Implementing and Analyzing Three Machine Learning Models For Brain Tumor Recognition

Vivian Chang, Ashwin Chintalapati, Chessa Park, Artem Tesov, and Keying Zhang

Abstract

Magnetic resonance imaging is a medical imaging technique that uses a magnetic field and computer generated radio waves in order to create images of the human body's organs and tissues. Radiologists use these images to determine what condition your body is in, detect any tumors or other anomalies, and reach a specific diagnosis regarding your health. However, some tumors may be difficult to easily identify in a limited time frame. Throughout the years, scientists have focused on improving both the versatility as well as the accuracy of these MRI machines. One way to do this was by making the process more automated. Machine learning allows systems to recognize patterns, improve from experience, and predict outcomes. Such models can be used to identify brain tumors from brain MRI images.

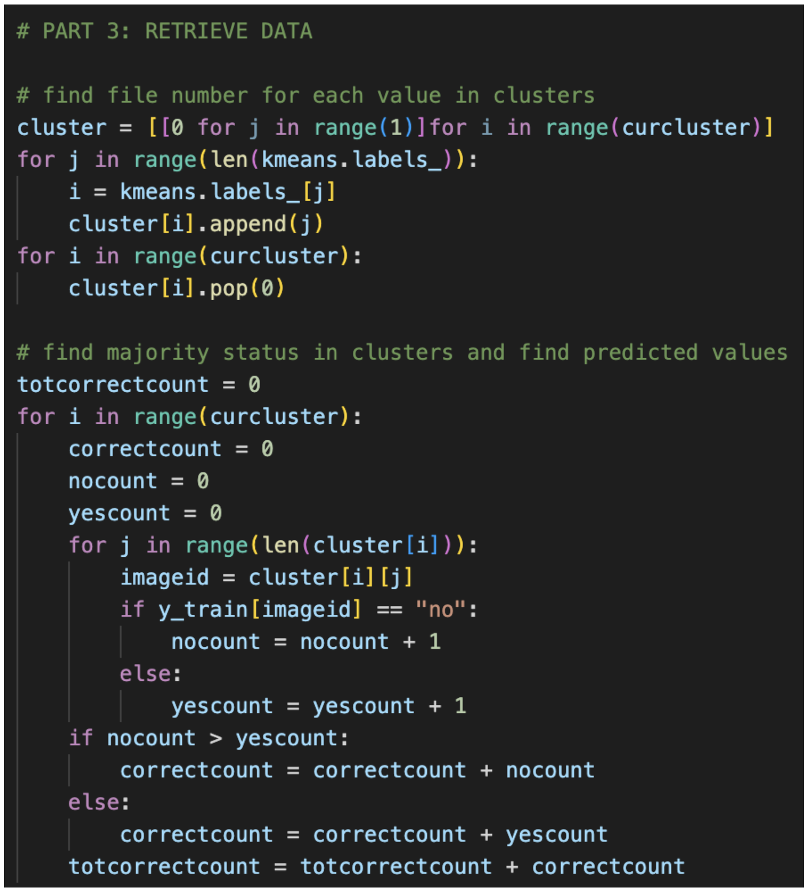
Here we implement the following three models: K-means algorithm with image clustering, K-means algorithm with feature clustering, and convolutional neural network for image classification. In addition, we analyze the performances of a few of these models using the following metrics: accuracy, ease of training and tuning, generalizability, and explainability.

Methods

a) K-means algorithm with image clustering

In this K-means model, each cluster is meant to represent a group of images with similar features that determine whether the images have tumors or not. For example, in a model with five clusters, three might represent groups of images with tumors and two represent groups of images without tumors. Each value in the cluster is a flattened one-dimensional array representing the image file, where the elements inside the array are the RGB values normalized to a range of 0-1.

This algorithm is implemented in VSCode using an available K-means algorithm from the Sklearn Library.[11] After the K-means algorithm returns the cluster labels, we use the following code to determine the predicted tumor statuses of the images.



