# Detect and Classify Intracranial Hemorrhages from Computed Tomography Data

Ryan Lussier

**Problem Statement**

Intracranial hemorrhages affect 40,000-67,000 people per year in the US alone with the mortality rate ranging from 35-52%. Typically patients experiencing severe symptoms such as loss of consciousness or acute headaches go through an urgent computed tomography (CT) and the CT images must be reviewed by a skilled radiologist to complete the diagnosis. Quickly and accurately identifying the type and location of the hemorrhage is critical for treatment of the patient.

The client for this model is hospitals. The RSNA says there is currently a shortage of radiologists. A successful model could not only remove some of the burden from radiologists but also has the potential to improve detection/diagnosis rate which will also benefit patients at said hospitals. Automating the analysis of CT images has the potential to simplify and expedite the ICH diagnosis process.

The data for this project is compiled by the RSNA and consists of xxx computed tomography images, associated metadata, and labels. We plan to utilize a convolutional neural network (CNN) to help detect ICH's and classify them into four subcategories to aid in diagnosis and treatment. In addition, we plan to explore ensemble methods and potentially GAN's to see if they can boost performance.

The deliverables for this project will be a fully trained model, performance evaluation of that model, a paper outlining the project and potential next steps as well as a slide deck for discussing key elements of the project.

**EDA**

The dataset for this project is primarily made up of dicom files which is a common file format for CT scans. The data came from an old Kaggle competition so the dicom files were already divided into training and test sets. The training set consists of roughly 750,000 dicom files and the test set 121,000. In addition to the scans, there are two csv files containing all of the classification labels for both the training and test data sets.

To work with dicom files, we will use the pydicom library which provides the tools to extract data from the dicom files to view and store in formats more suitable for machine learning tasks. A dicom file is made up of both metadata as well as the image itself. The following figure is an example of the metadata found in a dicom file. In addition to information such as patient ID, image dimensions, and pixel spacing there is the image itself. There is also a rescale

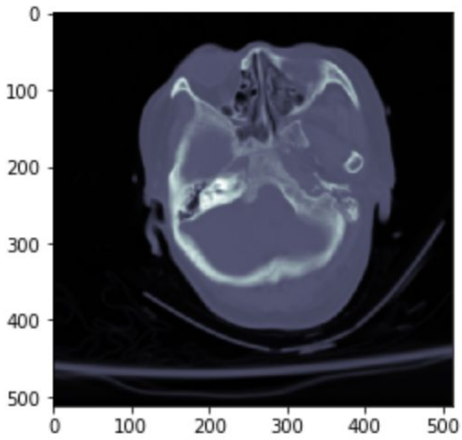
intercept and slope which will be used to scale the data into Hounsfield units, which are a scale for measuring the radiographic intensity/density.

```
Dataset.file_meta -------------------------------
(0002, 0000) File Meta Information Group Length  UL: 176
(0002, 0001) File Meta Information Version       OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID         UI: CT Image Storage
(0002, 0003) Media Storage SOP Instance UID      UI: 9999.238806058494015810637175892458903833803
(0002, 0010) Transfer Syntax UID                 UI: Explicit VR Little Endian
(0002, 0012) Implementation Class UID            UI: 1.2.40.0.13.1.1.1
(0002, 0013) Implementation Version Name         SH: 'dcm4che-1.4.38'
-------------------------------------------------
(0008, 0018) SOP Instance UID                    UI: ID_2baac3f47
(0008, 0060) Modality                            CS: 'CT'
(0010, 0020) Patient ID                          LO: 'ID_5dd962d3'
(0020, 000d) Study Instance UID                  UI: ID_0973ecd0cc
(0020, 000e) Series Instance UID                 UI: ID_3283f2e9a8
(0020, 0010) Study ID                            SH: ''
(0020, 0032) Image Position (Patient)            DS: [-103, 44.7747216, 102.273802]
(0020, 0037) Image Orientation (Patient)         DS: [1, 0, 0, 0, 0.927183855, -0.374606593]
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0010) Rows                                US: 512
(0028, 0011) Columns                             US: 512
(0028, 0030) Pixel Spacing                       DS: [0.48828125, 0.48828125]
(0028, 0100) Bits Allocated                      US: 16
(0028, 0101) Bits Stored                         US: 12
(0028, 0102) High Bit                            US: 11
(0028, 0103) Pixel Representation                US: 0
(0028, 1050) Window Center                       DS: [00040, 00040]
(0028, 1051) Window Width                        DS: [00080, 00080]
(0028, 1052) Rescale Intercept                   DS: "-1024.0"
(0028, 1053) Rescale Slope                       DS: "1.0"
(7fe0, 0010) Pixel Data                          OW: Array of 524288 elements
```

