Detection of Intracranial hemorrhage and its subtypes

Ritvik Khandelwal
ritvik02.kh@gmail.com

Shrenik Ganguli

***Abstract*—Intracranial hemorrhage is one of the top 5 fatal diseases in the US. In this paper, we aim to automate the process of detecting these bleeds in order to relieve overworked radiologists and flag patients in need of immediate attention.**

Keywords—hemorrhage, radiology, image classification, transfer learning, bag of tricks

I. Introduction

Hemorrhage in the brain (Intracranial Hemorrhage) is one of the top five fatal health problems. The hemorrhage causes bleeding inside the skull (typically known as cranium). It accounts for approximately 10% of strokes in the U.S thus diagnosing it quickly and efficiently is of utmost importance.

Intracranial Hemorrhage can have many consequences depending on the location, size, type of bleed in the brain. The relationship is not that simple to interpret. A small hemorrhage might be equally fatal as a big one if it is in a critical location. If a patient shows acute hemorrhage symptoms such as loss of consciousness, then an immediate diagnosis is required to carry out the initial tests, but the procedure is very complicated and often time-consuming. The delay in diagnosis and treatment can have severe effects on patients, especially the ones in need of immediate attention. The solution proposed in this project aims to make this process efficient by automating the preliminary step to identify the location, type, and size of the bleed, the doctors can provide a better diagnosis and a faster one as the work of identification is done by the algorithm. Also, the algorithm treats every scan with the same scrutiny, reducing human error and learning more as it goes.

Thus, the challenge is to build an algorithm to detect acute intracranial hemorrhage and its subtypes. This project will be a step towards identifying each of these relevant attributes for a better diagnosis.

II. Background and Related Work

Deep learning has been used extensively in medical imaging in the last decade whether it is detecting diabetes using retinopathy or pneumonia using Chest X-ray. Image segmentation has been used for color coding scans which is then used for applications like automatic measurement of organs, cell counting, or simulations based on the extracted boundary information. A recent paper by the Qure.ai team [1] shows the use of deep learning for classifying MRIs. Their approach included the use of CNNs as well as NLP techniques on the corresponding medical reports. The dataset included 313,318 anonymous CT scans from several centers in India. One major problem they faced was that of 3D scans. The mean age of the dataset was 43.4 years and 42.87% of the population was female. They managed to achieve an AUC of 0.94 ± 0.3 on all the sub categories. Other interesting papers include Expert-level detection of acute intracranial hemorrhage on head computed tomography using deep learning.[2], Extracting 2D weak labels from volume labels using multiple instance learning in CT hemorrhage detection.[3] In the latter, they used bags of 2D slices and calculated the losses per slice. These losses were then max pooled together to calculate the gradient and update parameters of the model. However, they used only about 600 images for this project.

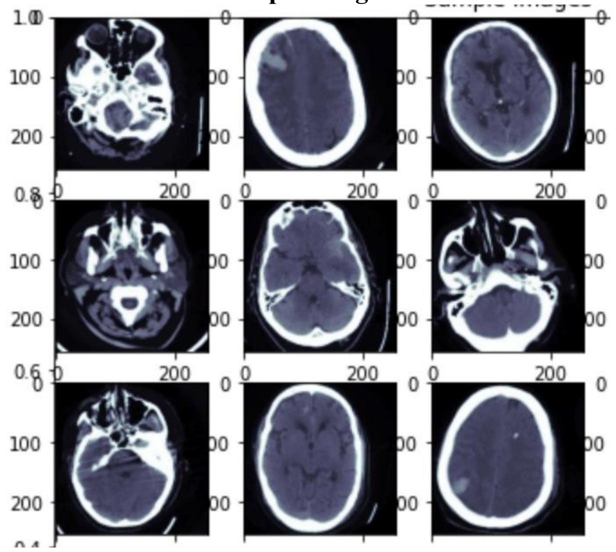
III. Implementation

This section explains in steps the work done till date.

