesions, for analysing brain development, and for image-guided interventions and surgical planning. This diversity of image processing applications has led to development of various segmentation techniques of different accuracy and degree of complexity.

### 1.0.1 Aim of the Project

The goal of image segmentation is to divide an image into a set of semantically meaningful, homogeneous, and non-overlapping regions of similar attributes such as intensity, depth, color, or texture. The segmentation result is either an image of labels identifying each homogeneous region or a set of contours which describe the region boundaries. In this project, we aim to automatically segment the various tissues present in the 3D brain volumes, which are White Matter (WM), Grey Matter (GM) and Cerebro-Spinal Fluid (CSF).

### 1.0.2 Description of the Dataset

The Internet Brain Segmentation Repository (IBSR) provides manually-guided expert segmentation results along with magnetic resonance brain image data. Its purpose is to encourage the evaluation and development of segmentation methods. The dataset includes 1.55mm data. Eighteen subjects currently available (10 subjects were used for training, 5 for validation and 3 for testing). For each subject, there are T1-weighted volumetric images that have been 'positionally normalized' into the Talairach orientation (rotation only). It is also to be noted that this data has been processed by the CMA 'autoseg' biasfield correction routines. The 18 scans (256 x 256 x 128) of the subjects have varying resolutions (0.84 x 0.84 1.5mm$^3$, $0.94x0.94x1.5mm^3 and 1.0x1.0x1.5mm^3$).

### 1.0.3 Strategy used

For this project, we decided to use a deep learning approach to perform the segmentation of the 3D MRI volumes instead of multi-atlas. PyTorch was the framework of choice used for this project, while training and experiments were run on Google Colab using GPU.

# 2. Method (Proposal Analysis)

This chapter summarizes all the work we did for this project, from description of the used architectures to the set up of the experiments.

## 2.1 Architecture

For this project we used two different architecture, where the second ( and final one) was chosen due to an improvement over the first as it will be shown in the following pages.

### 2.1.1 The U-NET

The first architecture we used for this project was the one presented from (PostDoc) Sergi Valverde during the seminar we attendended in early December. It is a U-NET architecture was developed in 2015 from the University of Freiburg, Germany with the goal of biomedical (microscopy) image segmentation.

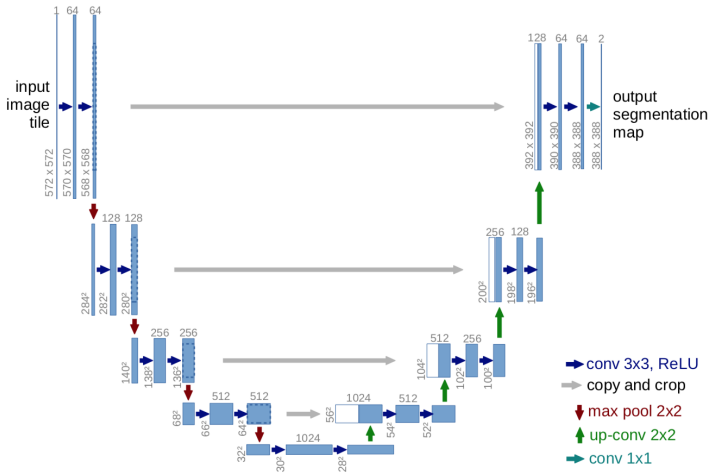



Figure 2.1: U-NET Architecture

As we can see from the figure above, the network is an encoder - decoder network, it means that the first part of the network, the encoder, compress and extracts features from the image, arriving at the last layer having a single vector representing the image. The decoder part reconstruct the segmented image through a series of up-scaling layers. This is an architecture that was developed for 2D image segmentation, therefore to use it with 3D images ( as in our case) we need to adapt it to work with patches since working with the entire 3D volume leads to big problems such as lack of memory.

The network we used has 3 convolutional layers (with the goal of feature extraction) each followed by a max pooling layer with the goal of downscaling the image.

For the decoder part we have as well 3 convolutional layers each followed by an up scaling layer in order to reconstruct the segmented image.

The main difference between this network and other convolutional Neural Network aimed at the segmentation task like the SegNET, is that the U-NET model introduces the copy and crop elements, so activations for each of the encoder layers are concatenated (or added) to the same level decoding layers in order to help to reconstruct the image structure prior to segmentation.

