Brain Tumor Classification

George Larres - *Data Preprocessing, Model Creation*
David Thompson - *Model Testing, Report*
Jason Whitlow - *Model Creation*

Abstract

Brain tumors are abnormal cell masses that can significantly impact brain function, making early and accurate diagnosis critical. Magnetic Resonance Imaging (MRI) is a diagnostic tool used to detect brain tumors. This study uses the Brain Tumor MRI Dataset, a selection of 7,023 grayscale MRI images categorized into four classes: glioma, meningioma, pituitary tumor, and no tumor. Two custom CNN models were developed to classify MRI images into these categories, utilizing feature extraction techniques and addressing challenges such as image variability, resizing effects, and equipment inconsistencies. Both models did really well with accuracy at 99% and recall at 46%.

Introduction:

A brain tumor is a mass of abnormal cells that grows inside of the brain. Brain tumors can be classified into different types depending on the type of cells and where they are in the brain. Glioma tumors are malignant and form in the brain or spinal cord from glial cells. Meningioma tumors develop from the meninges which are the protective layers surrounding the brain and spinal cord. A pituitary tumor is an abnormal growth of cells in the pituitary gland. Both meningioma and pituitary tumors are typically benign, but can be malignant.

Brain tumors can be recognized by trained medical professionals using Magnetic Resonance Imaging (MRI). It is also possible to recognize brain tumors by taking MRI's and feeding them through a Convolutional Neural Networks (CNN). Using neural networks can improve diagnostic accuracy, speed and efficiency. It can also help in earlier detection and provide assistance to medical professionals when facing complex cases.

CNNs are a type of deep learning architecture that is capable of performing the task of image classification. The purpose of a CNN is to extract features from data. This is achieved through applying kernels across the layers of the neural network. In CNNs, each layer learns progressively more abstract features. Pooling, unpooling, batch normalization, dropout, and flattening are all additional operations that can be applied to the model structure to improve feature learning. More layers allow the network to model complex relationships in data, leading to better performance on challenging tasks. Data preprocessing is critical in CNNs (and in machine learning in general) to ensure good performance and accurate predictions. The quality of the input data directly affects the network's ability to learn and generalize.

This study uses the **Brain Tumor MRI Dataset**, a collection of MRI scans designed for research in tumor detection and classification. The dataset combines images from the **Figshare**, **SARTAJ**, and **Br35H** datasets, comprising a total of 7,023 grayscale MRI images categorized into four classes: glioma, meningioma, pituitary, and no tumor. MRIs capture variations in tissue density and other physical properties as single-intensity grayscale images, making them suitable for CNN-based analysis.

Image 1: No Tumor

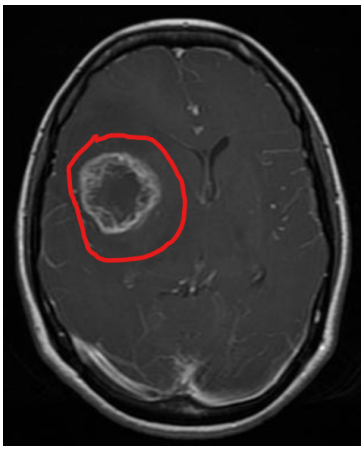


Image 2: Glioma Tumor

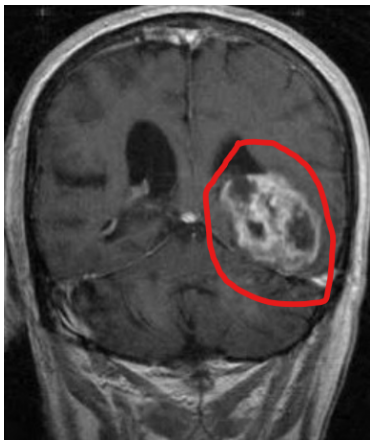


Image 3: Meningioma

Image 4: Pituitary Tumor (Top View)