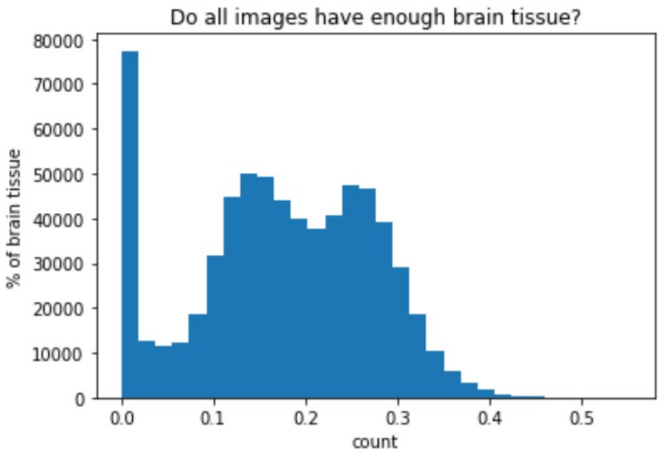
A. Data Cleaning

The data is given in the form of DICOM files. A DICOM file consists of metadata and image data packed into a single file. The information in the metadata consists of various tags. It also removes confidential information about patients to maintain the integrity. We have 194082 images in the dataset. We start by converting the metadata into a dataframe since the DICOM format is slow to use. We then view some of the images which are in the form of slices from top of the brain to the bottom of the brain as shown.

Sample Images



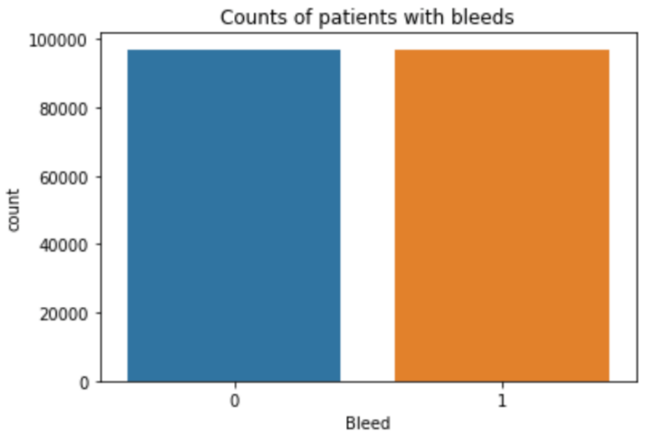
Sometimes the slice can be completely blank (a slice before or after the brain). To remove such slices, we use a column called “img_pct_window”. It tells us the percentage of brain pixels in the slice. We remove images with less than 20\% value in that column.



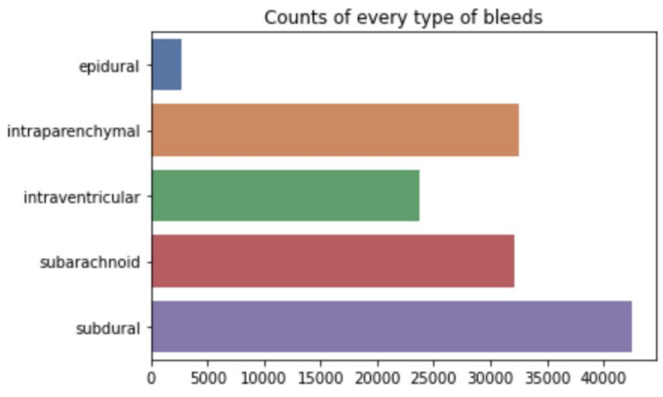
We also fix the column Rescale_Intercept, resize and crop the images to (256,256) and save them as ".jpg".

B. Data Exploration

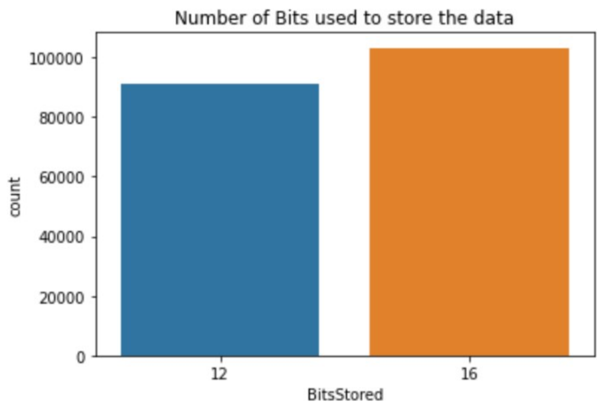
We look at the distribution of the data.



We see that we have a pretty balanced dataset when it comes to images with and without bleeds. We also check the counts of the different subtypes of bleeds.

