Brain CT Image Hemorrhage Segmentation

Anurag Arasan John Ciolfi Carson Cook Tatiana Ediger Nathaniel Fernandes
Department of Mathematics, Northeastern University

May 1, 2022

Abstract

Our goal with this project is to train various classification models to determine whether a given brain CT scan contains evidence of any one of five types of hemorrhages, multiple different types of hemorrhages, or no hemorrhages. The provided data set has labeled images of CT scans which we can use to train and test our models. The scans are formatted such that a model will receive four image files per CT scan – each a different rendering type of the same image (different rendering types may show various hemorrhages more clearly). The model should then return whether the images show evidence of a hemorrhage (or multiple hemorrhages) or not. If there is evidence of a hemorrhage (or hemorrhages), it should return the type of hemorrhage it believes exists (or the types, if multiple).

1 Introduction

Zeta Surgical, a medical imaging company, has provided our team with a dataset consisting of various brain CT scan slices. Each of these imaged CT scans contains some form of hemorrhage (bleeding) within it. The different types of hemorrhages include intraparenchymal, intraventricular, subarachnoid, subdural, and epidural. There is also an additional category for images with multiple kinds of hemorrhages. Depending on the type of hemorrhage(s) the patient has, different strategies will be used for treatment. Therefore, it is essential that the detected hemorrhage is labeled accurately. Some images in the set are labeled, and some are not. Using machine learning and computer vision, our team will perform the regression and the segmentation of these CT scan images.

2 Related work

Prior to building our models, we researched different papers describing different techniques and approaches towards classifying hemorrhages.

In *Diagnosis classification of brain hemorrhage*, Davis, et al. propose a methodology for detecting and classifying different hemorrhages. Images are converted to grayscale, re-sized, and enhanced via techniques like smoothing to reduce noise and streamline edge detection. The Watershed algorithm is utilized to ”classify distinct regions in the system using watershed lines.” The regions serve as input to a ”Grey Level Co-occurrence Matrix (GLCM)” to extract different features for classifiers.

In *Brain Hemorrhage Classification in CT Scan Images Using Minimalist Machine Learning*, Ramirez, et al., multiple image processing techniques were employed to ”select the most relevant attributes that make up a dataset.” Contrast Limited Adaptive Histogram Equalization (CLAHE) functions by increasing gray intensity to minimize noise. Mean Filter uses a mask, or kernel, to traverse the image and replace a pixel by the mean value of neighboring values. The techniques aim to sharpen edges to increase model accuracy.

3 Summary

3.1 Data Processing

For efficient and consistent training and testing, we built a convenience method to extract feature data for our models from the image files. This tool traverses through labelled folders of images while extracting and aggregating each image's pixel data to one layer (grayscale). Depending on the model, we used various degrees of sub-sampling in the form of image re-scaling.

We tested two main techniques for sub-sampling: naive slicing and interpolation. The naive slicing technique involved taking every n^{th} row and column from the data where $\frac{512}{n}$x$\frac{512}{n}$px is the desired sub-sampled size. For our interpolation approach, we used a resize method from the Python library ”CV2”. This method implements bi-linear interpolation for down-scaling or up-scaling images. The benefit of this interpolation approach is that new pixel values are calculated using the average pixels values surrounding them in the original image. In this way we retain some sort of ”context” from the original image, which then can be passed to our models. Results from both can be seen below:

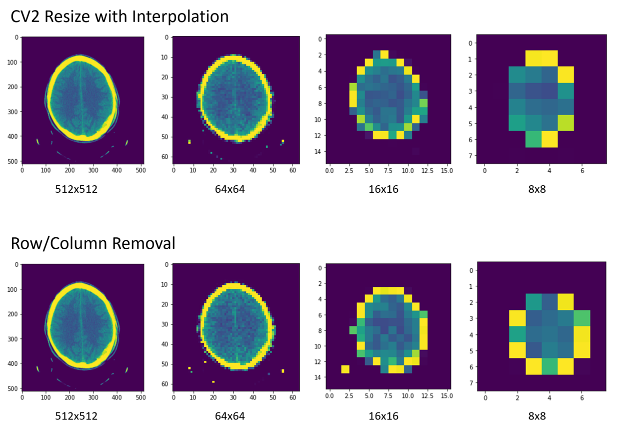


Figure 1: Comparison of Sub-sampling Methods

Images that were not 512x512px were dropped, though future work could try to salvage these for training and bias purposes by padding the image with pixels similar in color to the background.