### 2.1.2 ResUnet

Another Unet architecture variant (shown in Figure 2.2 below) was also used as an improvement to the original Unet architecture discussed above. This architecture variant uses Residual blocks in the encoder and decoder parts of the Unet in order to exploit the res-connections at each block to help for a better gradient flow through the network layers.

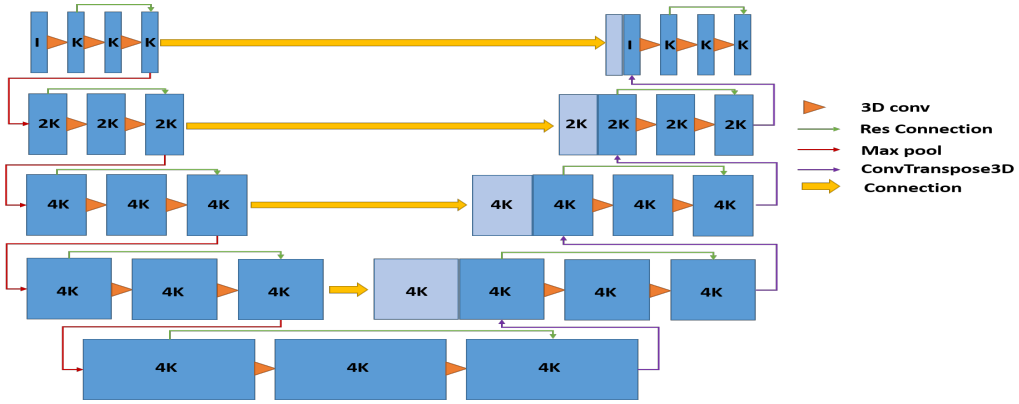



Figure 2.2: ResUnet network, representing the encoding and decoding part, where $k$ is the initial number of feature maps

The input patch of 3D volume is first being fed to the network through a series of convolutional layers inside the first Residual block (with skip connection between input and output). The output of the Residual block is then down sampled with a convolutional layer of stride 2 to reduce the size of feature maps. The same cycle of Residual convolutional block and down-sampling is repeated for a couple of times (4-5 is usually enough for medical imaging applications) to reach the most abstract features of the input (know as the latent space). For the decoding part, the output of the latent space Residual block is up-sampled a couple of times to reconstruct the segmentation of the input volume patch as in the normal Unet architecture.

In this architecture, each Residual block is composed of $n$ repetitive layers, these layers are a 3D convolutional layer and a normalization layer followed by an activation Function, where the input of the first convolutional layer in the block is concatenated to the output of the output of the activation layer. as shown in Figure 2.2 above.

The ResUnet adapted in our project is the one implemented for brain Glioma segmentation[1], which has 5 down sampling steps to the latent space followed by their counterpart in the decoder side. The input first goes through a normalization layer and a single convolutional layer with padding before feeding it to the first

[1]https://github.com/karinvangarderen/glassimaging

Residual block. A Fully connected convolutional layer at the end is also added before the classifier layer that gives the classes probabilities.

## 2.2 Design and Implementation

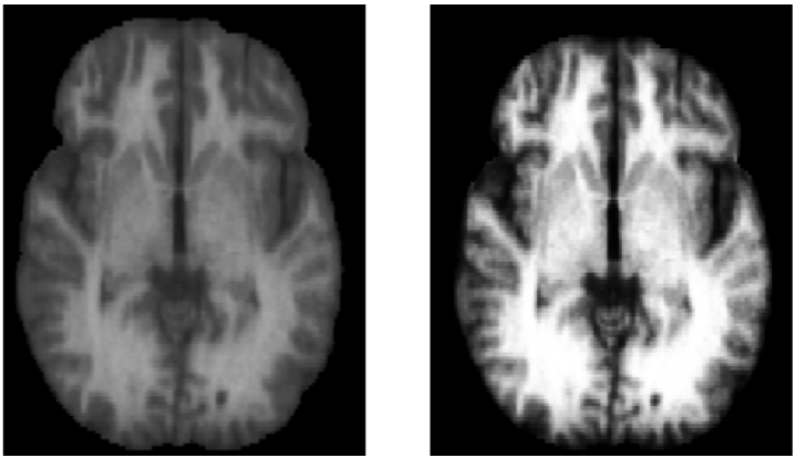
In this section all the experiments will be introduced and then commented in the result section.