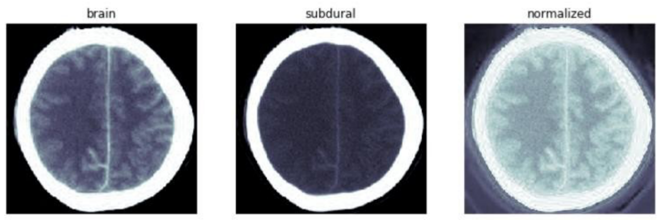


We see that "subdural" is the most common type of bleed present. We then move onto analyzing the metadata. One interesting column in the "bits stored" column which indicates the number of bits used to store the data. It has two distinct values: 12 and 16.



This might indicate that the data might have come form two different organizations. It is usually better for deep learning models that the data comes from same distribution but, we will move forward with this data.

Finally, the images can be viewed in many different windows.



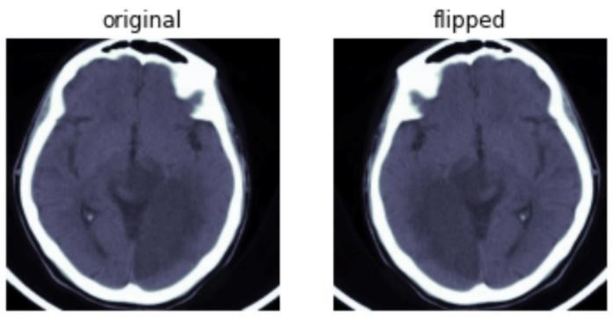
However, this is for the human perception. A neural network accepts floating point data and does not require any windowing.

C. Data Augmentation

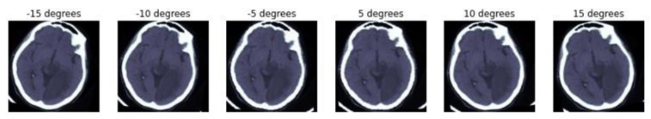
The first step to train a good deep learning model is to have lots of data. However, this is not always possible. Hence, we do small random transformations on the data that do not change what is inside the image (for the human eye) but changes the pixel values. Models trained with Data Augmentation generalize better.

Below are some of the transformations which can be used for our data.

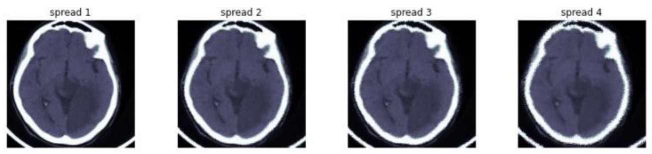
Flip: The bleed can occur on any side of the brain. Hence, flipping gives the model more cases to consider.



Rotation: The slices do not always appear straight. They might have different orientations. Hence, we randomly rotate few images in range of (-15, 15) degrees.



Blur: During an MRI, if the patient moves, the MRI might become blur. Hence, we apply blurring effect on some images. The blur effect is increased sequentially left to right.



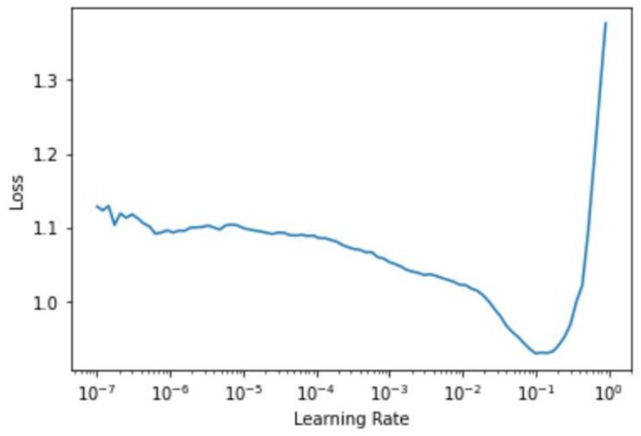
D. Modelling

(i) Binary Classification

Recent research has shown that metadata can prove useful in image classification. Our initial goal was to use a combination of metadata and images for the task. We try to classify the hemorrhages solely using the metadata to gauge its usefulness. We find that even after using a robust model like Random Forest, we get an accuracy of 50%. Hence, we decide to discard the metadata and focus on the images itself.

For our baseline model, we start with a ResNet18 model. Transfer learning has shown good results when it comes to image classification. It also reduces training time and resource consumption significantly.

