CS 2470 Final Project: Detection of ELVOs with 3D Convolutional Neural Networks

Victòria Gras Andreu Ashley Weber

Abstract

Strokes due to emergent large vessel occlusions (ELVOs) are the most common type of stroke, often leading to significant disability. This paper studies the use of convolutional neural networks on 3D images of CT scans to detect the presence of ELVOs.

1 Introduction

In recent years the use of machine learning algorithms for image recognition tasks has seen numerous applications in medical diagnosis. Medical images are a crucial part of a patient electronic health record and are currently analyzed by human radiologists. Unfortunately, radiologists are limited by speed, fatigue and experience. It takes years, and a big financial investment, to train a qualified radiologist. Therefore, it is ideal for medical image analysis to be carried out by machine learning algorithms.

Digital medical images arise in many different tests, to name a few: ultrasound, X-ray, computed tomography (CT) scans, and magnetic resonance imaging (MRI) scans. Each test can examine different organs (sometimes multiple organs) and generates different types and amount of data. In this paper we'll be focusing on images generated by CT scans of the brain. This test generates a 3-dimensional image of the brain.

Some of the most popular deep learning algorithms involved in the study of medical images are 2-dimensional and 3-dimensional convolutional neural networks (CNNs). Since CNNs maintain local spatial relationships while performing dimensionality reduction they are the top choice to perform image recognition. While CNNs and Recurrent Neural Networks (RNNs) have proven to give good results they are examples of supervised machine learning algorithms, which means that they require a significant amount of labeled training data which is usually hard to find. Here we trained and tested our model with approximately 1200 labeled images provided by the Rhode Island (RI) Hospital where 542 of them were positives.

In this paper we trained a model to detect the presence of an emergent large vessel occlusion (ELVO) given the CT scan of the patient's brain. Currently strokes are the fifth leading cause of death in the United States and there's about 1 million new strokes every year in the US alone. It is also a leading cause of disabilities. When a stroke is caused by a blockage of a large artery in the brain, direct removal of the clot through surgery is the best treatment known, which has been proven to be very successful. With a fast diagnosis and treatment, the patient's chance of survival (without disabilities) increases exponentially. Moreover, the recovery process is also much shorter. Therefore, being able to analyze images effectively through deep learning is vital for saving lives and reducing medical costs. It's a win-win situation for both patients and heath insurances!

2 Related Work

Deep learning is broadly used to detect anomalies in the brain and other organs. A couple of references that we found particularly interesting and inspirational were [1] and [4]. In [1], Julian de Wit explains how he trained a model to predict the development of lung cancer in a patient given a set of 3D CT images. His solution won the second place for the 2017 national data science bowl. In [4], the authors describe a model similar to the one we've implemented to classify brain hemorrhage.

Our data has been used in the past to detect ELVOs. A group of researchers at Brown University created a model using 2D CNNs on slices of the brain and ResNet-50 pre-trained on ImageNet data. They achieved a validation accuracy of 86.3% and AUC of 0.917, see [2].

3 Medical Background: understanding the data

In order to understand the data, we need to introduce first some medical terminology. An emergent large vessel occlusion (ELVO) is a blockage of blood flow to the brain which often results in a stroke (see Figure 1). ELVOS appear in different areas of the brain, not necessarily in only one location. As mentioned in the Introduction, the best way of detecting them is through a CT scan, which creates a 3D image of the brain.

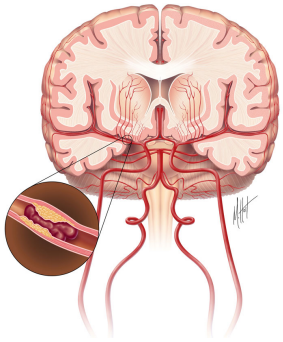

Figure 1: ELVO, blockage of blood

In order to detect ELVOs, human radiologists often look for asymmetries in the brain. In a healthy brain, the blood circulates symmetrically in both hemispheres. If a blockage has occurred, the blood flow in half of the brain is compromised and therefore the scans show an asymmetry. Figure 2 shows three CT scans: one with no ELVO and two with ELVOs in opposite sides of the brain.

4 Preprocessing and architecture

4.1 Preprocessing

The raw CT scans had already been converted to 3D numpy arrays by Brown researchers. A dictionary was created where each key represented the anonymous ID of the patient and the values were the numpy arrays. Since the images are CT scans, each entry in the numpy array is a value in the Hounsfield scale, a quantitive scale for describing radio intensity.