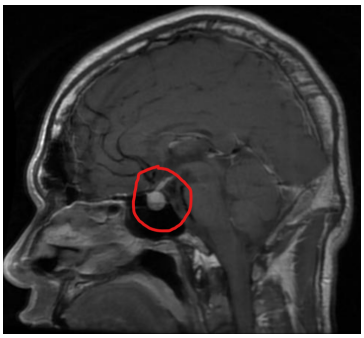


Image 5: Pituitary Tumor (Side View)

MRIs capture variations in tissue density and other physical properties using electromagnetic signals. These are naturally represented as a single intensity value per pixel, which is why the resulting images are grayscale. However, using CNNs for brain tumor detection poses several challenges. Tumors may appear different depending on the viewing angle (e.g., top, side, or rear), and resizing images for model training can lead to the loss of critical features. Additionally, variations in equipment and imaging protocols across datasets introduce inconsistencies that the model must overcome to generalize effectively.

Accuracy, Recall, and F1 score all analyze the effectiveness of a classification model. In medical applications, **false negatives** (failing to detect a tumor when one is present) are far more serious than **false positives** (flagging a tumor when there isn't one). Recall is the measure of a model's ability to accurately classify true positive cases. A missed diagnosis can delay treatment, potentially worsening the patient's condition or even becoming life-threatening. Therefore, maximizing Recall ensures that as many true tumor cases as possible are identified. When training a CNN for brain tumor detection, Recall should be a key focus because of its alignment with the critical need to minimize missed tumor diagnoses. However, the model should still aim for reasonable Precision and other balanced metrics to ensure practicality in a clinical setting.

Methodology:

This project uses the pytorch library to create a CNN that will classify the 4 types of brain images. Several models were tested when building the CNN. Ultimately 2 models were chosen for their high accuracy. Hyperparameter tuning was an important part of the testing process. Accuracy would vary between 20-40% depending on the chosen hyperparameters. Initially we struggled with improving accuracy as it began at less than 1 percent. Through modifications of hyperparameters such as batch size, learning rate, learning algorithm, and epochs we were able to improve the accuracy to above 20 percent. While this was a significant improvement it was still worse than randomly selecting one of the 4 classes. We believe some of the incorrect classification was being caused by an issue with the preprocessing of the image files. To resolve this issue, we decided to normalize the images using the torchvision normalize function which improved the performance significantly. This change and experimenting with different hyperparameters allowed our model to generalize well.