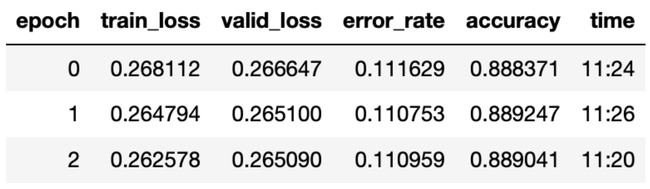
We use Leslie Smith's learning rate finder [5] to find a good learning rate.



In this technique, we run a mock training loop on our data varying our learning rate from a very low value to a value as high as 10. We then calculate the loss and plot it against the learning rates. The best learning rate would then be the one where the loss decreases the most.

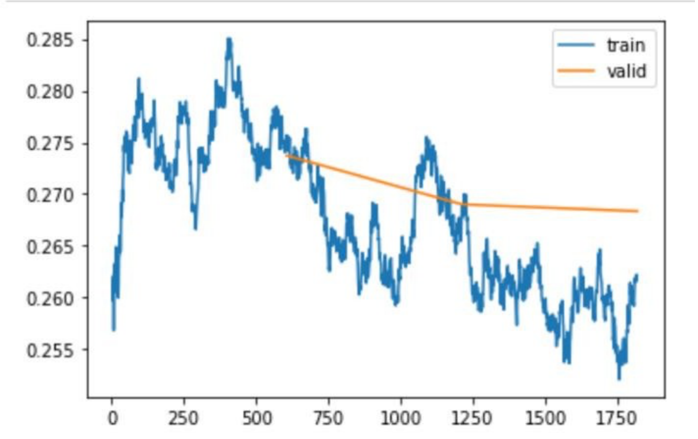
The batch size has been set to 256. We randomly sample 20% of the data as validation set. For hyperparameter tuning, we use Leslie Smith's one cycle training policy [4].

In this method, we initially freeze the pretrained part of the model and only train the new head. However, we do update the Batch Norm layers of the ResNet to maintain mean = 0 and std deviation = 1 of each layer. We have achieved ~89% accuracy with and without pretraining in just 5 epochs.



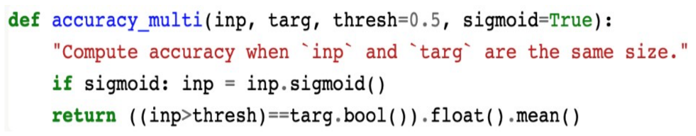
epoch	train_loss	valid_loss	error_rate	accuracy	time
0	0.268112	0.266647	0.111629	0.888371	11:24
1	0.264794	0.265100	0.110753	0.889247	11:26
2	0.262578	0.265090	0.110959	0.889041	11:20

Plot of the loss is as shown below:



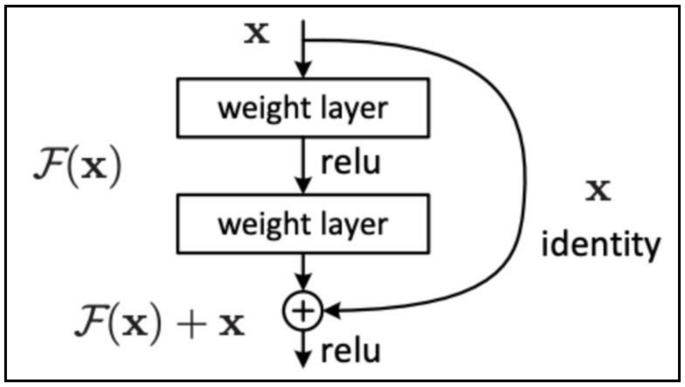
(ii) Multilabel Classification

For Multilabel Classification, we only include the images that have bleeds. We also create a new column in our data frame, which will indicate all the subcategories of bleeds present in an image, separated by semicolons. We use ResNet50 as our pretrained model. The other hyper parameters are the same as the ones used in Binary Classification except for accuracy. The first step is to decide a threshold. We have used threshold = 0.5 for our implementation. Then, instead of taking the maximum value from our sigmoid outputs, we take all the classes with a value greater than our threshold. This way we get multiple classes per image. We can then compare them with the ground truths and get the accuracy. Our new accuracy formula looks as follows:



```
def accuracy_multi(inp, targ, thresh=0.5, sigmoid=True):
    "Compute accuracy when `inp` and `targ` are the same size."
    if sigmoid: inp = inp.sigmoid()
    return ((inp>thresh)==targ.bool()).float().mean()
```