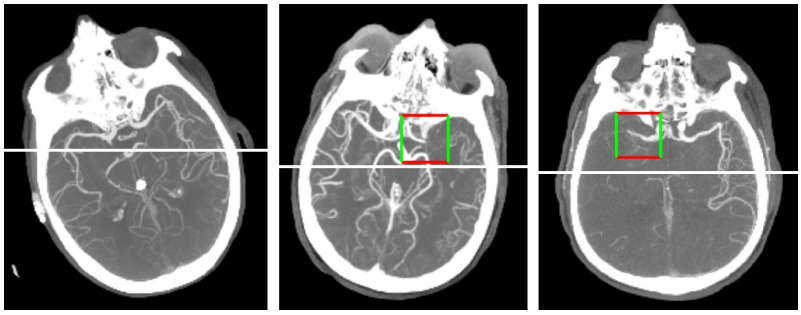

Figure 2: CT scans of brains with ELVOs. The left hand side image has no ELVO

The labels were located in a two column Pandas DataFrame. The first column contained the patient anonymous ID and the second was either 1.0 or 0.0, representing the presence or not of an ELVO.

The first step of the preprocessing was removing duplicates (using a function provided to us by the Brown research group) and images with no labels. At this point, the number of images was 1197.

A few problems we faced was that the image shape and location of the brain within the image was not consistent throughout the dataset. Since we decided to feed into the model the full 3D image of the brain as in [4], we first cropped the images to make them all the same size. This also reduced the dimensionality. We cropped the images to shape $(190, 200, 200)$ using a modified cropping function that the Brown research group provided. Images whose shape were smaller than $(190, 200, 200)$ were removed, this amounted to removing 21 images.

Moreover, since the values of the pixels we were interested in fell into the interval $[-200, 400]$ according to the Hounsfield scale, we set all the values greater than 400 equal to 400 and similarly changed all values less than -200 in order to reduce noise. Finally we normalized the data to have range $[0, 1]$.

4.2 Architecture

We created a network with three 3D CNN layers, three max pooling layers, and two fully connected layers with dropout between them, see Figure 3. We inputed the data into the model with a batch size of 5. The first convolution consisted of eight filters, a kernel size of $[10, 10, 10]$, a stride of 5 and SAME padding. The second convolution had sixteen filters, a kernel size of $[3, 3, 3]$, a stride of 1 and SAME padding. The third convolution had thirty-two filters, a kernel size of $[3, 3, 3]$, a stride of 1 and SAME padding. In all convolutions we used a bias and relu activation function. The max pooling layers had a pool size of $[2, 2, 2]$ and a stride of 2. Since we had 2 fully connected layers, we had one hidden layer. This hidden layer had size 1500. We also used relu as an activation function between layers. We used a training dropout of 0.5. Finally, the last fully connected layer had a final shape of $[5, 2]$, where 5 is the batch size. We used [3] as a reference for the coding of the model.

We used the softmax cross entropy loss function with logits and Adam optimizer with learning rate of 10^{-4}.