### 2.2.1 Preprocessing

Since one of the problem with the given dataset is that the images are coming from different scanner, and so the level of intensities change a lot from image to image, one of the way to improve results is to apply a sort of preprocessing to the images, before sending it to the network. We tried three different preprocessing techniques which are Contrast Enhancement, feature scaling and normalization.

#### 2.2.1.1 Histogram equalization

Histogram equalization is a technique used to improve contrast in an image.It spread out the most frequent intensity values, stretching out the intensity range of the image. An example of a result of histogram equalization is shown in the next figure.



(a) Original IBSR_01 (b) Equalized IBSR_01

Figure 2.3: Original Image (a) and Preprocessed Image (b)

As mentioned in the introduction, the dataset we have has heterogeneity in image intensities, so histogram equalization appeared to be a fast preprocessing form which could immediately give results.

#### 2.2.1.2 Normalization

This method is one of the two methods implemented in Sergi's code and it is simply a normalization of the image which results in a new image having $\mu = \mathbf{0}$ and $\sigma = \mathbf{1}$, according to the following formula:

$$x_{new} = \frac{x - \overline{x}}{\sigma}$$

#### 2.2.1.3 Feature scaling

The last method, is a re-scaling method in order to obtain values in range [0,1]. It is given by the following formula:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

This normalization technique, together with the previous solves the problem of heterogeneity in the intensities.

### 2.2.2 Architecture's experiments

As discussed in the previous section, two main networks were adapted in our work: Normal Unet and ResUnet (Unet with Residual blocks). To Train the networks, different parameters were optimized and tuned. Table 2.1 below shows the different training parameters we experimented with:

Table 2.1: Training parameters

| **Parameter** | **Range/Options** |
|---|---|
| Optimizer | Adam-Adadelta |
| Learning rate | 0.001 - 0.0001 |
| Patch size | 16, 32, 64 |
| step size | 8, 16 |
| Batch size | 32, 64, 128 |
| Early stopping | no, yes (10 epochs patience) |
| Learning rate scheduler | no, yes (5 epochs patience) |

After a couple of training experiments we found it best to use Adam optimizer with an initial learning rate of (0.001) that is reduced to half if validation loss is not improved after 5 epochs and with early stopping if it did not improve after an additional 5 epochs. Best batch size was 32, a higher batch size took longer time to train for with no much improvement, and a smaller batch size performed slightly worse. For Patch size, a smaller patch size gets used up when the encoder goes deeper and down sampled to the latent space, and a bigger patch size sometimes gets the GPU out of memory and takes longer time to train, so a patch size of 32x32x32 was a good trade of and gave us the best performance.

We started the experiments with the base Unet composed of 4 blocks of conv layers followed by max pooling with a Relu activation function. Then we started adding and experimenting with other layers. We tested the network with a normalization layer after every conv layer. Normalization layers included Batch norm and instance norm layers, where instance normalization normalizes the input across each channel independently instead of all channels as in batch norm. A drop out layer was also implemented to help the network generalize.

Then we adapted the Unet architecture with Residual blocks described in the previous section. We tested the network with different configurations, such as by adding dropout within the residual blocks or at the beginning and end of the network. Different Normalization layers were also tested in this network including Group Normalization, which gives more freedom and can be adjusted to be a batch norm or instance norm layer.

Another experiment tested different number of feature maps in each conv layer (mainly k=16 and k=32 as this is hindered by the RAM capacity of the GPU). The effect of $n$ number of conv layers in the Residual block was also tested with two values n=2 and n=3. Other hyperparmaeters tuning experiments was also conducted to end up with the final dice scores of the best performing model as will be further analyzed in the results section

### 2.2.3 Loss Functions

At its core, loss functions are used to evaluate how well our algorithm models the dataset.

#### 2.2.3.1 Cross Entropy

Cross-entropy is a measure from the field of information theory, building upon entropy and generally calculating the difference between two probability distributions. It is a measure of the difference between two probability distributions for a given random variable or set of events. The cross entropy formula takes in two distributions, p(x), the true distribution, and q(x), the estimated distribution, defined over the discrete variable x and is given by:

$$H(p,q) = -\sum_{\forall x} p(x)log(q(x))$$

This loss examines each pixel individually, comparing the class predictions (depth-wise pixel vector) to our one-hot encoded target vector. Because the cross entropy loss evaluates the class predictions for each pixel vector individually and then averages over all pixels, we're essentially asserting equal learning to each pixel in the image. This can be a problem if your various classes have unbalanced representation in the image, as training can be dominated by the most prevalent class.