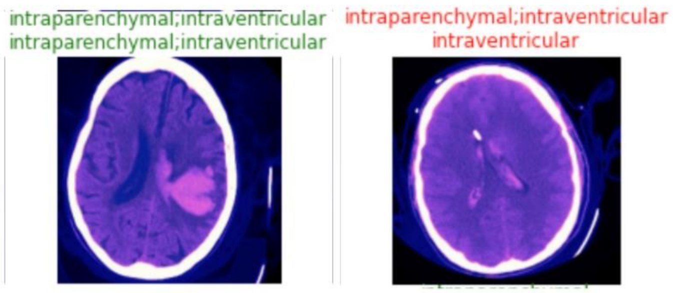
This architecture gives us an accuracy of about 91.3%. A ResNet pretrained model includes a number of res blocks which are also known as identity or skip connections. ResNets have been revolutionary for image classification and the reason they work so well is because they solve the problem of gradient diffusion. A typical ResBlock architecture is shown below:


x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

What it does it, for every 2 convolutions, it adds the input to those convolutions to their output. If we consider our convolutions as functions, then the output goes from:

$$c2(c1(x)) \Rightarrow x + c2(c1(x))$$

We can view some of the results below.


intraparenchymal;intraventricular
intraparenchymal;intraventricular
intraparenchymal;intraventricular
intraventricular

The output has two lines above every image where the upper line is the prediction and lower line is the actual value. The output in green implies correct prediction and red implies a wrong prediction.

IV. Model Optimization

A. *Different Architectures*

(i) ResNet101

Whenever one wants to increase the accuracy of a classification model, the easiest way to do so is to increase the number of layers. Hence, we change the pretrained model from ResNet50 to ResNet101. This increases our accuracy from 91.3% to 91.7%.

(ii) DenseNet121

The next architecture we try is the DenseNet121 architecture. In DenseNet, each layer receives collective knowledge from all preceding layers. For this, the output of a convolution is concatenated with the input. This concatenated feature-map is passed to the next layer. Since this increases the number of parameters, we reduce the batch size by half. With the DenseNet, we get the accuracy of 91.8%

(iii) AlexNet

The final architecture we try is the AlexNet. This is another famous architecture for image classification. It won the ImageNet competition in 2012. It uses overlap pooling to reduce the size of the network and trains quite fast.

Although, the training is much faster than the other architectures we used, the accuracy was not better. It came to 89%.

B. *Weight Decay*

It is one of the regularization techniques in Deep Learning. It prevents our model from getting too complex by penalizing complexity. It does so by adding the sum of squares of all parameters to the loss function. However, this can make the loss so huge that the best model would set all the parameters to 0. To prevent this from happening, we multiply the sum of squares with another small number. This number is called weight decay (wd).

$$loss = crossentropy(\hat{y}y) + wd * \sum parameter^2$$

We use weight decay value = 0.01 This helps the model learn much better.

C. *Mixed Precision Training*

In neural networks, inputs, parameters and activations are stored using 32 bits. This gives us a high amount of precision. However, these high precision computations, require more time and memory. Hence, using 32-bit precision for all our calculations, we perform some of them in 16 bits. Weight updates need to be as precise as possible hence, we haven't reduced the bits for weight updates.

In theory, mixed precision training should reduce the training time by half. However, in practice we see only a slight decrease in the training time. But, in our implementation, the time did not reduce much.

D. *Progressive Image Resizing*

Research has shown that training a model on smaller sized images and then gradually increasing the size during the train process helps the model learn better. Hence we start by training a model with images of size (64,64), we then increase it to (128,128) and finally to (256,256). Progressive image resizing does not give good results immediately but only after a lot of epochs.

V. References

[1] Sasank Chilamkurthy, Rohit Ghosh, Swetha Tanamala, Pooja Rao and Qure.ai team. Development and Validation of Deep Learning Algorithms for Detection of Critical Findings in Head CT Scans.

[2] Weicheng Kuo, Christian Häne, Pratik Mukherjee, Jitendra Malik, and Esther L. Yuh, Expert-level detection of acute intracranial hemorrhage on head computed tomography using deep learning.

[3] Samuel W. Remedios, Zihao Wu, Camilo Bermudez, Cailey I. Kerley, Snehashis Roy, Mayur B. Patel, John A. Butman, Bennett A. Landman, and Dzung L. Pham, "Extracting 2D weak labels from volume labels using multiple instance learning in CT hemorrhage detection".

[4] Leslie N. Smith, " A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay".

[5] Leslie N. Smith, "Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates".