Brain Tumor Detection CNN

Final Project Report
ECS 170 SS2 2023

Andrew Bender[1], Abhiram Borra[2], Arnav Kaushik[2], Britney Nguyen[1], and George Zavala[1]

[1]Department of Computer Science, University of California, Davis
[2]Department of Cognitive Science, University of California, Davis

September 13, 2023

1 Introduction

The objective of this project was to build a machine-learning model that would classify brain MRI scans for brain tumor detection. The goal was to learn how to build a Convolutional Neural Network that would provide the team with a basic understanding of the applications of artificial intelligence in healthcare. This model with sufficient modifications can be used to assist in a healthcare setting to detect the presence of brain tumors by assessing patterns in brain scans. This model can help to speed up the process of tumor detection in MRI scans.

Throughout the process of developing this project, the goal to learn about working with data, and how to design and optimize a Convolutional Neural Network was accomplished.

2 Background

2.1 Brain Tumors

Brain tumors can be classified into two types based on their activity status- benign tumors and malignant tumors. Benign tumors are tumors that are non-cancerous in nature [1]. These tumors are masses of cells that grow relatively slowly in the brain [1]. On the other hand, a malignant brain tumor is a cancerous growth in the brain[2]. These malignant tumors can grow rapidly and destroy brain tissue, causing either fatality or loss of bodily function [2]. At present, a staggering 1 million people in the United States are living with a brain tumor [3]. Out of these, 72% are supposed to be benign, and the remaining 28% are malignant [3]. The relative survival rate for people living with malignant tumors is only 35.7% [3]. Malignant brain tumors are estimated to result in about 18,990 deaths in 2023 [3].

2.2 Importance of our Model

As morbid as it sounds, brain cancer is estimated to be the 10th leading cause of death by cancer in 2023 across sexes and ages [3]. Hence, machine learning algorithms that utilize CNNs like this one become extremely important in assisting professionals to detect the presence of brain tumors definitively. Models like ours can extract quantitative information from diagnostic images, including complex patterns that can be extremely difficult for the human eye to detect or quantify [4]. Hence,

[1] "Benign Brain Tumour (Non-Cancerous)." 2017. Nhs.uk. October 20, 2017. https://www.nhs.uk/conditions/benign-brain-tumour/).

[2] "Malignant Brain Tumour (Brain Cancer)." NHS. 2019. https://www.nhs.uk/conditions/malignant-brain-tumour/.

[3] "Brain Tumor Facts." n.d. National Brain Tumor Society. https://braintumor.org/brain-tumors/about-brain-tumors/brain-tumor-facts/.

[4] Cè, Maurizio, Giovanni Irmici, Chiara Foschini, Giulia Maria Danesini, Lydia Viviana Falsitta, Maria Lina Serio, Andrea Fontana, Carlo Martinenghi, Giancarlo Oliva, and Michaela Cellina. 2023. "Artificial Intelligence in Brain Tumor Imaging: A Step toward Personalized Medicine." Current Oncology 30 (3): 2673–2701. https://doi.org/10.3390/curroncol30030203.

through the intersection of machine learning and biomedical imaging for oncology, we can create better and personalized medicine measures for patients in need [4].

2.3 Functioning of MRI Machines

The best way to detect brain tumors is through imaging the brain through MRI (Magnetic Resonance Imaging) scans. Magnetic Resonance Imaging is a non-invasive method of brain imaging that makes use of extremely powerful electromagnets which have the capability of creating a strong magnetic field that makes protons in the body to align with that field [5].

When a certain radio-frequency current is passed through tissue, protons in the tissue's atoms are stimulated and go out of equilibrium; this makes them pull in the opposite direction of the magnetic field of the electromagnets[5]. When the magnets are turned off, sensors in the MRI machine detect the energy released as the protons realign with the magnetic field [5]. The amount of energy released depends upon the tissue type and its density [5]. Hence, practitioners are able to get an imprint of the brain and view it in cross-sections.

2.4 Computer Vision

Computer Vision is a field of Artificial Intelligence that helps computers to obtain meaningful information from digital images, videos and other visual inputs [6]. Computer Vision involves using a plethora of data that is fed into machine learning algorithms; and these algorithms are then trained on this data to provide us with the desired output, which in this case is assessing the presence of brain tumors.