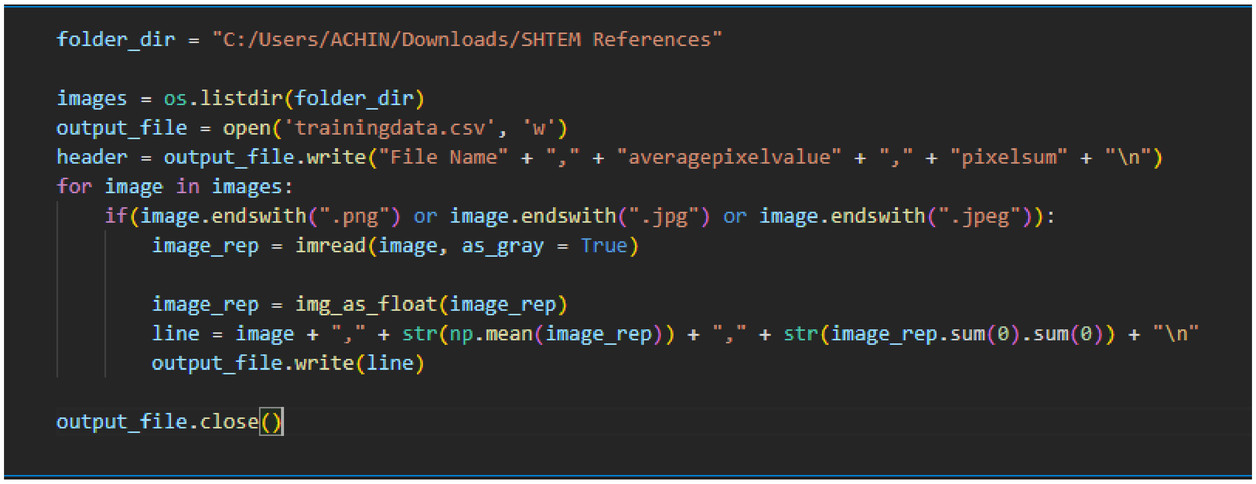
```python
# PART 3: RETRIEVE DATA

# find file number for each value in clusters
cluster = [[0 for j in range(1)]for i in range(curcluster)]
for j in range(len(kmeans.labels_)):
    i = kmeans.labels_[j]
    cluster[i].append(j)
for i in range(curcluster):
    cluster[i].pop(0)

# find majority status in clusters and find predicted values
totcorrectcount = 0
for i in range(curcluster):
    correctcount = 0
    nocount = 0
    yescount = 0
    for j in range(len(cluster[i])):
        imageid = cluster[i][j]
        if y_train[imageid] == "no":
            nocount = nocount + 1
        else:
            yescount = yescount + 1
    if nocount > yescount:
        correctcount = correctcount + nocount
    else:
        correctcount = correctcount + yescount
    totcorrectcount = totcorrectcount + correctcount
```

In the above code, we first find the tumor status of each image, which was given in the file names of the training set; this was set as “True”. Here, we change all the images in a cluster to have the status either “no” or “yes” based on the majority status of that cluster; this changed depiction represents “Predicted.” The next step for the model would be for an image of unknown status to be placed in a cluster through the model and have a predicted status returned.

b) K-means algorithm with feature clustering

In order to properly implement the K-means algorithm with feature clustering, we needed to be able to isolate and iterate through images, obtaining the two specific features we want. The following step will be using VS Code and Jupyter Notebooks to analyze and form our clusters, then verifying accuracy of the model. Depending on the resulting accuracy, we change dimensions, scale our values, or do both, while also using the elbow method to verify our practicality of the algorithm.

The following code allows us to input image statistics into a .csv file:

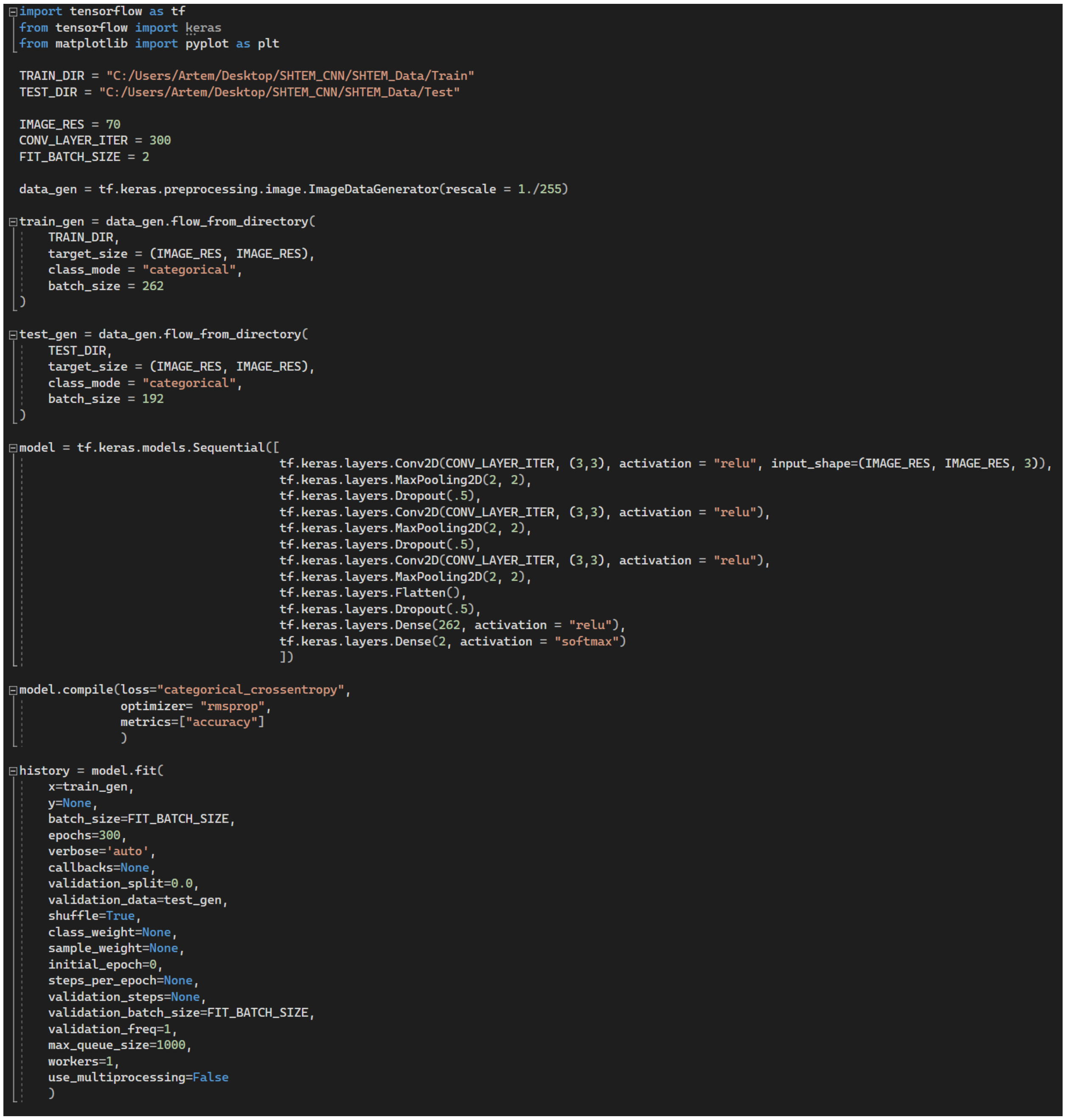


```
folder_dir = "C:/Users/ACHIN/Downloads/SHTEM References"

images = os.listdir(folder_dir)
output_file = open('trainingdata.csv', 'w')
header = output_file.write("File Name" + "," + "averagepixelvalue" + "," + "pixelsum" + "\n")
for image in images:
    if(image.endswith(".png") or image.endswith(".jpg") or image.endswith(".jpeg")):
        image_rep = imread(image, as_gray = True)

        image_rep = img_as_float(image_rep)
        line = image + "," + str(np.mean(image_rep)) + "," + str(image_rep.sum(0).sum(0)) + "\n"
        output_file.write(line)

output_file.close()
```

Now, we can analyze the data and use the K-Means algorithm to find clusters and centroids. We test for model accuracy, as well as ease of training/tuning ability (how easily the model can be used for multiple different parameter values). The final components we check for are generalizability, as well as explainability. Both of these are strengths of the K-means model, as the concept is rather easy to understand, and can be used for any two dimensional set of data.

c) Convolutional Neural Network For Image Classification

An alternative approach with image classification was taken for our third model: a Convolutional Neural network. Unlike the K-means algorithm implementations, this model would be able to train itself to classify images based on the details that the brain images were composed of, and identify images containing components indicating the presence of a brain tumor. For this implementation, the Tensorflow[3] python library and its higher level Keras sub-API. Keras was used because of convenience, since low-level optimization is not needed for a proof of concept implementation. The following was the code which processed images for the model, created the Convolutional Neural Network, and trained and tested the model using the common MRI dataset for testing, and an additional MRI dataset for testing accuracy (see Materials):