#### 2.2.3.2 Dice Coefficient Loss Function

Another popular loss function for image segmentation tasks is based on the Dice coefficient, which is essentially a measure of overlap between two samples. This measure ranges from 0 to 1 where a Dice coefficient of 1 denotes perfect and complete overlap. The Dice coefficient can be calculated as:

$$DSC(x,y) = \frac{2 * |X \cap Y|}{|X| + |Y|}$$

For the case of evaluating a Dice coefficient on predicted segmentation masks, we can approximate $|A \cap B|$ as the element-wise multiplication between the prediction and target mask, and then sum the resulting matrix.Because our target mask is binary, we effectively zero-out any pixels from our prediction which are not "activated" in the target mask. For the remaining pixels, we are essentially penalizing low-confidence predictions; a higher value for this expression, which is in the numerator, leads to a better Dice coefficient.

This Loss function tends to **minimize** the similarity between the two sets, therefore for our optimizing purpose we take **1 - DSC**.

# 3. Results

In this chapter we will comment on the results obtained from all the experiments described before.

## 3.1 Preprocessing

Table 3.1 shows the effect of preprocessing in the experiments we did. All the experiments of preprocessing were done using the standard U-NET architecture.

Table 3.1: Mean dice scores for the validation set for different experiments in the preprocessing

| Experiment\Tissue Type | CSF | GM | WM |
|---|---|---|---|
| Base Model (BM) | 90.30 | 93.97 | 92.96 |
| Feature Scaling (FS) | 90.38 | 93.96 | 93.05 |
| Contrast Enhancement | 70.60 | 87.1 | 74.8 |
| Contrast Enhancement v2 | 44.26 | 86.22 | 81.40 |

From the table above, the Feature Scaling (min-max normalization) seems to perform a bit better than the normalization, while we can notice that prepocessing has the worst impact on the validation, both modifiying all the data and modifying only the training data (v2). Especially in the CSF, we can notice a big drop, this is probably due to the fact that the contrast enhancement tends to make the CSF darker, so probably the network classifies it as background. The difference between the two version of contrast enhancement are due to the fact that the validation data (not enhanced) is very different from the training data, and as we can see from the curve, the training loss becomes small but it fails to generalize well.

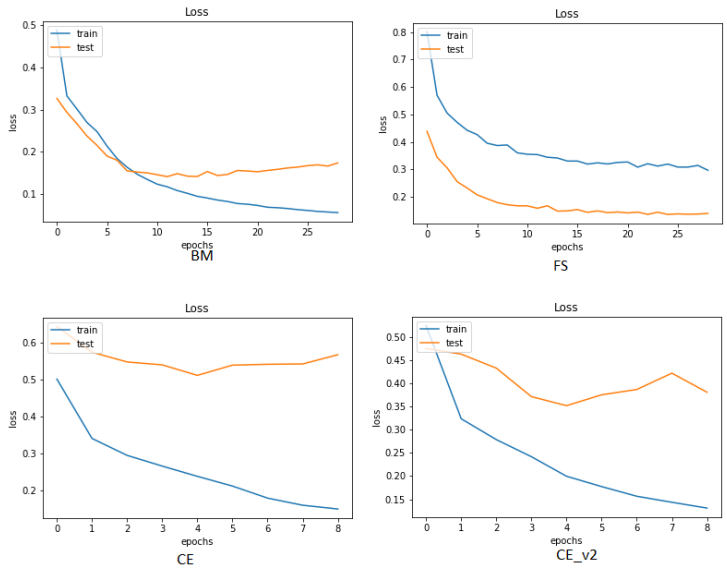



Figure 3.1: Dice scores for the validation set for different preprocessing methods

## 3.2 Architecture

Table 3.2 and Figure 3.1 below show the different experiments done with respect to the architecture of the network. It compares the base Unet architecture with the different addition aiming at improving the Dice scores of the segmentation results for each tissue type: CSF, GM and WM. Table 3.1 presents the mean dice scores for each model, while Figure 3.1 presents the dice scores for each sample in the 5-samples validation set.