Machine learning utilizes algorithms that help us to enable a computer to teach itself information about visual data. A Convolutional Neural Network "looks" by splitting an image into pixels that are given labels [6]. It uses the label to perform convolutions and make predictions about what it is "seeing"[6]. It does this process over several runs or epochs and checks its accuracy after every run, learns from its mistakes, and uses what it has learned from previous runs to give a better accuracy rate in the preceding run[6].

In this case, the CNN is trained on brain scans which are divided into sets of having the quality of possessing a tumor versus not having a tumor at all. After learning what a brain MRI scan with a tumor versus a scan without a tumor looks like, it "looks" at the MRI scans and trains and gives predictions about the presence of a brain tumor in our provided brain MRI scans.

3 Methodology

This project involved a deep understanding of the workings of Convolutional Neural Networks and how they can be utilized to do computer vision to classify whether tumors are present in the brain or not. Google Colab was used to create this project. The model was coded in the coding language Python.

3.1 Dataset

The dataset for this model titled "Brain Tumor" was taken from the website Kaggle. The dataset used was created by Jakesh Bohaju. The DOI citation for the dataset is 10.34740/kaggle/dsv/1370629. There are 3762 images from Brats2015 brain tumor with different MRI slices of the brain. A file titled *BrainTumor.csv* includes information about the images, stating five first-order features and eight texture features with the target level.

[5] "Magnetic Resonance Imaging (MRI)," National Institute of Biomedical Imaging and Bioengineering, accessed September 11, 2023, https://www.nibib.nih.gov/science-education/science-topics/magnetic-resonance-imaging-mri.

[6] IBM. n.d. "Computer Vision." IBM. https://www.ibm.com/topics/computer-vision.

3.2 Computer Vision

3.2.1 Libraries and Data Preprocessing

The following libraries were used for the models :

1. numpy: Imported as np to do numerical and array operations
2. tensorflow: Imported as tf to build and train our neural network
3. seaborn: Imported to supplement matplotlib to visualize data and create statistical graphics
4. pandas: Imported to do data manipulation (reading the csv file)
5. matplotlib: Imported to plot and visualize the MRI scans
6. os: Imported as an operating system to mount the dataset on the Google Drive
7. cv2: Imported to help with image processing

After importing the data into Google Colab, the labeling columns from the *BrainTumor.csv*file were dropped so data could be cleaned to be used accurately.

Finally, the brain scan images from the drive were put into the array *images_data* using Keras.

3.2.2 Data Processing

All the labels from the 'Class' column were stored in a numpy array to perform data manipulations on the file.

The brain scans and the *BrainTumor.csv* file were stored in a Google Drive folder and a variable path was set in order to access it. The images were added to a numpy array in sorted order so it would correspond to the labels. Every image was stored in image_arrays and was represented with the numbers 0 or 1, where 0 represented the color black and 1 represented the color white.

Finally, brain scan data and its classification data were split into training (15000 items), validation (15000 items), and test (762 items) sets.

3.2.3 CNN Model 1

Construction In CNN model 1, the type of model used was a Sequential type model which was constructed using Keras. Keras was also utilized in order to create the layers. Here is the description of each layer:

1. Input Layer: Convolutional 2-D Layer that takes the input from all the pixels in the image which happens to be 200x200, giving us 40,000 gray-scale values between 0 and 1.
   (a) Number of Nodes: 16
   (b) Size of Filter: 3x3
   (c) Activation Function: ReLu
2. Max Pooling Layer I: Decreases the dimensions of the image by half while keeping the important details.
   (a) Size of Filter: 2x2
3. Hidden Layer I : Takes input from the input layer and feeds it into the 32 perceptrons within the layer.
   (a) Number of Nodes: 32
   (b) Size of Filter: 3x3
   (c) Activation Function: ReLu
4. Max Pooling Layer II: Decreases the dimensions of the image by half while keeping the important details.