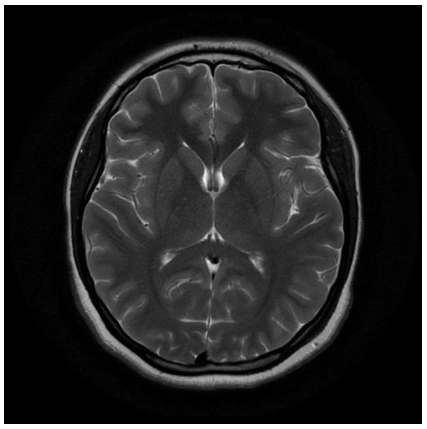
```python
import tensorflow as tf
from tensorflow import keras
from matplotlib import pyplot as plt

TRAIN_DIR = "C:/Users/Artem/Desktop/SHTEM_CNN/SHTEM_Data/Train"
TEST_DIR = "C:/Users/Artem/Desktop/SHTEM_CNN/SHTEM_Data/Test"

IMAGE_RES = 70
CONV_LAYER_ITER = 300
FIT_BATCH_SIZE = 2

data_gen = tf.keras.preprocessing.image.ImageDataGenerator(rescale = 1./255)

train_gen = data_gen.flow_from_directory(
    TRAIN_DIR,
    target_size = (IMAGE_RES, IMAGE_RES),
    class_mode = "categorical",
    batch_size = 262
)

test_gen = data_gen.flow_from_directory(
    TEST_DIR,
    target_size = (IMAGE_RES, IMAGE_RES),
    class_mode = "categorical",
    batch_size = 192
)

model = tf.keras.models.Sequential([
                                    tf.keras.layers.Conv2D(CONV_LAYER_ITER, (3,3), activation = "relu", input_shape=(IMAGE_RES, IMAGE_RES, 3)),
                                    tf.keras.layers.MaxPooling2D(2, 2),
                                    tf.keras.layers.Dropout(.5),
                                    tf.keras.layers.Conv2D(CONV_LAYER_ITER, (3,3), activation = "relu"),
                                    tf.keras.layers.MaxPooling2D(2, 2),
                                    tf.keras.layers.Dropout(.5),
                                    tf.keras.layers.Conv2D(CONV_LAYER_ITER, (3,3), activation = "relu"),
                                    tf.keras.layers.MaxPooling2D(2, 2),
                                    tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dropout(.5),
                                    tf.keras.layers.Dense(262, activation = "relu"),
                                    tf.keras.layers.Dense(2, activation = "softmax")
                                    ])

model.compile(loss="categorical_crossentropy",
              optimizer= "rmsprop",
              metrics=["accuracy"]
              )

history = model.fit(
    x=train_gen,
    y=None,
    batch_size=FIT_BATCH_SIZE,
    epochs=300,
    verbose='auto',
    callbacks=None,
    validation_split=0.0,
    validation_data=test_gen,
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_batch_size=FIT_BATCH_SIZE,
    validation_freq=1,
    max_queue_size=1000,
    workers=1,
    use_multiprocessing=False
    )
```

First, testing and training images are retrieved and formatted, with batch size matching the number of images in each directory. Then, a model is created with three convolutional layers, each with their respective pooling layer and a dropout layer. Afterwards the model is compiled using the categorical cross entropy loss function contained in Keras, an optimizer, and a metric. Finally the model is trained and tested over 300 epochs, at a batch size of 2. It should be noted that GPU limitations required us to balance the image resolution, image batch size, and kernel count. The GPU used in this implementation was a NVIDIA RTX 2080 with 8GB VRAM. Though batch size was low, it had negligible effects on the test accuracy when compensated for with high epoch counts. Finally, Matplotlib was used to chart the accuracy data produced per epoch. This would allow us to visualize the accuracy of the model, as well analyze the significance that epoch count has on model accuracy.

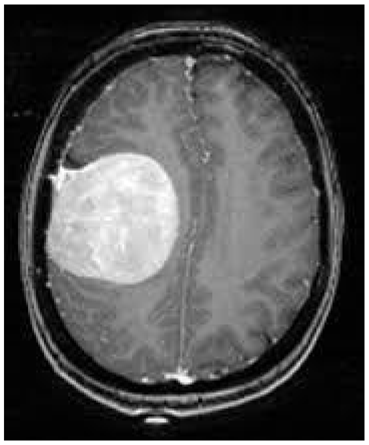
Materials

The training dataset used by all three models is the "Brain MRI Images for Brain Tumor Detection" set from Kaggle[1]. Below are two MRI scan files from the dataset.

