ELEC6604 Neural Networks, fuzzy systems and genetic algorithms.

Assignment 3

Convolutional Neural Network for MRI Classification

GE YE 3035541510

Description of the task

I will use the MICCAI2012 dataset and the code implemented by Dr. Grantham Pang as a basis for the assignment, and the main objectives include investigating the segmentation results and other structures and parameters.

Database and setup of the problem

The dataset contains 15 manually segmented brain MRIs and 20 without segmentation. The challenge was to develop algorithms to segment the 20 MRIs into the 134 anatomical regions. Dr. Pang has offered us 15 patients for training contained in three files, and 20 patients for testing in four files.

I will not use all these data for training because it is too time-consuming. If the accuracy is good enough, 5 patients would be enough for training. I will not use the load_weights function when I start a new trial in order to examine the impacts of parameters independently.

Structure of the CNN

The original structure of the CNN given in Brebisson's [1] paper is rather complicated, including six 29*29 patches one 13*13*13 patch, and Dr. Grantham Pang has surely modified it to be more practical and trainable. The performance of the CNN model will be validated by the mean dice coefficient. Brebisson benchmarked his work against the multi-atlas methods of 2012 MICCAI challenge which has a mean dice coefficient of 0.725, and achieved a result of 0.74, while Dr. Pang's implementation achieved 0.6989. Considering the reduction of 2D patches, this result is truly satisfying. I will start from this implementation, and the model diagram generated from the Colab Notebook can be shown in Figure 1.

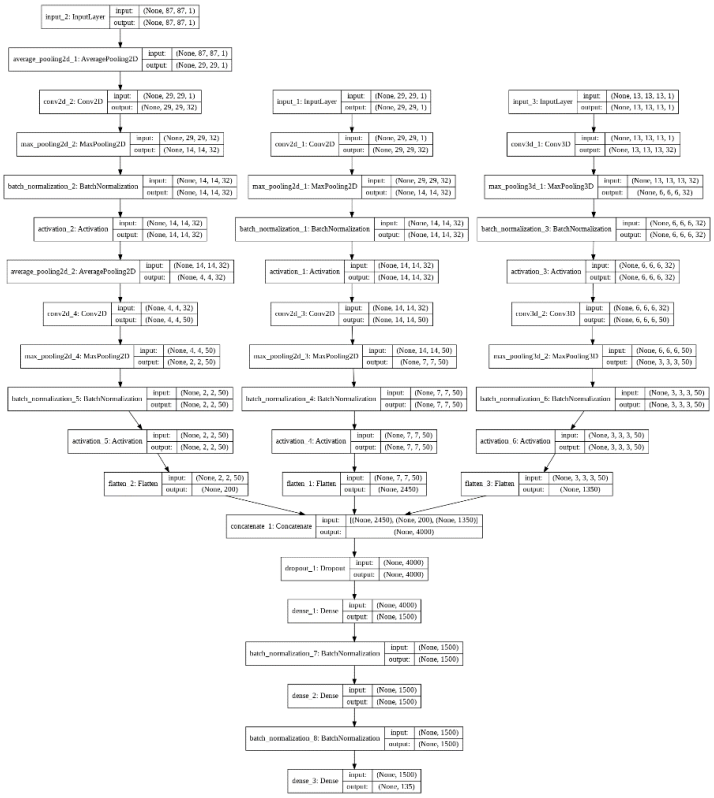

Figure 1 Original Model Diagram

In order to save the time of training in order to come up with more creative adjustments, I decided to use one set of data file, which consists of five patients. After first attempt, I got a dice coefficient of 0.6590, which is quite good, considering that I am using 1/5 of the time in the original version to achieve a similar training result.

In the next part of the assignment, we would vary the structure of the CNN and other parameters, including extension to 2D or 3D planes, the usage of Batch Normalization, filter size and number, activation function type, max pooling, dropout rate, etc.

Parameters for variation

Like has been mentioned in the last part, the parameters for variation can include including extension to 2D or 3D planes, the usage of Batch Normalization, filter size and number, activation function type, max pooling, dropout rate, etc.

We will start from the input first then the following structures, each time we change one set of parameters or one part of the structure only.