Table 3.2: Mean dice scores for the validation set for different experiments in the model architecture

| Experiment\Tissue Type | CSF | GM | WM |
|---|---|---|---|
| Base Model (BM) | 90.30 | 93.97 | 92.96 |
| Early Stop (ES) | 90.23 | 94.04 | 93.52 |
| BatchNorm and Dropout (BN&D) | 91.30 | 94.50 | 93.98 |
| Instance Normalization (IN) | 91.37 | 94.75 | 94.43 |
| ResUnet | 92.05 | **95.07** | 94.76 |
| ResUnet with Dropout (ResUnet_D) | **92.35** | 95.04 | **94.86** |

As we can see from both Table 3.2 and Figure 3.1 above, our best dice scores on the validation set were obtained by using the ResUnet architecture with dropout layers of a 0.1 rate inserted after every conv layer in the Residual blocks, in both the encoder and decoder parts of the Unet.

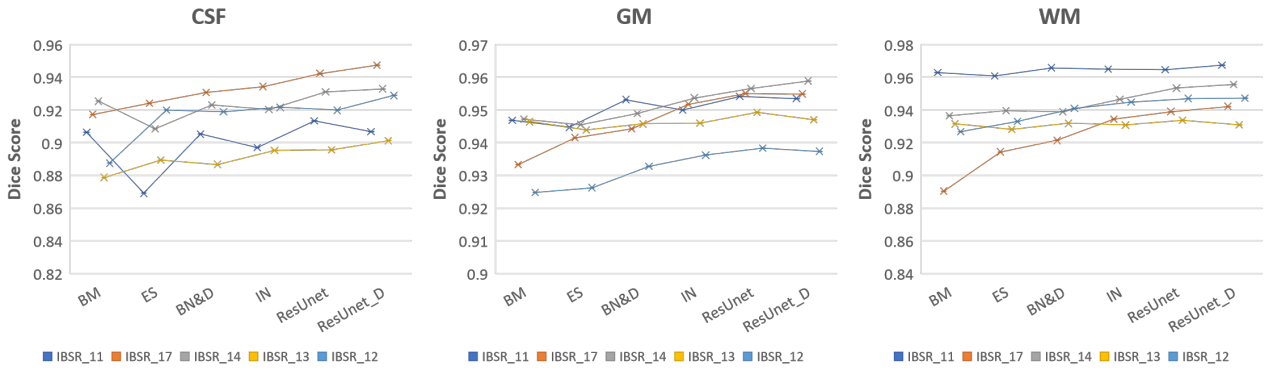



Figure 3.2: Dice scores for the validation set for different experiments in the model architecture

The flow of these experiments starting with the base model and finishing with the best performing one was done through analyzing not only the dice scores but also the loss functions at each experiment. Figure 3.2 below shows the training and validation loss functions of some of the performed experiments (mentioned in Table 3.1)