Brain with no tumor, "1 no.jpeg"

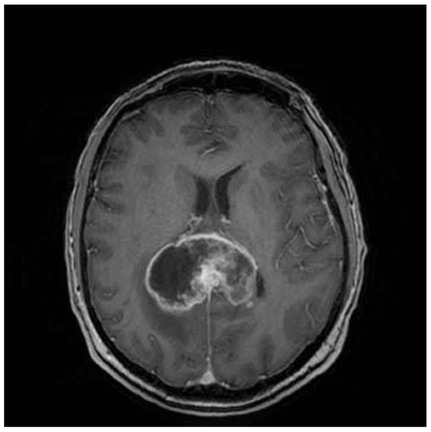


Brain with tumor, "Y1.jpg"

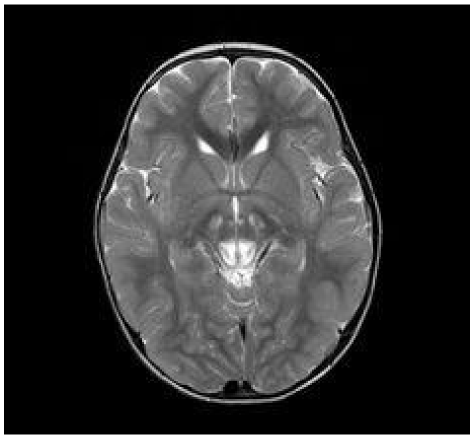


It should be also noted that the Convolutional Neural Network implementation used an additional 192 images from the "Brain Tumor Classification (MRI)" set, also acquired from Kaggle[2]. Some images from this dataset were not included due to the inconsistency in their framing with the first dataset.

Brain with tumor, "image(32).jpeg"



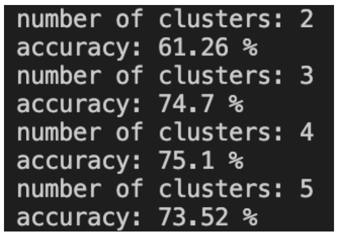
Brain with no tumor, "image(2).jpeg"



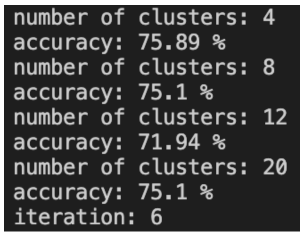
Results

a) K-means algorithm with image clustering

After implementation, the model returns a raw accuracy score of around 75%, which varies for cluster sizes ranging from 2 to 8 and is highest at cluster size 4. Therefore, it looks like k=4 is the optimal cluster size for this model.



```
number of clusters: 2
accuracy: 61.26 %
number of clusters: 3
accuracy: 74.7 %
number of clusters: 4
accuracy: 75.1 %
number of clusters: 5
accuracy: 73.52 %
```

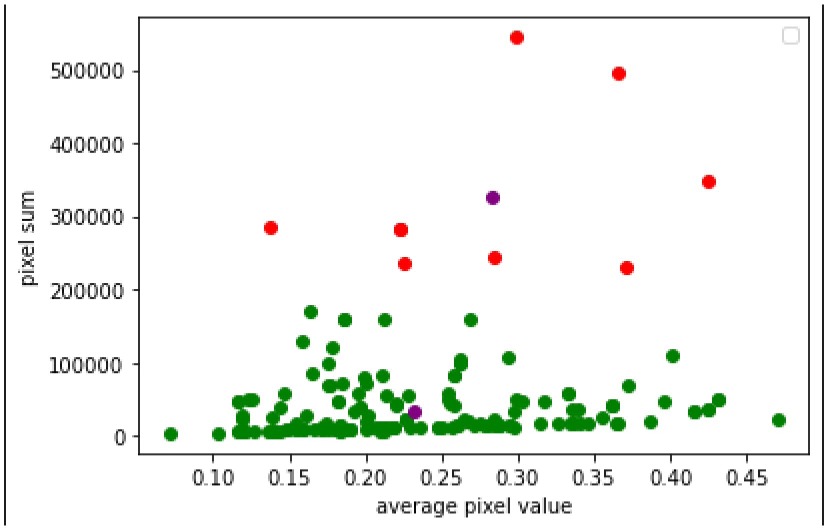


```
number of clusters: 4
accuracy: 75.89 %
number of clusters: 8
accuracy: 75.1 %
number of clusters: 12
accuracy: 71.94 %
number of clusters: 20
accuracy: 75.1 %
iteration: 6
```