(a) Size of Filter: 2x2

5. Hidden Layer II : Takes input from the previous layer and feeds it into the 16 perceptrons within the layer.
   (a) Number of Nodes: 16
   (b) Size of Filter: 3x3
   (c) Activation Function: ReLu
6. Max Pooling Layer III: Decreases the dimensions of the image by half while keeping the important details.
   (a) Size of Filter: 2x2

   Note: The Model was flattened after this layer by using the *Flatten()* function which turns the pixels into an array.
7. Fully-Connected layer: Takes the input from the previous layer and processes it in 256 perceptrons.
   (a) Number of Nodes: 256
   (b) Activation Function: ReLu
8. Output Layer: Takes the input from the fully-connected layer and processes it in a single perceptron to provide the final output through the sigmoid function
   (a) Number of Nodes: 1
   (b) Activation Function: sigmoid

   Note: These are the descriptions of the Activation Functions:
   (a) ReLu[7] : Introduces non-linearity in the model. It essentially takes into input a weight and based on a certain threshold, the ReLU function will either output 0 or the weight itself.
   (b) Sigmoid [7] : Used for the output layer to output a single node, which is either a value of 1 (indicating a tumor found) or 0 (indicating a tumor not found). Takes the sum of weights plus the bias from the previous fully connected layer and output a 0 or 1 based on the sigmoid function. By default, the sigmoid function is centered at 0.5

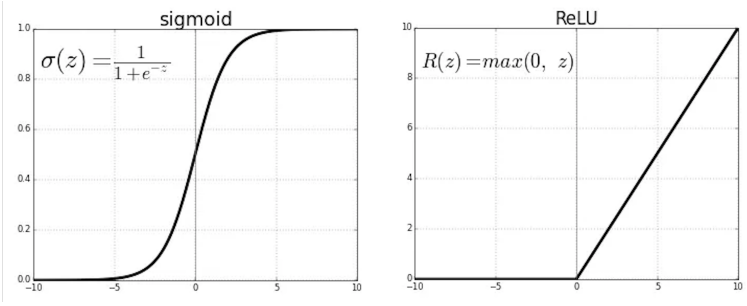

Figure 1: Visualization of ReLu and Sigmoid Functions

[7] "Activation Functions in Neural Networks." Medium. Towards Data Science. September 6, 2017. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

Hyperparameters The hyperparameters for CNN Model I were defined randomly.

1. Optimizer: Adam
2. Loss Function: Binary Cross Entropy
3. Hidden layers: 2
4. Nodes: 16 or 32
5. Activation functions: ReLu
6. Learning rate: 0.01 (Default)