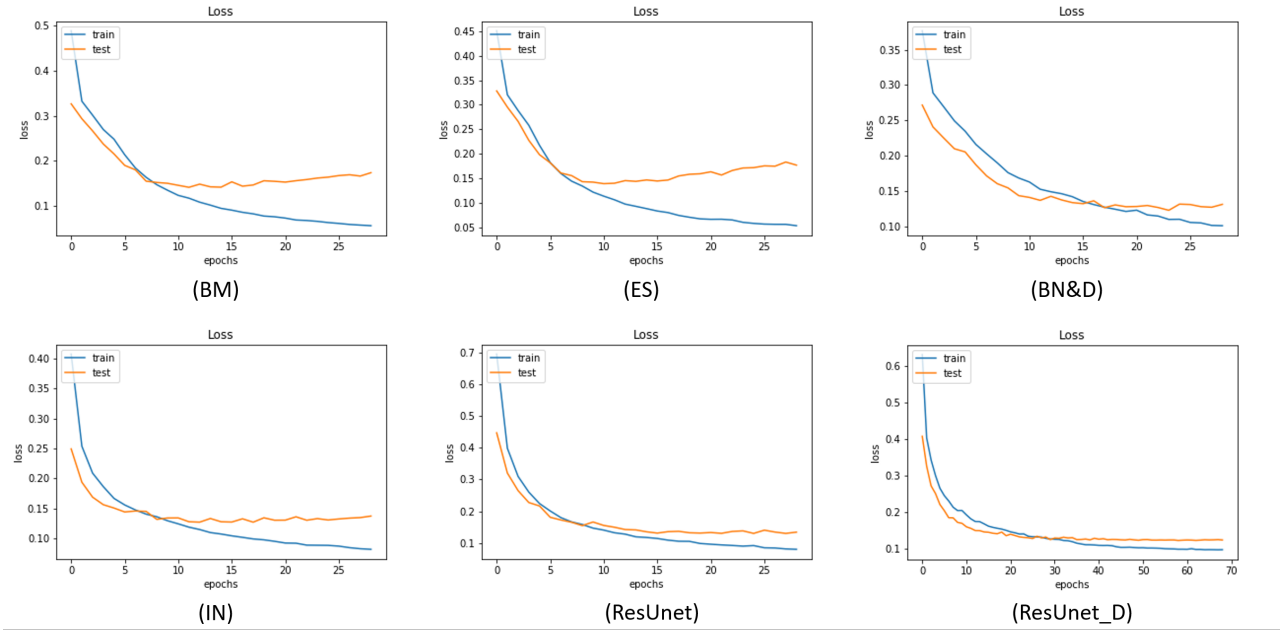



Figure 3.3: Training and validation loss functions for some of the network experiments mentioned in Table 3.1

From Figure 3.2 after running the experiment for the Base Model (BM) we noticed a clear sign of overfitting where the training loss continues to decrease while the validation loss stagnate at some point and even started to increase. This phenomena hinder the deep learning models from generalizing, where the model can easily get very high dice scores on the training data while failing to achieve good dice scores on new data. A simple approach to tackle this was to implement early stopping; meaning the network will save only the last model when the validation loss decreases and then stops after a couple of epochs if there was no further improvement,

however early stopping was not enough as the gap between validation and training was still present and needed to be decreased if we want the network to learn better (see Figure 3.2 ES). Another possible solution was to insert batch normalization and drop out layers of 0.3 rate after every conv layer in the network, where the batch-Norm layers will regularize the output of conv layers and the drop out layers will help the model to generalize. As we can see in Figure 3.2 for BND the gap between training and validation loss became smaller and hence the mean dice scores improved. Moreover, to help the learning process to converge faster and better (see Figure 3.2 IN), instance normalization of the conv layers outputs was implemented instead of the batchNorm layer.

As discussed in the proposal analysis section, a variant of the Unet architecture was also adopted in our work, which is the ResUnet. The residual blocks in the decoder and encoder helped the network to learn more (training loss goes lower) without risking too much overfitting. This mild overfitting was further reduced with a drop out layer of 0.1 rate inserted between conv layers in the residual blocks, and therefore having a smooth learning curve for the loss funtions as shown in Figure 3.2 ResUnet_D leading to a higher mean dice scores (check Table 3.1 above).

Other experiments have also been tested in this project, but they did not lead to a significant improvement in the dice scores. These experiments include analyzing the effect of the activation function in the learning process (Relu versus Leaky_Relu) where the difference was barely noticeable. The number of conv layers in the residual blocks was tested for two values n=2 and n=3 where n=2 conv layers per residual block performed better. The effect of the feature maps number was also tested for k=16 and k=32 where the training process appeared to be faster and led to better dice scores for k=16.

Figure 3.4 below shows some samples of the segmentation of our model along with the input and ground truth segmentation for all the scans in the validation set