The tool will return a 2-D array where each row represents a window scan and each column represents a pixel (or subsampled pixel) intensity that our models can use for training. The labels for each row are also returned as a 1-D array with it's corresponding label: 'normal', 'epidural', 'intraparenchymal', 'intraventricular', 'subarachnoid', 'subdural', or 'multi'.

Once our data was in this format, we began to build and train our models. For this milestone we focused on Logistic Regression and building a Convolutional Neural Network.

3.2 Logistic Regression

We built a model for logistic regression to then classify the various CT scans as either normal, one of the various types of hemorrhages, or a mix of multiple hemorrhages. For training data, we down sampled the images to 64x64 pixels and used 1,000 from each of the categories. For the testing the model, we used 100 down sampled images from each of the categories. We found after preliminary test runs, that the brain bone window provided the best results, so we made the decision to only use this window for the training and testing of the model. To train the model, we tested various solvers from the scikit learn library, particularly the solvers that support multi classification regression. These solvers included SAG, SAGA, LBFGS, and Newton-CG. Each of these solvers used L2 penalties, except for SAGA, which uses a combination of L1 and L2 penalties. As expected, the results from the logarithmic and linear regression models were not very accurate. This is discussed further in the results section. Due to the large image size, we only used 1000 images that were then downsized to train our model. This downsizing and low training sample size could be the reason for the low accuracy of the regression techniques, it may be beneficial to use external computing power to help train the model in the future. However, we think that it is more likely that the model itself is intrinsically less suitable for this type of classification than other methods that utilize neural networks.

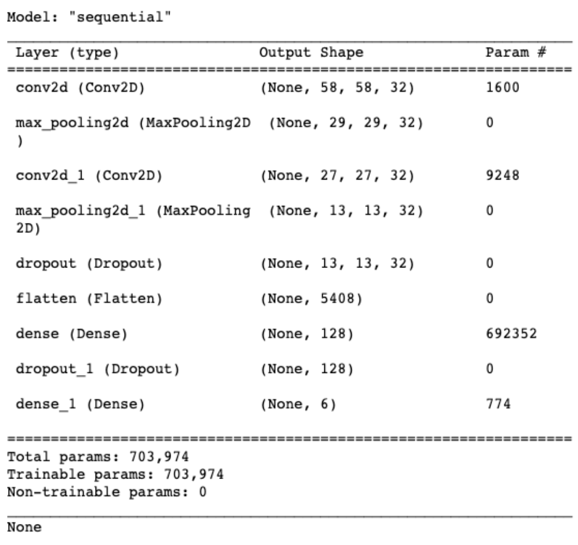
3.3 Convolutional Neural Net

We used TensorFlow to build a convolutional neural network.

We downsampled the images to 64x64px using the methods outlined in section 3.1, and we selected up to 5000 images from each hemorrhage type. We ended up with 22340 training images and 5336 test images (using an 80/20 split). For our model itself, we used a 2D convolutional neural network with two convolutional layers and two pooling layers. For pooling, we used max pooling and for our best model, we used ReLu activation functions. We also included two dense layers, a flatten layer, and in order to combat overfitting, we used dropout as a regularization technique. Dropout helps to prevent overfitting, because it ensures that the model uses all neurons to build the model, since any one neuron could be dropped, and does not just rely on one specific configuration.

As part of our processing building this model, we tweaked various parameters to find those which yielded the best results. We tried using various activation functions, including ReLu, sigmoid, and linear functions. We attempted changing the number of images we pulled from each hemorrhage type, ranging from 500 to 5000. We tested out different Pooling methods (average, min), and tested out different numbers of epochs. And finally, we used different windows (brain-bone, brain, max-contrast and subdural).

The figure below shows our final model



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 58, 58, 32)        1600

 max_pooling2d (MaxPooling2D  (None, 29, 29, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 27, 27, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 dropout (Dropout)           (None, 13, 13, 32)        0