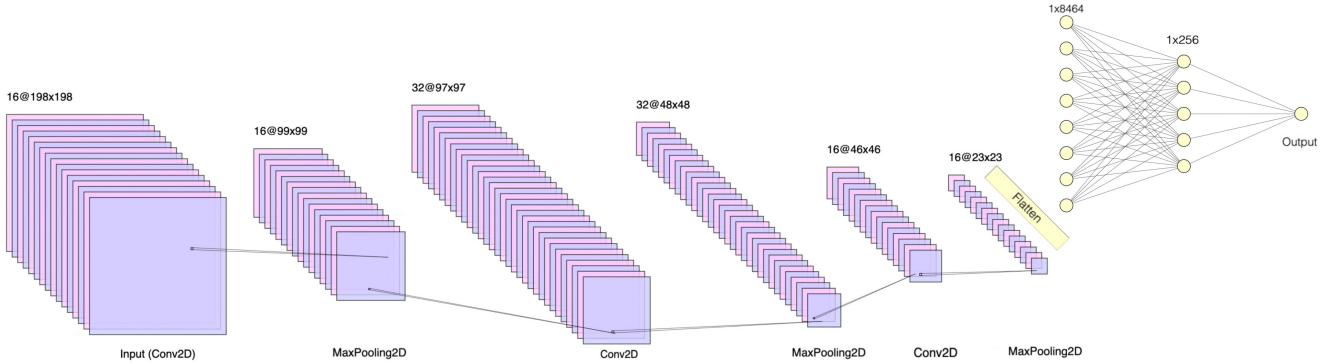

Figure 2: Visualization of CNN Model 1

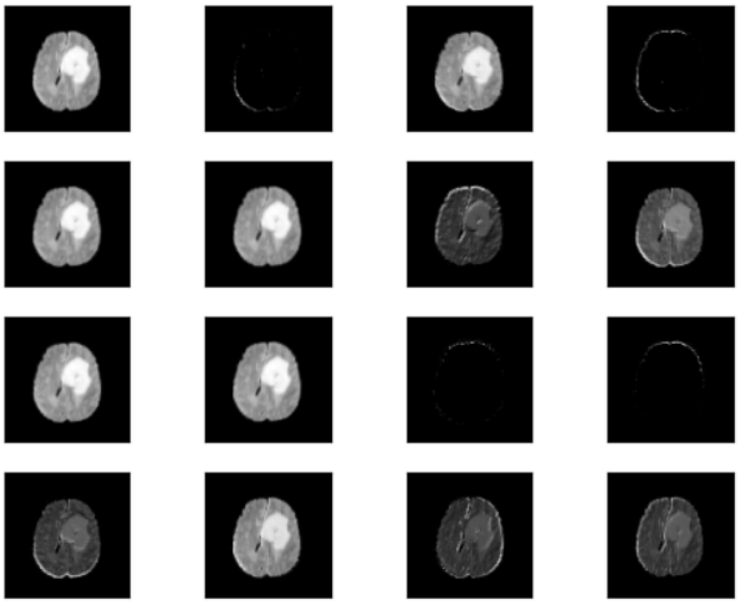

Figure 3: Visualization of an input image passing 16 filters in Hidden Layer I in CNN Model I

3.2.4 Final Model

Construction In the final model, the type of model used was a Sequential type model which was constructed using Keras. Keras was also utilized in order to create the layers. Here is the description of each layer:

1. Input Layer: Convolutional 2-D Layer that takes the input from all the pixels in the image which happens to be 200x200, giving us 40,000 gray-scale values between 0 and 1.
   (a) Number of Nodes: 16
   (b) Size of Filter: 3x3
   (c) Activation Function: ReLu
2. Max Pooling Layer I: Decreases the dimensions of the image by half while keeping the important details.
   (a) Size of Filter: 2x2
3. Hidden Layer I : Takes input from the input layer and feeds it into the 32 perceptrons within the layer.
   (a) Number of Nodes: 32
   (b) Size of Filter: 3x3
   (c) Activation Function: ReLu
4. Max Pooling Layer II: Decreases the dimensions of the image by half while keeping the important details.
   (a) Size of Filter: 2x2
5. Hidden Layer II : Takes input from the input layer and feeds it into the 32 perceptrons within the layer.
   (a) Number of Nodes: 32
   (b) Size of Filter: 3x3
   (c) Activation Function: ReLu

   Note: The Model was flattened after this layer by using the *Flatten()* function which turns the pixels into an array.
6. Output Layer: Takes the input from the second hidden layer and processes it in a single perceptron to provide the final output through the sigmoid function.
   (a) Number of Nodes: 1
   (b) Size of Filter: 3x3
   (c) Activation Function: sigmoid

   Note: The activation functions used in the Final model served the same purpose as in CNN Model 1 (refer to section 3.2.3)

Hyperparameters In order to design the best model, the hyperparameters had to be changed and tested. The Keras classifier was implemented at first but was found to be sub-optimal. A Keras tuner was implemented in order to run through all the possible hyperparameters and receive the most accurate set. The hyperparameter "tuner" was fairly slow but it returned an optimal set of parameters to use. The following were found to be the best set of parameters:

1. Optimizer: Adam
2. Loss Function: Binary Cross Entropy
3. Hidden layers: 2
4. Nodes: 32
5. Activation functions: ReLu
6. Learning rate: 0.001