1. Vary the input size
2. Vary filter size and number
3. Have more layers
4. Add extension to more 2D planes
5. Try the use of two 3D patches.
6. Modify the use of Batch Normalization
7. Modify activation function type
8. Modify max pooling
9. Modify dropout rate

Process

Modification of input size

Firstly, I changed the input size from (x1=29*29 x2=87*87 cube=13*13*13) to (x1=29*29 x2=54*54 cube=20*20*20). The intention of doing this is to see whether the input size would have impacts on the training results and running time, and it does. By making such modifications, the dice coefficient decreases from 0.6590 to 0.6142, which is slightly lower than before, but the training and testing time increased, with the training time changing from around 770s to 1200s, and the average testing time of each patient increases from 480s to 620s. Getting back to the modification, although we only changed the input size, the multi-layers' max-pooling, batch normalization, activation, convolution has not eliminated its' effects on the training, and this reminds me that we should be careful in choosing the inputs.

I will continue to change the input size to (x1=29*29 x2=100*100 cube=13*13*13) and to (x1=29*29 x2=87*87 cube=5*5*5) to see the impacts of changing different patches. In the first situation, the training time is around 727s, while the test time is around 430s on average. The dice coefficient is 0.6575, which is close to the one we started. From this trial, we can see that tiny changes in the second patch might not have a great impact, and we would test the third patch's influence by decreasing 3D sizes. If

we change the input size to (x1=29*29 x2=87*87 cube=5*5*5), the training time is 580s, while the test time is around 400s per patient. The final dice coefficient is 0.6243, which is quite good, considering that we are having a much smaller cube. It seems that no matter how we change the input size, the dice coefficient would not improve greatly in the ways we have altered the input. Maybe changing the cube size would have some improvements but it seems that (x1=29*29 x2=87*87 cube=13*13*13) is quite a good input size and I will continue to use this setting in the following trials.

Modification of filter size and number

In all the above trials, we apply 32 3*3 or 3*3*3 filters in the first layer, and 50 3*3 or 3*3*3 filters in the second layer, and I am going to change the filter size and number in respectively to evaluate their impacts. Firstly, I change the number of 3*3 filters to 20 in the first layer. The training takes around 700s, and testing costs 446s, while the dice coefficient is 0.6529. Next, I increase the number of 3*3 filters to 80 in the second layer, and the result shows that training takes 762s, testing takes 420s on average, while the dice coefficient decrease to 0.6293. To be honest, the modification of filter numbers does not seem to have a great impact on the training results, maybe this is because that multi-layers of normalization and convolution have cut down the impacts.

I try to change the filter size to see whether this would have greater impacts, and I find that increasing the filter size in the second layer has more impacts on the training and testing time, but they both decrease the dice coefficient to some extent. Increasing the filter size to 5*5 or 5*5*5 would have the training time of 726s, and testing time of 440s, while similar changes in the second layer would have a training time of 911s and testing time of 540s. The dice coefficient slightly decreases to around 0.64 in both cases. Maybe this is because that a bigger filter loses some important information in the process.

Modification of layers

I add a third layer to the model, and this layer is not as complete as the previous two layers because of the rather small input sizes entering this third layer. The layer's structure can be shown in Figure 2.

```
#Layer 3
x1_3 = Conv2D(50, (3, 3), padding='same',use_bias=False)(x1_2)
x1_3 = BatchNormalization()(x1_3)
x1_3 = Activation("relu")(x1_3)

x2_3 = Conv2D(50, (3, 3), padding='same',use_bias=False)(x2_2)
x2_3 = BatchNormalization()(x2_3)
x2_3 = Activation("relu")(x2_3)

cube_3 = Conv3D(50, (3, 3, 3), padding='same',use_bias=False)(cube_2)
cube_3 = BatchNormalization()(cube_3)
cube_3 = Activation("relu")(cube_3)
```

Figure 2 Additional layer

I assume this additional layer would increase the time for training and testing, but I am not quite sure on its effects on the dice coefficient. The training time takes around 807s, while the testing time is 516s. The dice coefficient slightly decreases to 0.6371. I decide to examine the details of the dice coefficient to see whether this is by chance. After I examine the previous trial's dice coefficient, I find that among the 134 regions, the

average denominator is around 17500, so this somehow eliminates the randomness in the average dice coefficient.

```
#Layer 4

x1_4 = BatchNormalization()(x1_3)
x1_4 = Activation("relu")(x1_4)


x2_4 = BatchNormalization()(x2_3)
x2_4 = Activation("relu")(x2_4)


cube_4 = BatchNormalization()(cube_3)
cube_4 = Activation("relu")(cube_4)
```

Figure 3 Another additional layer

I decide to add another layer, the fourth layer. The fourth layer is truly simple, as is shown in Figure 3. It turns out adding additional layers, with other parameters unchanged, might not improve the results. Maybe this means that too much manipulation with the input data might not do good to the results.