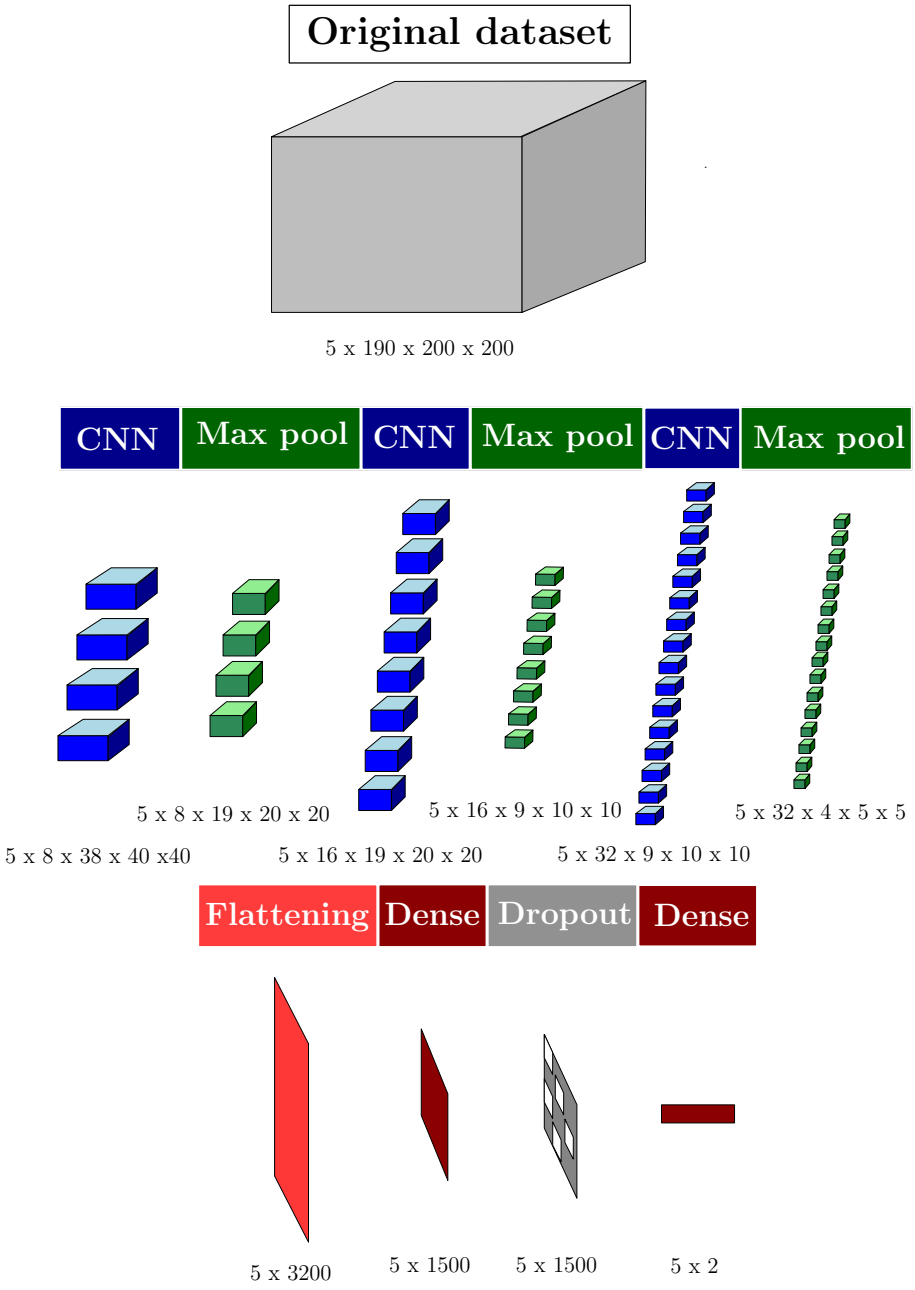

Figure 3: Architecture

5 Results

We used three different metrics to evaluate our results: accuracy, precision and recall. The latter ones are more commonly used in medical analysis. As a reminder, precision and recall can be computed as follows:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} \ + \ \text{False Positives}}, \quad \text{Recall} = \frac{\text{True Positives}}{\text{True Positives} \ + \ \text{False Negatives}}.$$

Our architecture achieved a peak accuracy of 67.5%, a precision of 69% and a recall

of 68% on the test dataset. However the model overfits with a training accuracy ranging from 86% to 95%. We rerun the model with these hyperparameters four times and got a test accuracy ranging from 62% to 67.5%.

6 Discussion

Overall we are quite happy with the results obtained in this project considering all the issues we had to face that were out of our control. We are aware that in the real world our results wouldn't be great, especially due to the overfitting problem. In what follows we'll elaborate on some of those issues as well as how those shaped our decisions in the preprocessing of the data and the design of the final model.

Preprocessing was one of the most important parts of our project. Based on the positive results from [4], we decided to feed into the model the 3D images of the brain. Our first attempt consisted in using the cropping dimensions suggested by the Brown research group, $(120, 200, 200)$. With this cropping we got accuracies of 59-60%. While studying the data we discovered that we were eliminating the location of some ELVOs. To fix this issue, we increased the dimensions of our images to $(190, 200, 200)$. With this change, our accuracy increased by 2-3%.

The choice of an architecture was inspired by [4] and [3]. CNNs seemed like the most natural choice to start training our model since they maintain local spatial relationships while performing dimensionality reduction. Since our images were so large, we decided to start with big kernels and strides to bring down the dimensions by five. We decided to use the remaining parameters as in [3].

To train our model we chose a batch size of 5 and ran it for 10 epochs. We concluded that a greater batch size would be computationally too expensive and more epochs wouldn't be helpful since our dataset is rather small. By monitoring the training and testing accuracies after each epoch we discovered that the model would reach a train accuracy of over 90% and a test accuracy of around 60% in 6 epochs. We kept the number of epochs at 10 because we observed some fluctuations and consistently got the highest test accuracy after epoch 10. We used 80% of the data to train and 20% to test.

In order to fix the overfitting of our model and improve the accuracy we tried different strategies:

- *Decreasing the number of parameters*: We decreased the number of filters from 16-32-64 to 8-16-32 and the hidden size from 2000 to 1500. We didn't observe any major changes in the accuracies with this modifications.

- *Decrease the number of epochs*: We started training our model with 15 epochs and decreased it to 10. We didn't observe any major changes except for the obvious one: our model finished training faster. With 10 epochs, training took about 4 hours.

- *Dropout*: We tried dropout with different rates ranging from 40-60%. Dropping 60% made the precision and recall lower and barely changed the accuracy, and didn't help much with overfitting, so we decided to drop 50%.

- *Batch normalization*: We added batch normalization at different locations of the architecture. The most successful one was adding it after the last max pooling layer. With batch normalization included we observed that the train accuracy wouldn't increase as fast after each epoch, so it took longer for the model to overfit. We

observed that keeping the is training set to true for testing would give us significantly better results than switching it to false. After some online research we discovered that it is a common issue when using small batches like ours and one way of fixing it is keeping the is training true for both training and testing. However the model didn't perform as well with new data than it did without the batch normalization and it was still overfitting so we ended up removing it.

- *Different learning rates*: We tried learning rates of 10^{-4} and 10^{-5}. We decided to use small learning rates because we're using a small batch size. The learning rate of 10^{-4} gave us significantly better results.

- *Augmentation*: Since adding data is one of the best ways to fix overfitting and improve accuracy, we implemented a rotation and translation function. Due to memory issues we were unable to apply those transformations to all of the training data. To add an extra 500 images we had to continuously delete variables we were no longer using. We created those 500 images by applying five translations to 100 different images and shuffling them with the original data. The whole process made the model slower since translating 500 images took about an hour and every time we wanted to rerun the model with a different dataset we had to re-download the data. The extra time was worth it since just adding those few images already improve the accuracy by 2-3%.

Provided that we have more memory, a faster platform, and more time, these are some approaches we would like to try:

- *Augmentation*: With more memory we would like to add more translations and rotations of the data.

- *Transfer Learning*: Training the model with pre-trained weights has been proven extremely successful when dealing with not very large datasets like ours. All of our limiting factors contributed in our decision to not implement transfer learning.

- *Recurrent Neural Networks*: There have been some papers that have shown combining CNNs and RNNs give better results in medical imaging classification problems. With more time, it would be nice to explore different ways of adding RNNs to our current model.

Joining an established researched group has shown us the importance of reproducibility of the code and the accessibility of platforms used on a day-to-day basis. Even though we spent over a month getting most of the set up, due to time restrictions we were not able to get what was necessary to run our model on a more powerful platform and had to share the GPU with another group making it so that only one person from each group could run their models at a time, constantly making sure no extra Jupyter notebooks were left open to avoid memory issues and dying kernels.

References

[1] J. De Wit, (2017, April 14) *2nd place solution for the 2017 national data-science bowl* [Web log post]. Retrieved from `http://juliandewit.github.io/kaggle-ndsb2017/`

[2] M. Dong, A. Kim, S. Subzwari, H. Triedman, A. Wang, L. Zhu, U. Centintemel, M. Stib, A. Yao, R. McTaggart, C. Eickhoff, G. Baird and J. Boxerman, *Deep Learning for ELVO Stroke Detection*, Submitted to Machine Intelligence Conference, (2018), `https://openreview.net/forum?id=rkldWGAY_m`

[3] Jian, (2017) *3D CNN with Tensorflow*. Retrieved from `https://www.kaggle.com/shijianjian/3d-cnn-with-tensorflow`

[4] K. Jnawali, K. and M. R. Arbabshirani, and N. Rao and A. A. Patel, *Deep 3D convolution neural network for CT brain hemorrhage classification*, Proc. SPIE 10575, Medical Imaging 2018: Computer-Aided Diagnosis, 105751C (27 February 2018); doi: 10.1117/12.2293725