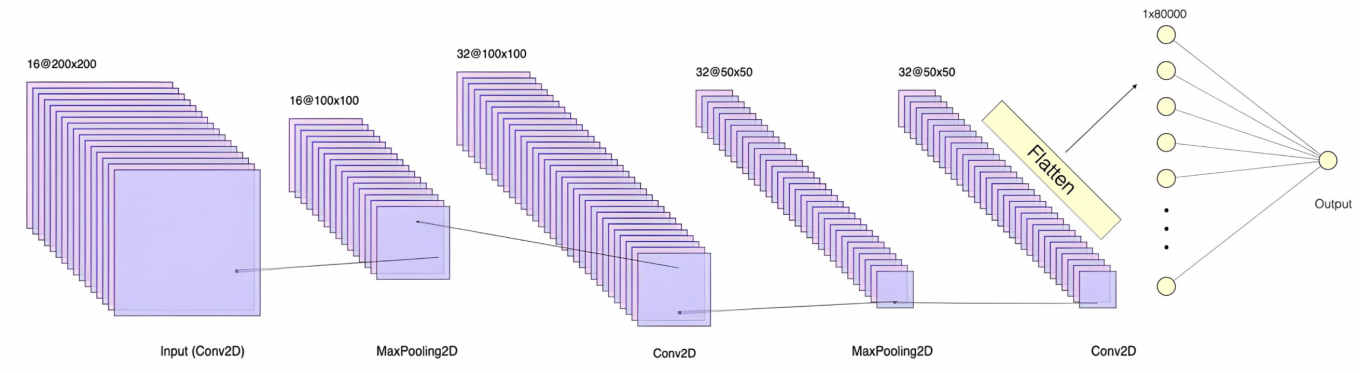

Figure 4: Visualization of the Final CNN Model

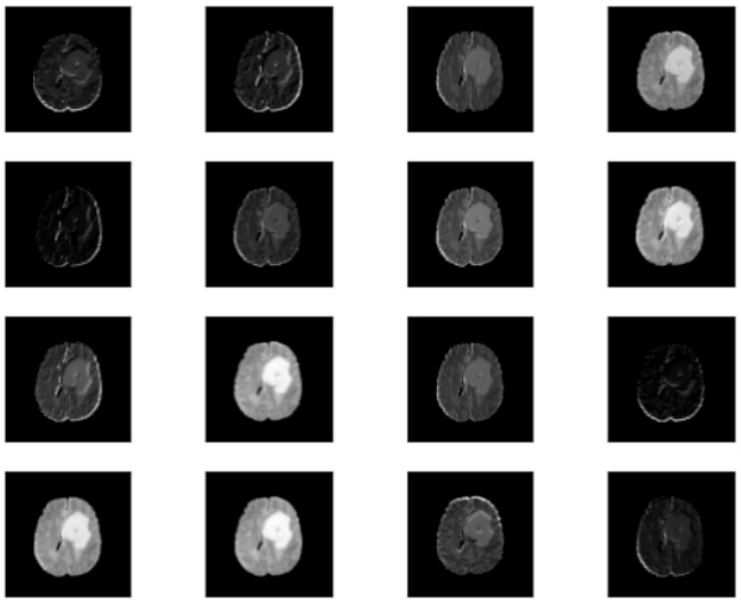

Figure 5: Visualization of an input image passing 16 filters in Hidden Layer I in the Final Model

4 Results

4.0.1 Performance of CNN Model 1

The initial convolutional neural network (CNN) model was developed with a set of randomized hyperparameters. Upon evaluation of its training performance, the model yielded an accuracy rate of 96% and a loss rate of 7.84%. When assessed against the validation dataset, the accuracy was observed to be 91.6% while the loss rate rose to 23.36%. However, a significant disparity was noted when the model was subjected to the testing dataset: it achieved a mere 34.27% accuracy. This considerable drop in the testing accuracy highlights potential overfitting or other underlying issues that warrant further investigation.

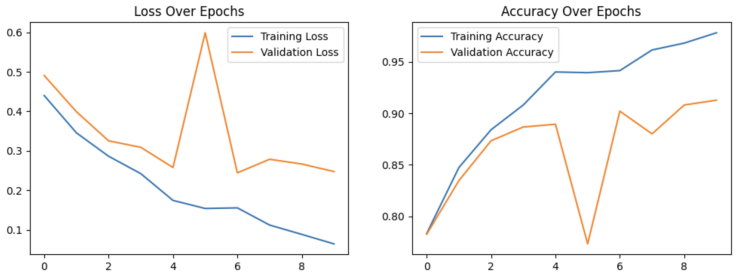

Figure 6: Graphical Representation of Loss and Accuracy over Epochs for CNN Model 1

	precision	recall	f1-score	support
Class 0	0.80	0.89	0.85	258
Class 1	0.94	0.89	0.91	504
accuracy			0.89	762
macro avg	0.87	0.89	0.88	762
weighted avg	0.89	0.89	0.89	762