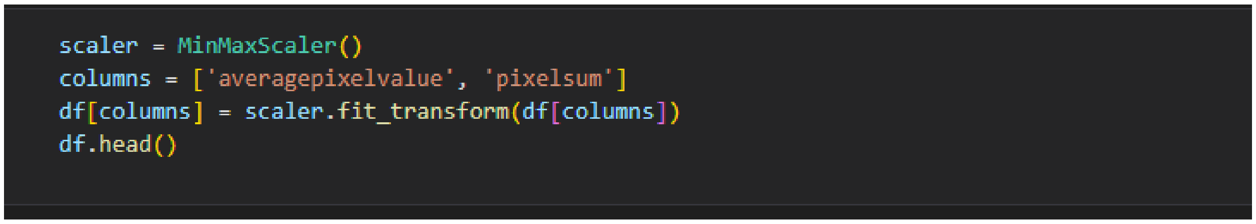
Based on this implementation, we now look at the four metrics. The accuracy metric receives a score of average, with raw accuracy around 75%. The model shows much potential, since this is only the first version of the model. With further optimization, such as removing outliers, and training on a larger data set, the accuracy score has the potential to increase significantly. The ease of training/tuning metric receives a score of average, since the K-means algorithm only returns unlabeled clusters and requires an additional algorithm to retrieve data. The generalizability metric receives a score of good, since the model scales well to other datasets, including those that are much larger. The explainability metric receives a score of good, since the K-means algorithm is a simple concept and can easily be implemented.

b) K-means algorithm with feature clustering

#1: Average pixel value vs pixel sum

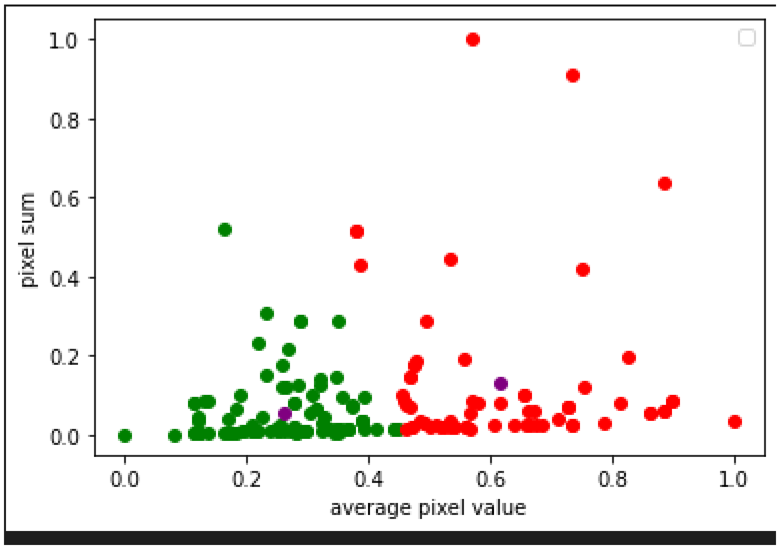


Looking at the preliminary results for analyzing average pixel value vs average pixel sum, we face an unfortunate complication. Since clusters are formed by calculating the Sum of squared error between a data point and its centroid, both the x and y axes will have to be on the same scale. In this case, since y axis numbers are skewed, we must scale both parameter values using the MinMaxScaler.



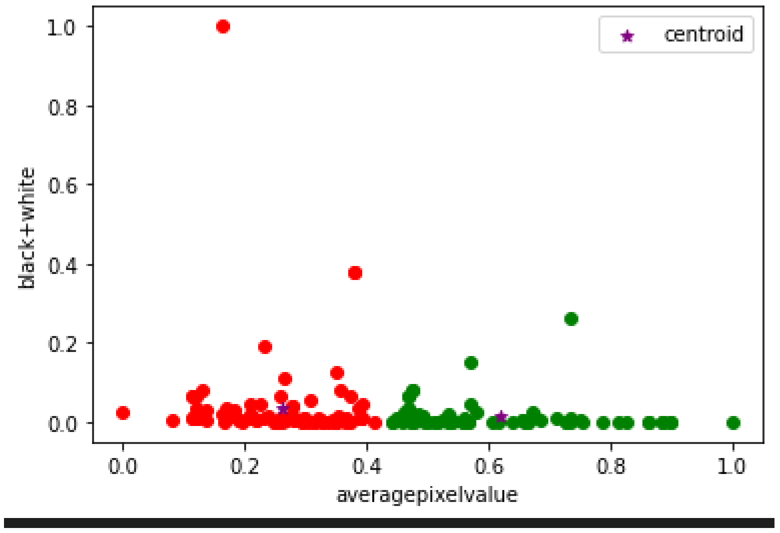
```
scaler = MinMaxScaler()
columns = ['averagepixelvalue', 'pixelsum']
df[columns] = scaler.fit_transform(df[columns])
df.head()
```

After scaling all relevant data, we achieve the following graph:



About 72% of the data points are in the correct clusters.