Next, let's take a look at one of the image files. To do this, we extract the pixel data from the cmd file object as:

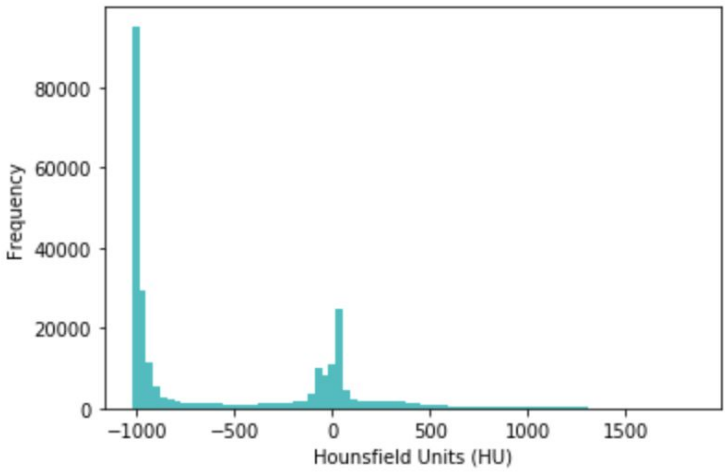
scan = pydicom.dcmread('filename.dcm').pixel_array

and then plotting the array using matplotlib.pyplot.

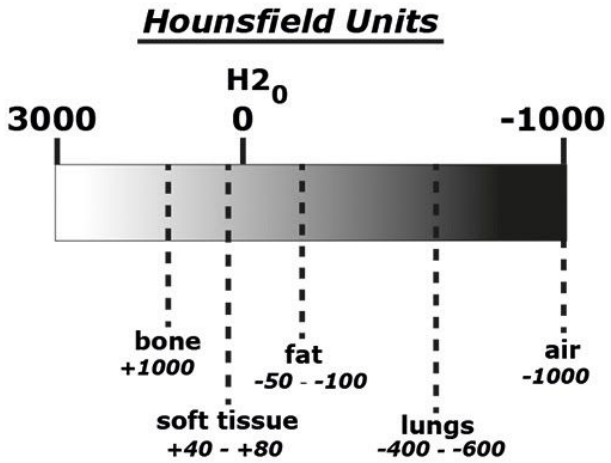



We see what appears to be a cross section of a human head centered on black background. The bulk of the head appears dark grey with some areas of white intensity. We can also look at the distribution of the pixel intensities. Before doing this, however, we should use the scaling intercept and slope to convert the raw pixels to Hounsfield units. This accounts for

differences in intensity measurements between different CT imaging machines and allows us to compare the distribution to a standard scale.



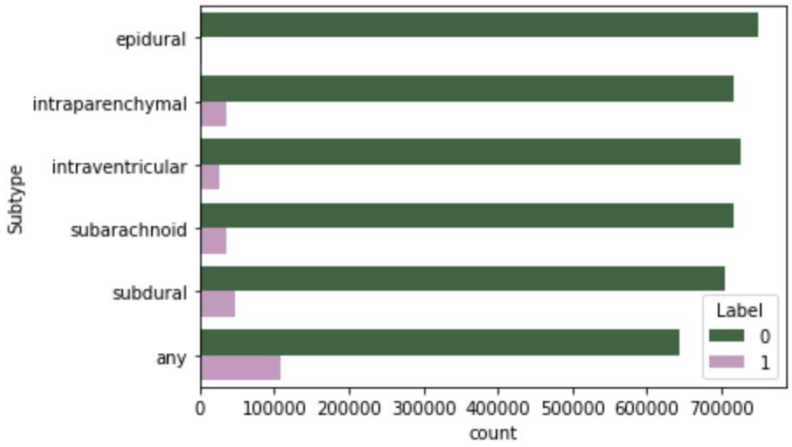

By comparing to the scale below, we see that the image has its largest peak around -1000 which is representative of air. In addition we see a peak around 0 which is representative of soft tissue and fat. While there is not a noticeable peak around 1000 on this distribution, it is helpful to note that this value corresponds to bone which registers as a light grey or white color.




As the goal is to identify and classify hemorrhages, each of these image files must have a label. Let's take a closer look at this portion of our data which is stored as a csv file. Importing this into a pandas dataframe and taking a look at the first few rows we see a column for each scan ID, a binary label, and the subtype of hemorrhage.

| | ID | Label | Subtype |
|---|---|---|---|
| 0 | ID_12cadc6af | 0 | epidural |
| 1 | ID_12cadc6af | 0 | intraparenchymal |
| 2 | ID_12cadc6af | 0 | intraventricular |
| 3 | ID_12cadc6af | 0 | subarachnoid |
| 4 | ID_12cadc6af | 0 | subdural |
| 5 | ID_12cadc6af | 0 | any |
| 6 | ID_38fd7baa0 | 0 | epidural |
| 7 | ID_38fd7baa0 | 0 | intraparenchymal |
| 8 | ID_38fd7baa0 | 0 | intraventricular |
| 9 | ID_38fd7baa0 | 0 | subarachnoid |

The subtypes fall into one of 6 categories. Epidural, intraparenchymal, intraventricular, subarachnoid, subdural, and “any.” By pivoting the label column we can create features for each subtype and label these columns based on the subtype feature. These can then be easily converted to a vector to be stored as Y values when training the model. This also allows us to easily create a seaborn countplot. The two main observations for this plot are that the data is imbalanced and that there appear to be no observations of epidural hemorrhages which means it may be appropriate to drop this feature/label.




**Transformation**

While the full dataset contains roughly 750,000 scan images, I decided to randomly sample 10,000 image files to make working with the data more tractable. If time and resources

allow, the model can be retrained on the full data set to see how this impacts model performance.

First we loop through the training data folder where the sample data was stored to apply some preprocessing. The data is scaled by applying 'apply_modality_lut' from the pydicom library. This scales the pixel array based on the rescale slope and intercept per the following formula: $(pixel\ value * Rescale\ Slope + Rescale\ Intercept)$. After scaling the data is normalized based on the maximum of the absolute value of the pixels for each image.

```
# definte training and test paths for data input/output
train_path = '/Users/ryanlussier/ICH Detection and Classification/data/train/'
test_path = '/Users/ryanlussier/ICH Detection and Classification/data/test/'

# create directory object for training data
directory = os.fsencode(train_path)

ids = []
images = []

#loop through all files in training data directory
for file in os.listdir(directory):
    filename = os.fsdecode(file)
    if filename.endswith(".dcm"):
        # read dcm file
        scan = pydicom.dcmread(train_path + filename)
        ids.append(filename.split('.')[0])
        # extract pixel array
        image = scan.pixel_array

        # transform to hounsfield scale
        image = apply_modality_lut(image, scan).astype(dtype='float16')

        # max normalize each image
        mx = np.max(np.abs(image))
        image_normed = image/mx

        #append pixel array to images list
        images.append(image_normed)

        continue
    else:
        continue
```

These processed images were added to a dataframe with their corresponding ID's and then joined on the ID's to create a new dataframe with the Y-values vector column created during exploratory data analysis. This new data frame was then saved as two pickel files to be recalled later as needed. The data was stored in two pickel files because saving as a single file repeatedly killed the notebook kernel because of insufficient RAM.

**Modeling**

For modeling, we will implement a convolutional neural network (CNN). CNN's are widely used to solve image classification problems and have a history of success, so they are an appropriate place to start for this problem. The Keras model will be used to implement the model because of its simple syntax and low barrier to entry.

The x and Y values can be extracted from the last joined/pickled dataframe. The x values in this case will be made up of all of the pixel arrays while the Y values will be the label vectors. To ensure compatibility with Keras, these series are both extracted from the dataframe and converted to numpy arrays. These arrays are then split using sklearn.model_selection.train_test_split to create a validation consisting of a random sample of 20% of the total training set. While the Y vectors are already in the correct format for Keras, the x input will be reshaped so the first element of the array contains the image files, the second and third elements of the array contain the image dimensions, and the last element is the number of channels. In this case, the images are grey-scale so the number of channels is simply one.

Next, the architecture of the model must be defined. As a starting point, we will use the conventional structure shown below. A batch size of 80 was used with 25 epochs. The number of nodes in each convolutional layer was kept small to start to reduce training time. The relu activation function was used on all layers except the last fully dense layer where softmax was used instead. On the last layer, the number of nodes is set to equal the number of classes.

```
batch_size = 80
num_classes = 6
epochs = 25

model = Sequential()
model.add(Conv2D(16, kernel_size=(7, 7),
                 activation='relu',
                 input_shape=(512,512,1)))
model.add(Conv2D(32, (5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

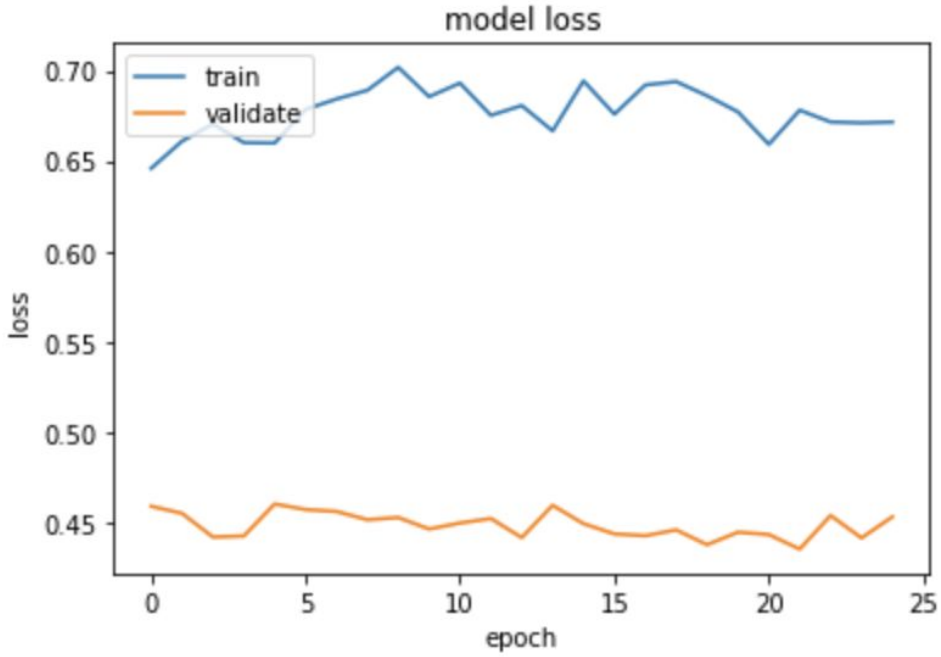
After defining and building the model architecture, the model must be compiled. Here we define the loss function as categorical cross-entropy, the optimizer as Adadelta, and establish accuracy as the metric for evaluating model performance. Cross-entropy was used as it seemed generally accepted as the loss function for multi-class problems. The model is then fit and stored in an object named history so it can be accessed later to plot loss and accuracy and use these plots to assess model performance and determine whether or not the model is learning.

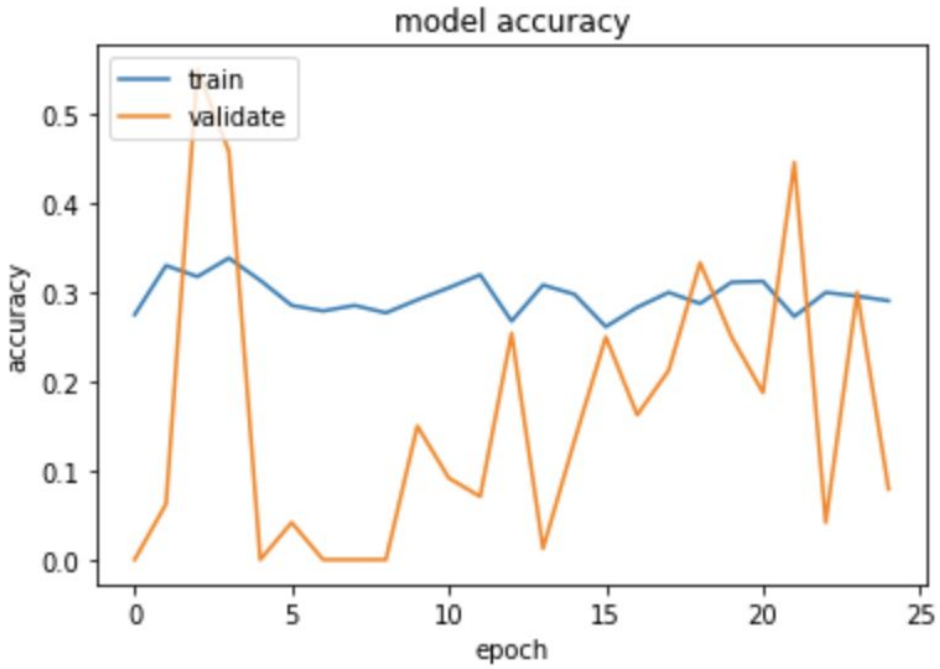
These plots can also be used to evaluate which epoch training should stop on to prevent overfitting.

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history = model.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_test, y_test))
score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

After completing 25 epochs of training, we plot the loss and see that the CNN is failing to optimize the loss function. In the case of categorical cross-entropy, the function is minimized so a model which is learning should show a downward trend. A look at the accuracy plot tells the same story with the training results showing no improvement and the validation set being very irregular.






**Conclusion**

Ultimately, the results of initial attempts at utilizing a simple CNN to classify brain hemorrhages were not successful. With more time and research, there could likely be significant improvements. Taking a closer look at data preprocessing and attempting to augment the images to improve contrast/signal by applying a filter with erosion, dilation and thresholding is one example that has shown promise in some applications in medical imaging. Additionally, utilizing transfer learning rather than training the model from scratch could enable the model to better recognize simple features and have a better starting point for adapting to the complexity of these CT images. Lastly, scaling up the model by utilizing a more powerful machine or cloud computing service to enable the use of increased nodes, layers and/or the entire training set could also help improve the model performance.