Figure 7: Visualization of the Classification Report for CNN Model I

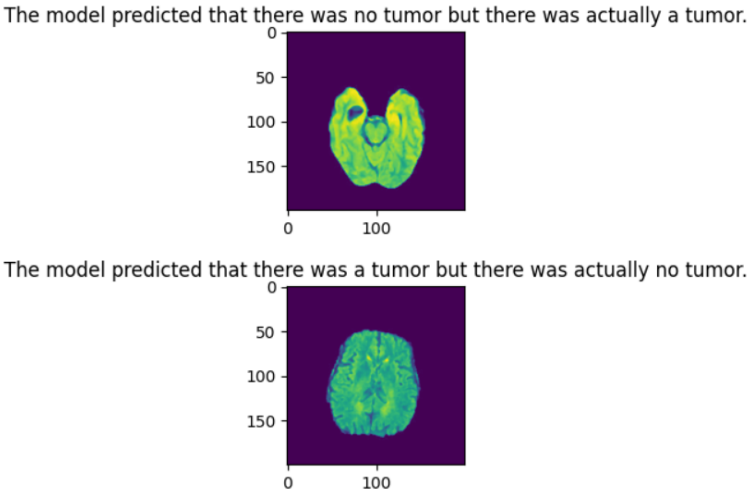

Figure 8: Misclassification Examples for CNN Model 1

4.0.2 Results from Hyperparameters Tuning

Incorporating the derived values into the final model resulted in an accuracy rate of 100% for the training dataset. During the training phase, a loss rate of 0.0092 was observed. For the validation dataset, the model reported a loss rate of 0.2543 and an accuracy of 92.67%.

4.0.3 Performance of Final Model

Upon the integration of the identified hyperparameters, the model exhibited a notable increase in performance. It achieved a 96% accuracy in correctly identifying instances with a brain tumor and a 74% accuracy for instances without a brain tumor.

The overall accuracy of the model on the test set was 87%, based on 762 samples. The weighted average values for precision, recall, and the F1-score were 0.89, 0.87, and 0.88 respectively.

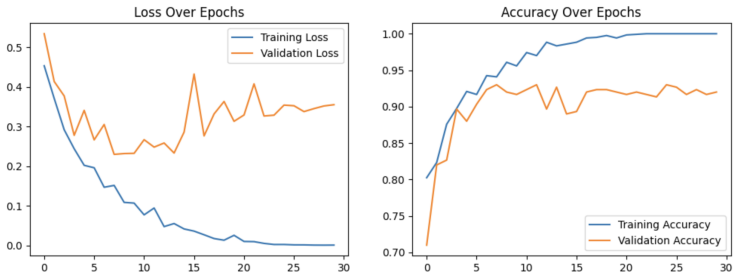

Figure 9: Graphical Representation of Loss and Accuracy over Epochs for the Final Model.

	precision	recall	f1-score	support
Class 0	0.77	0.91	0.83	268
Class 1	0.95	0.85	0.90	514
accuracy			0.87	782
macro avg	0.86	0.88	0.87	782
weighted avg	0.89	0.87	0.88	782

Figure 10: Visualization of the Classification Report for the Final Model

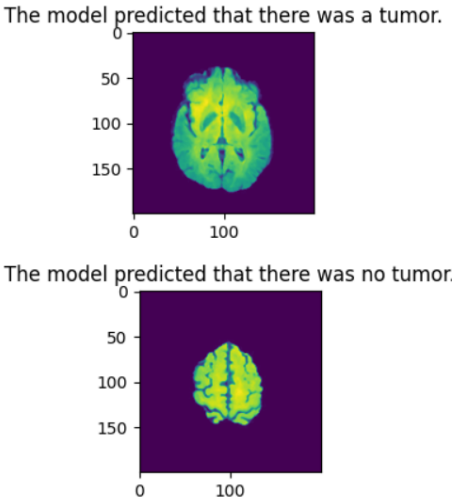

Figure 11: Accurate classification Examples for Final Model

5 Discussion

The first CNN model that was constructed provided sub-optimal results; and hence, the methodology to approach building a CNN had to be altered. A new model was built that served as our final model. This model was then run and hyperparameters were tuned; this helped to create a better model with better rates of accuracy.