 flatten (Flatten)           (None, 5408)              0

 dense (Dense)               (None, 128)               692352

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 6)                 774

=================================================================
Total params: 703,974
Trainable params: 703,974
Non-trainable params: 0
_________________________________________________________________
None
```

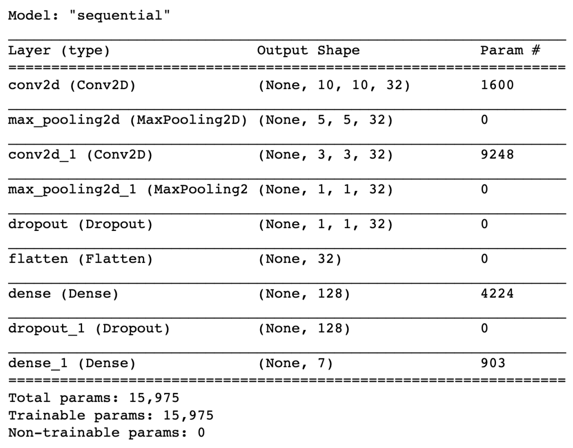
Figure 2: CNN Final Model Layers

4 Preliminary Results

Our first pass building models for this project yielded mixed results. We first started off by building a model for both linear and logistic regressions to then classify the various CT scans as either normal or one of the various types of hemorrhages. As expected, the results from the logarithmic and linear regression models were not very accurate. This is discussed further in the results section. Due to the large image size, we were only able to use 1000 images that were then downsized to train our model. This downsizing and low training sample size could be the reason for the low accuracy of the regression techniques, it may be beneficial to use external computing power to help train the model in the future. However, we think that it is more likely that the model itself is intrinsically less suitable for this type of classification than other methods that utilize neural networks.

We also began to build a Convolutional Neural Network to try to classify each of our images into each of the five types of hemorrhages or normal. At this point, our accuracy was not quite at the level we were hoping for. Our goal for the next milestone was to tweak our model and see what we could do to improve this accuracy. We want to tried different arrangements of layers, different loss/optimization functions, different activation functions, and potentially we would like to try a deep neural net in addition to a CNN model. Originally, when we were first exploring our data, we tried to just classify each image into ’hemorrhage’ or ’no hemorrhage.’ This produced great results. It was once we tried to get more granular and detect the type of hemorrhage where it got trickier. Another thing we tried was training our model on more data, for efficiency, we trained with only 1000 data points from each hemorrhage dataset, and only used ’brain_bone_window.’

In the figures below, we can see that our original model was trained on only 15,975 total parameters, and it’s general structure included two convolutional layers,



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 10, 10, 32)        1600
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 5, 5, 32)          0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 3, 3, 32)          9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 1, 1, 32)          0
_________________________________________________________________
dropout (Dropout)            (None, 1, 1, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 128)               4224
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 903
=================================================================
Total params: 15,975
Trainable params: 15,975
Non-trainable params: 0
```

Figure 3: CNN Preliminary Model Layers

At that point, the model accuracy was fairly low, and for the next stage of this project our goal was to tweak the model to improve it. The preliminary model accuracy and model loss are shown in Figure 3.

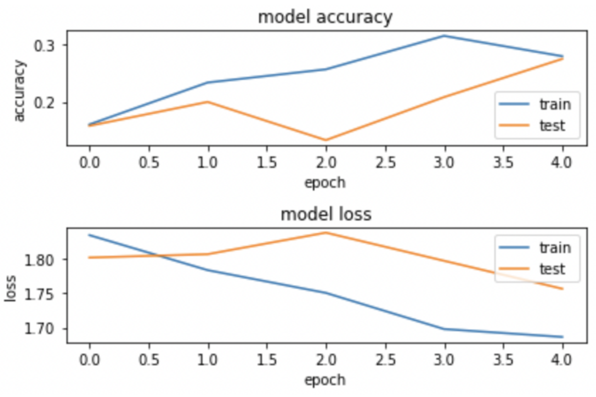


Figure 4: Preliminary CNN Model Accuracy and Loss

5 Results

5.1 Logistic Regression

Using logistic regression from the scikit-learn library, we created a logistic regression based on the hemorrhage types except for multi, subdural, and subarachnoid, forming the regression over 1000 iterations. We split the data into training and testing data and evaluated the accuracy using the residual sum of squares. The current regression model is reasonably accurate, producing far better results than the linear regression. A confusion matrix further supported that the regression was mostly accurate. The next step for the logistic regression would be to include more hemorrhage types and again check the accuracy, hopefully demonstrating that the regression is not overfitted to the data.

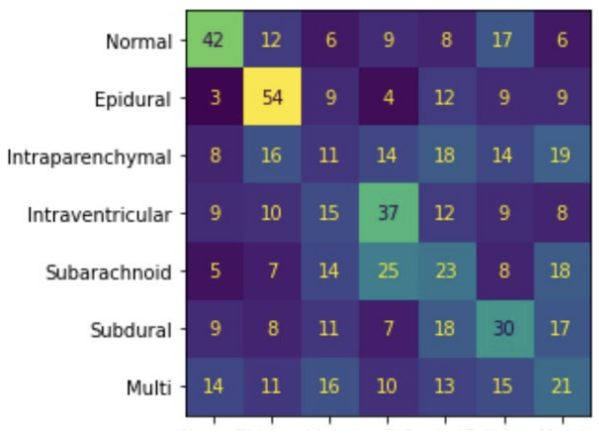


Figure 5: SAG

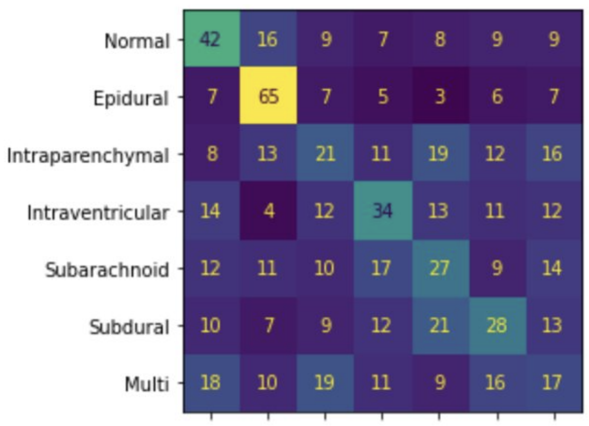


Figure 8: Newton-CG

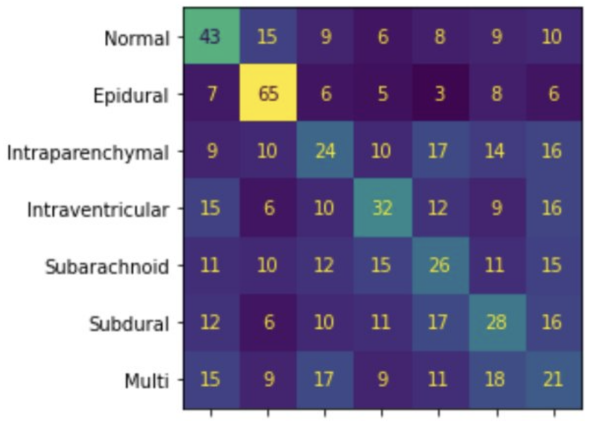
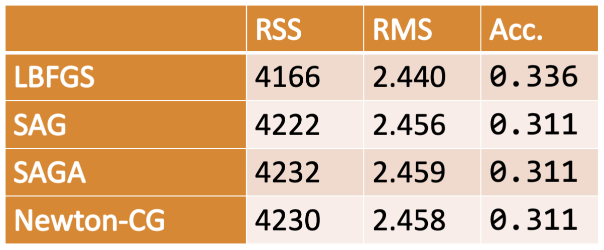


Figure 6: LBFGS

These confusion matrices show what the model predicted correctly vs. not using logistic regression.



	RSS	RMS	Acc.
LBFGS	4166	2.440	0.336
SAG	4222	2.456	0.311
SAGA	4232	2.459	0.311
Newton-CG	4230	2.458	0.311

Figure 9: Accuracies of Logistic Regression Models

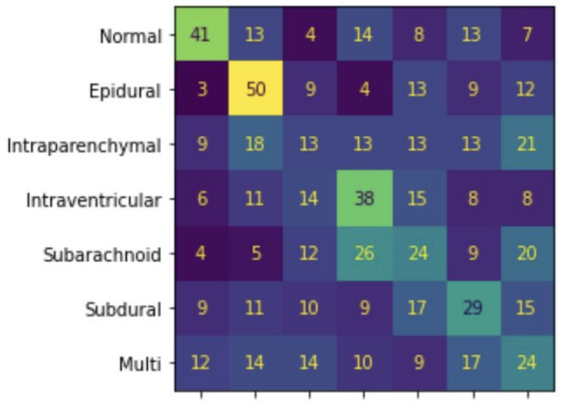


Figure 7: SAGA

5.2 Convolutional Neural Net

We constructed a Convolutional Neural Net using Tensorflow and plotted preliminary model results with Matplotlib. For our loss function we used categorical crossentropy, and used rmsprop as our optimizer.

We iterated on our preliminary model by training it on significantly more images, using the brain window instead of the brain bone window, and increasing the number of epochs during fitting.

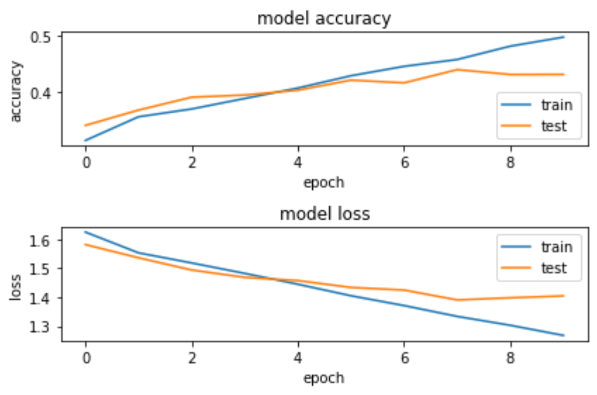


Figure 10: CNN (ReLU) Model Accuracy and Loss

Our best performing model incorporated ReLU as the activation function. The test data accuracy was 0.4307, and the training accuracy was 0.4973. While the training accuracy was higher, there does not appear to be significant overfitting. We also created new models with Sigmoid and Linear as activation functions.

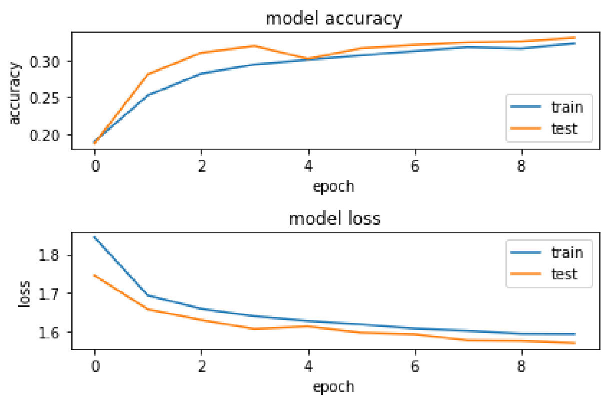


Figure 11: CNN (Sigmoid) Model Accuracy and Loss

We also saw worse results when using Sigmoid as an activation function compared to ReLU. The test data accuracy was higher than the training accuracy; the test data accuracy was 0.3237, and the training data accuracy was 0.3759.

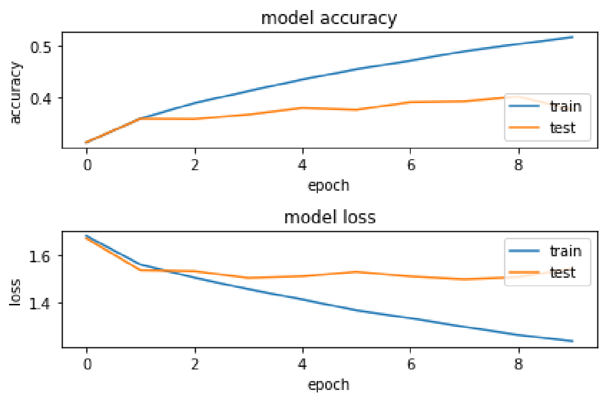


Figure 12: CNN (Sigmoid) Model Accuracy and Loss

We also saw worse results when using Linear as an activation function compared to ReLU; the model accuracy was 0.3759. However, the training accuracy was 0.5164, indicating that there was overfitting.

6 Conclusion

In summary, we were able to classify hemorrhage type from the given data using two separate methods. With convolutional neural nets, we yielded a max accuracy of 0.497, and with logistic regression, we yielded a max accuracy of 0.336.

From this, we saw that a CNN tended to work better than logistic regression for this problem, but overall it is a really hard problem to tackle. Moving forward, we could try different methods, or continuing to tweak our CNN to improve accuracy.

Overall, this project was a fun opportunity to apply what we learned in class to an important real-world application. It allowed us to really dig into understanding interesting data, learn more about brain scans and hemorrhages, and dive into work with TensorFlow and scikitlearn!

7 Acknowledgements

We would like to thank Zeta Surgical and CTO Raahil Sha for providing the data and presenting this problem.

References

[1] Solorio-Ramírez, José-Luis et al. "Brain Hemorrhage Classification in CT Scan Images Using Minimalist Machine Learning." Diagnostics (Basel, Switzerland) vol. 11,8 1449. 11 Aug. 2021, doi:10.3390/diagnostics11081449

[2] V. Davis and S. Devane, ”Diagnosis & classification of brain hemorrhage,” 2017 International Conference on Advances in Computing, Communication and Control (ICAC3), 2017, pp. 1-6, doi: 10.1109/ICAC3.2017.8318764.