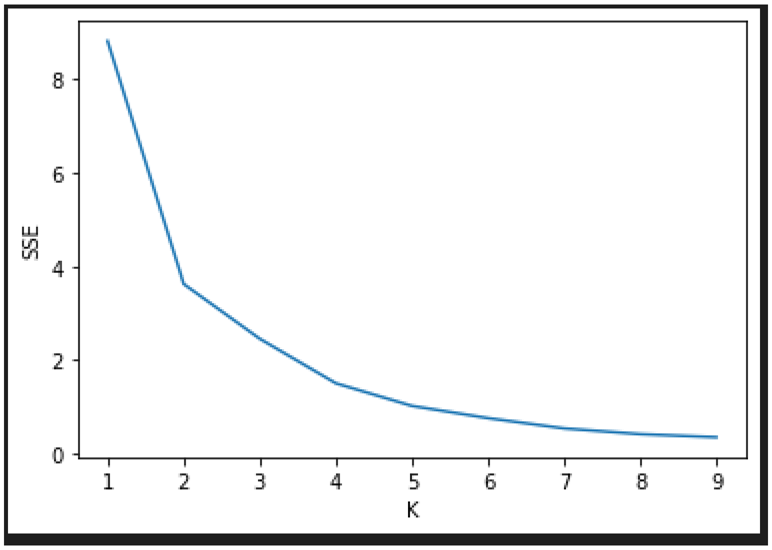
Similarly, we achieve the following for average pixel value vs. black+white pixels:



This graph has far less skew or outliers compared to the previous graph, indicating it may be more accurate. Indeed, when verifying the model, we see it has 75% accuracy, a slight uptick.

The Final Test: The Elbow Method

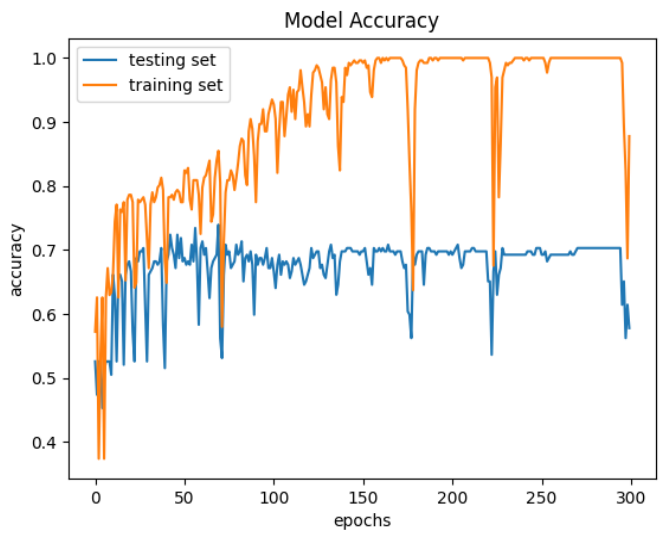
The elbow method is a test to determine the correct number of clusters, and we use it to determine the practicality of using the K-means algorithm with this data. For us, we want to obtain two clusters given our data, one yes cluster and one no cluster. The elbow method is done by plotting a graph of K vs. the standard square error. The "elbow" of the arm depicted in the graph is the point at which the ideal number of clusters is achieved.

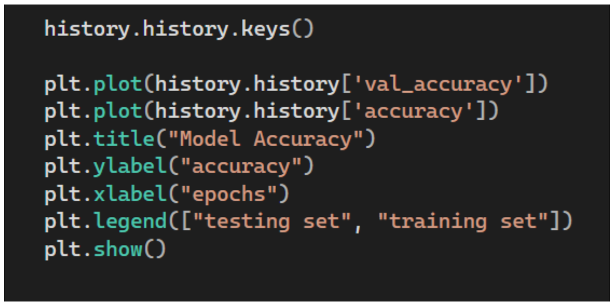


k=2 is the ideal number of clusters to use for the model, which confirms initial suspicion. We have good reason to believe that the K-means algorithm is a great tool to use to create these models, and analyze brain tumors. If this model continues to be improved, and training data increases, it can provide a long term solution to MRI image anomaly detection.

c) Convolutional Neural Network For Image Classification

Running the program yielded a testing dataset accuracy rate of about 70% (see below). Altering various parameters within the restrictions of the limited VRAM and dataset size yielded negligible benefits. It should be noted that outliers such as accuracy levels at ~175 and ~225 epochs were results of the low batch size, and are not present running on hardware allowing for large batch size.





```
history.history.keys()

plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title("Model Accuracy")
plt.ylabel("accuracy")
plt.xlabel("epochs")
plt.legend(["testing set", "training set"])
plt.show()
```

Analyzing these results we can now inspect them in the context of our four metrics: In terms of accuracy, the 70% accuracy rate places it about on par with the K-means algorithm, however more advanced implementations of this method have yielded near 100% accuracy rates[4] in the past, indicating that there is a significant amount of potential optimization that can be made (if hardware and dataset limitations were not present). Tuning and generalizability of this method are very straightforward, since changing parameters is relatively intuitive and the sub-classes of yes and no can be created by altering a few lines of code. The "explainability" of the CNN is more complicated, especially if low-level optimization is wanted, however this is simply the nature of convolutional neural networks and most advanced computer vision as well.

Conclusion

All three models share similar accuracy rates of 70-75%. To be more specific, the K-means clustering by image method achieved an accuracy of 75%, the K-means clustering by image feature achieved an accuracy of 72% for the pixel sum method and a 75% for the black and white method, and finally, the convolutional neural network achieved an accuracy rate of 70%. With high accuracy rates, we are able to conclude that these models, if tuned further, can accurately identify tumors from MRI scans.

Future Direction

To improve our performance results, we can look to increase the data set available for testing. With more scans to test our models, we can more accurately determine specific areas of improvement. Additionally, we hope that we can improve our models with more advanced technology. For instance, by creating models with a more advanced Graphics Processing Unit (GPU), we can develop our code to be more efficient and extensive so that it can cover finer details in each MRI scan.

These methods are generally easy to train/tune and, because each program scales well, has great generalizability. In addition, with the exception of the convolutional neural network method, each model is relatively easy to explain. As a result, health care professionals can easily adapt and use these or similar models in their workplace. With the development and application of our models into the medical field, we hope to accelerate the diagnosis time for brain tumors using MRI scans.

References

[1] Chakrabarty, N. (2019, April 14). *Brain MRI images for Brain tumor detection*. Kaggle. Retrieved August 4, 2022, from https://www.kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection
[2] "Sartaj". (2020, May 24). *Brain tumor classification (MRI)*. Kaggle. Retrieved August 4, 2022, from https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri
[3] Alphabet inc. (n.d.). *Tensorflow*. TensorFlow. Retrieved August 4, 2022, from https://www.tensorflow.org/
[4] Chattopadhyay, A., & Maitra, M. (2022, February 25). MRI-based brain tumour image detection using CNN based Deep Learning Method. Neuroscience Informatics. Retrieved July

28, 2022, from
https://www.sciencedirect.com/science/article/pii/S277252862200022X#:~:text=In%20our%20work%2C%20CNN%20gained,the%20result%20obtained%20so%20far

[5] Bien, N. (2018, February 9). *Don't just scan this: Deep learning techniques for MRI*. Medium. Retrieved August 5, 2022, from
https://medium.com/stanford-ai-for-healthcare/dont-just-scan-this-deep-learning-techniques-for-mri-52610e9b7a85

[6] Busch, H. von. (2019, June 6). *Artificial Intelligence for MRI*. Siemens Healthineers. Retrieved August 5, 2022, from
https://www.siemens-healthineers.com/magnetic-resonance-imaging/news/artificial-intelligence-for-mri.html

[7] Gupta, S. (2021, January 25). *Image clustering using K-means*. Medium. Retrieved August 5, 2022, from
https://towardsdatascience.com/image-clustering-using-k-means-4a78478d2b83?gi=c7160b685ba0

[8] Mishra, V. (2019, January 9). *K means algorithm explained with an example*. Medium. Retrieved August 5, 2022, from
https://medium.com/@7vaibhavmishra/k-means-algorithm-explained-with-an-example-4ed3e2764ce1

[9] Piazza, G. (2018, April 17). *Artificial intelligence enhances MRI scans*. National Institutes of Health. Retrieved August 5, 2022, from
https://www.nih.gov/news-events/nih-research-matters/artificial-intelligence-enhances-mri-scans

[10] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011

[11] Franklin, S. J. (2020, January 2). K-means clustering for image classification. Medium. Retrieved July 28, 2022, from
https://medium.com/@joel_34096/k-means-clustering-for-image-classification-a648f28bdc47