While the model demonstrates high accuracy in identifying the presence of brain tumors, it concurrently reflects a reduced accuracy in cases where tumors are absent. This observation can be perceived from dual perspectives. Firstly, a false positive result may cause emotional distress for the patient and may potentially misguide medical professionals. Conversely, when dealing with critical diagnoses such as brain tumors, a false positive is arguably preferable to a false negative. Adjusting the model to minimize false positives might inadvertently compromise its ability to accurately detect tumors. Given the significant consequences associated with missed brain tumor diagnoses, maintaining the model's current orientation was deemed essential, even after looking at the trade-off.

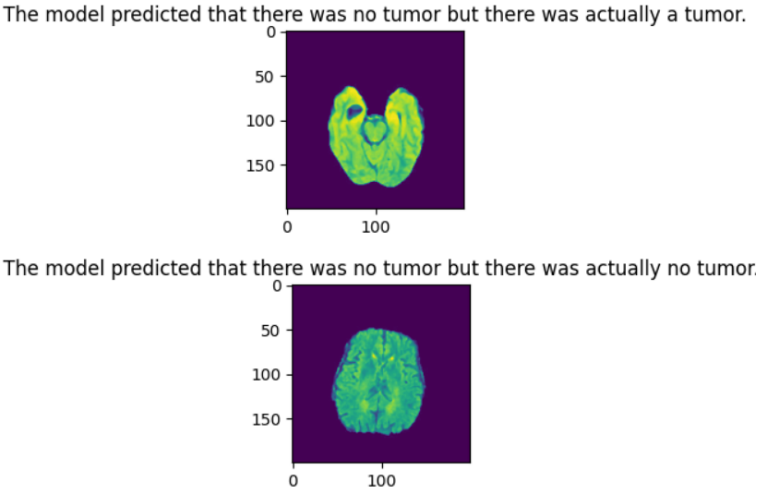

Figure 12: Misclassification Examples for Final Model

6 Conclusion

Constructing this model was a challenge, but our team was successfully able to implement a CNN model to predict the presence or absence of a brain tumor using images of MRI scans with considerable accuracy.

This model can be further potentially be tuned and optimized to work for another dataset that provides data that is split based on different types of tumors with different features (varied location, size, and severity). If trained with a bigger and more generalizable dataset, it can also be possibly used in the fields of healthcare, However, we must proceed with caution while using machine learning models in healthcare– a patient's life should not be put on risk by using a CNN model as the only diagnostic method for tumors. This model should only be used as as supplemental aid for highly trained and skilled professionals for the safest use.

7 Contributions

7.1 Andrew

Helped with preprocessing and visualization tools for our neural network. Created images that show the functionality of individual filters in the hidden layers. Also helped to create filter graphics for the final report.

7.2 Abhiram

Developed the Hyperparameter tuning algorithm and contributed to building the final model based on the optimal hyperparameters generated.

7.3 Arnav

Prepared the final report (write-up and creating graphics) in LaTeX, and prepared the presentation slides. Did background research about tumors, datasets for tumors, and technical information about tumor classification models.

7.4 Britney

Stored the data and matched them to their labels. Extracted accurate and misclassified examples from the models and helped debug the hyperparameter turning algorithm. Also cleaned up and edited the majority of the Colab, updated the Github, and assisted with the final project write-up.

7.5 George

Contributed by implementing a CNN on my own which came with its problems due to the mismatching data. Helped with the data processing as well as the early stage of the first CNN.

8 GitHub

Here is the link for the GitHub titled "BrainTumorMRIDetectionAI" that includes the code and slides for this project:

https://github.com/Bnguy0926/BrainTumorMRIDectectionAI