More 2D planes

I add one more 2D plane to the model, being the fourth patch, the medium patch with an input size of 54*54. I will expand this size if training time in acceptable. The result shows that the training time increases greatly, with the test time slightly. The dice coefficient doesn't improve. I enlarge this medium patch to 100*100 to see whether there is a chance. The training time greatly increases to 1892s, while the test time increases to 792s. However, the dice coefficient is 0.6495, remaining below the original setting. I think that maybe the original size is the optimal setting, but we will keep exploring different opportunities.

Two 3D patches

In order to generate two 3D patches as inputs, I replace the medium and large patch with a 3D patch of 26*26*26. I am doing this instead of adding a new patch because in previous trials I already experience a crash, and adding more 3D patches would consume the RAM more drastically. The result shows that a 2D plane + two 3D patches is not a wise choice, since it not only increases the runtime greatly but also decreases the dice coefficient. Maybe it will be useful if we can handle the crash problem and keep the 2D plane.

Modify Batch Normalization

Firstly, I get rid of all the batch normalization process in the first layer. It is astonishing that removing the batch normalization in the first layer would decrease the dice coefficient to 0.5885, I continue to remove the batch normalization in the second layer to validate this phenomenon. Removing batch normalization in both layers would increase the dice coefficient to 0.6086, and removing the batch normalization in second layer would have a dice coefficient of 0.6332. I conclude that removing BN in the first layer would have a much greater impact than in the second layer. What if I add additional steps of BN in the first layer? I try adding Batch Normalization after the first step in each patch. The dice coefficient slightly beats the one in the original setting, at

the cost of additional training and test time. Batch Normalization is generally good for training, and we should consider them in future work.

```
x1_1 = Conv2D(32, (3, 3), padding='same',use_bias=False)(x1)
x1_1 = BatchNormalization()(x1_1)
x1_1 = MaxPooling2D()(x1_1)
x1_1 = BatchNormalization()(x1_1)
x1_1 = Activation("relu")(x1_1)

x2_1 = AveragePooling2D((3,3),strides=3)(x2)
x2_1 = BatchNormalization()(x2_1)
x2_1 = Conv2D(32, (3, 3), padding='same',use_bias=False)(x2_1)
x2_1 = MaxPooling2D()(x2_1)
x2_1 = BatchNormalization()(x2_1)
x2_1 = Activation("relu")(x2_1)

cube_1 = Conv3D(32, (3, 3, 3), padding='same',use_bias=False)(cube)
cube_1 = BatchNormalization()(cube_1)
cube_1 = MaxPooling3D()(cube_1)
cube_1 = BatchNormalization()(cube_1)
cube_1 = Activation("relu")(cube_1)
```

Figure 4 more Batch Normalization

Modify Activation function type

Previously, we used 'relu' as the activation function in all the patches and in the first two dense layers, and 'softmax' in the output. I first reverse their position and then revise the activation function to be 'relu' or 'softmax' respectively in the patches and the dense layers and output. Reversing the 'relu' and 'softmax' would decrease the dice coefficient so dramatically to 0.1030, while replacing the 'relu' in the dense layers by 'softmax' would not make a big difference, resulting in 0.6440. There is a saying that 'relu' could result in dead neurons, which means that some gradients can be fragile during training and can die. It can cause a weight update which will makes it never activate on any data point again. [2] With this finding, I replace the 'relu' in the layers with 'tanh', hoping to reserve more information on the neurons. However, the dice coefficient turns out to be 0.6432. Maybe this means that 'relu' is good enough for the current training, and changing the activation function would not improve it greatly.

Modify max pooling

Firstly, I delete all the max pooling in the first layer, trying to figure out what kind of impacts this would have, and it turns out that removing Max Pooling in the first layer would increase the training and test time but have little influence on the dice coefficient. I continue to add more max pooling in the first layer, and it decreases runtime and dice coefficient at the same time.

Modify dropout rate

The dropout rate is an effective regularization method to reduce overfitting and generalization error in deep neural networks.[3] Usually, a probability of 0.5 is for retaining the output of each node in a hidden layer and a value close to 1.0 is for retaining inputs from the visible layer. In our experiment, I vary the dropout rate value to 0.2 and 0.8 to see the difference between 0.5. According to Brownlee, a good value for dropout in a hidden layer is between 0.5 and 0.8. Input layers use a larger dropout rate, such as of 0.8. It turns out that the dropout rate of 0.5 is the most suitable for our case, and the dropout rate of 0.8 and 0.2 would both decrease the dice coefficient by several percent.