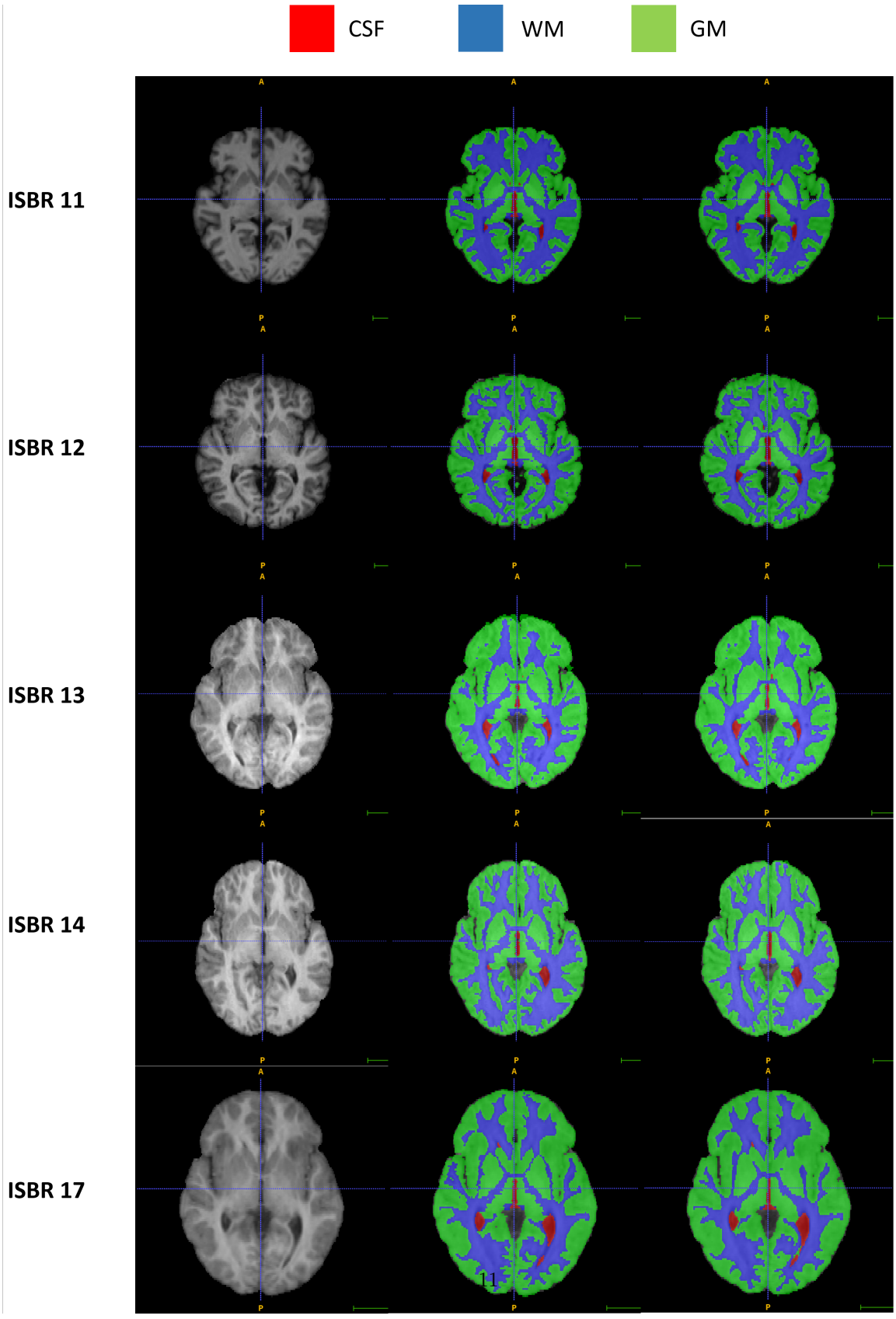





Figure 3.4: sample axial slices from the validation set showing the input image (left), ground truth segmentation (middle) and our segmentation (right)

## 3.3 Loss Function

The last parameter we worked on is the Loss Function. As mentioned we implemented also the Dice Loss in addition to the Cross Entropy. The next table shows the results of the best model (ResUnet) on the Validation set using the two losses.

| Experiment\Tissue Type | CSF | GM | WM |
|---|---|---|---|
| Cross Entropy | 92.35 | 95.04 | 94.86 |
| Dice Loss | 92.31 | 95.30 | 94.91 |

The table above shows the the Dice Loss performs really similarly to the cross Entropy loss.In our code we implemented these two losses in a modular way so by setting a flag to true or false we can use the Cross Entropy or Dice.

# 4. Conclusions and Project Management

During this Project we learned how to deal with a segmentation problem through the Deep Learning framework. We were able to understand, adapt and improve a given architecture to solve our task.

In our work the deep learning models seem to work better when inputs are not preprocessed (contrast enhanced), which was surprising as the dataset volumes were acquired using different scanners and have different intensity ranges. It seems that preprocessing cause losing some of the features that deep learning models can pick up and use it for segmentation. Unet Architecture and its variants proved to be powerful in brain tissue segmentation task.

Also in this work we tested the usefulness of normalization in deep learning, either for the input or in between convolutional layers. Different ways of normalization were applied, where instance normalization gave the best performance by enhancing the loss curve and improving the dice scores.

We developed this project in group of three, scheduling different meeting and giving different tasks both before the holidays and during holidays. We mainly used Google Colaboratory to develop our code and share our data on Google Drive.