Results

Table 1 Summary of Results

	Training time	Average testing time	Dice coefficient
Original	772s	480s	0.6590
Input size 1*	1200s	620s	0.6142
Input size 2*	727s	430s	0.6575
Input size 3*	580s	400s	0.6243
Filter number 1*	700s	446s	0.6529
Filter number 2*	762s	420s	0.6293
Filter size 1*	726s	440s	0.6414
Filter size 2*	911s	540s	0.6433
Add one layer	807s	516s	0.6371
Add one more layer	836s	510s	0.6467
Add more 2D plane*	1026s	518s	0.6481
Add bigger 2D plane*	1892s	792s	0.6495
Two 3D patch*	1933s	966s	0.6100
No BN in first layer	646s	363s	0.5885
No BN in second layer	658s	374s	0.6332
No BN in both layers	611s	355s	0.6086
More BN in the first layer	786s	434s	0.6595
Swap activation	730s	405s	0.1030
Relu+softmax	681s	397s	0.6440
Replace Relu with tanh	674s	402s	0.6432
No MP in first layer	1603s	666s	0.6584
More MP	510s	344s	0.5827

Dropout rate = 0.8	678s	378s	0.6134
Dropout rate = 0.2	693s	385s	0.6341

Explanation:

*Input size 1: x1=29*29 x2=54*54 cube=20*20*20*

*Input size 2: x1=29*29 x2=100*100 cube=13*13*13*

*Input size 3: x1=29*29 x2=87*87 cube=5*5*5*

*Filter number 1: 20 3*3 or 3*3*3 in the first layer*

*Filter number 2: 80 3*3 or 3*3*3 in the second layer*

*Filter size 1: 5*5 or 5*5*5 in the first layer*

*Filter size 2: 5*5 or 5*5*5 in the second layer*

*Add more 2D plane: x3=54*54*

*Add bigger 2D plane: x3=100*100*

*Two 3D patch: cube2=26*26*26*

Discussion of the results and possible improvements

In my experiments, I vary different parameters such as input sizes, filter numbers and sizes, layer numbers and layer sizes, batch normalization, activation functions, max pooling and dropout rate. It turns out some factors would have a great impact on the training results, while others don't have big influences. Generally, the setting implemented by Dr. G. Pang is very effective and efficient, since it simplified the model implemented by Brebisson and achieved quite good training results. In my experiments, I revise different parameters trying to beat the original training result, and find that most of the modifications would not improve the dice coefficient, including changing the input sizes and filter numbers or sizes, whether enlarging or shrinking, some would even increase the training and test time greatly, including adding more 2D planes or 3D patches. More batch normalization in the first layer would slightly improve the dice coefficient, while swapping the activation 'relu' and 'softmax' would dramatically decrease the dice coefficient. Removing Max Pooling in the first layer would not influence the dice coefficient, but would increase the training and test time significantly. Also, adding extra Max Pooling would decrease training and test time and dice coefficient at the same time. The dropout rate of 0.5 is quite suitable for our current experiment setting.

However, I believe the dice coefficient is not a perfectly reliable evaluation on the training results, due to the randomness of calculating the coefficient in different sections. If possible, we can document the accuracy for each patient and analyse the data with more advanced techniques, or we can repeat each setting several times and calculate the mean value of the dice coefficient.

References

[1] A. de Brebisson and G. Montana, 'Deep neural networks for anatomical brain segmentation', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2015, pp. 20–28.

[2] A. S. Walia, 'Activation functions and it's types-Which is better?', *Towards Data Science*, 29-May-2017. [Online]. Available: https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f. [Accessed: 08-Apr-2019].

[3] J. Brownlee, 'A Gentle Introduction to Dropout for Regularizing Deep Neural Networks', *Machine Learning Mastery*, 02-Dec-2018. .

Table of Summary

Classification task	MRI Brain segmentation
Objective of investigation	investigate the segmentation results with other structures or parameters
Image size, Color or grayscale	35 T1-weighted brain MRIs divided into 15 training and 20 testing data.
Reference/source of the images	MICCAI2012 dataset
Number of classes	134
Number of training images	15 training patients
Number of testing images	20 testing patients
Structure of CNN	As shown in Figure 1
Parameters of variation	input size, filter size and number, layers, 2D planes, 3D patches, Batch Normalization, activation function type, max pooling, dropout rate
Training time	500s-1600s, depending on the structure and parameters
Testing accuracy	Normally the dice coefficient is around 0.65
Software platform	Tensorflow/Keras
Hardware platform	Google Colab