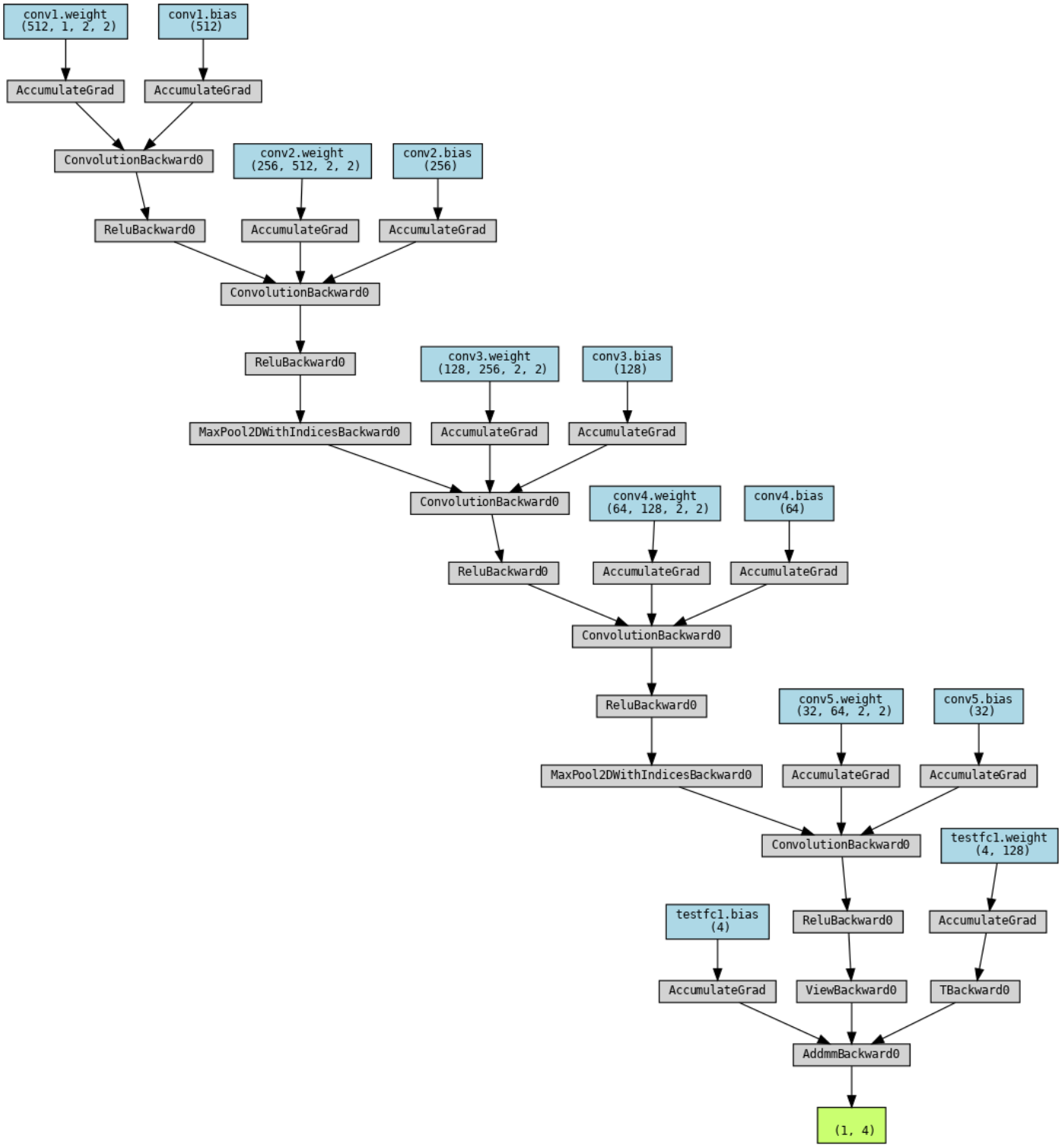
Our main focus during the initial stage of the project was improving accuracy as much as possible. Our model used a kernel size of 3 initially, with a stride of 2 and the number of in channels and output channels decreasing (downsampling) in each of the convolutional layers. The model also had a learning rate of 0.001, used the Adam optimizer, and had several fully connected layers after convolution. We began training our models with a small number of epochs, usually around 10-15 to get an idea of how well the model was performing and to notice any trends in the performance after each epoch. The model had a very low accuracy and performance was static, which made us believe the model was getting stuck, so we increased the learning rate. Changing the learning rate did not improve the performance of the model. We then changed the optimizer from Adam to Stochastic Gradient Descent, which improved the model's performance but it was still not performing well. After getting out of the astonishingly low accuracy slump the model was in, which we believed was being caused by a loss of information in the image and incorrect preprocessing, this made us decrease the kernel size to 2 in our convolution layers and redo our image preprocessing file. We began increasing the number of epochs in increments of 10. As we experimented and noticed the uptrend in the accuracy and the improvement of the loss, we settled on 50 epochs since it was fast to train and gave us an incredibly high accuracy and a low loss. Multiple convolutional layers were also used in order for the model to extract more features from the images. We started with 3 layers initially, using ReLU non-linear activation functions and a max-pool for each convolutional layer. Due to the sub-par results we were getting, we then experimented with using a max-pool only on the first and third convolutional layers which increased accuracy and lowered the loss. With this, we decided to add 2 more convolutional layers for a total of 5 layers and then applied a ReLU on every convolutional layer and a max-pool to the second and fourth convolutional layers after applying ReLU. This led to a significant performance leap while still being very efficient (usually taking 10-20 seconds per epoch).

Preprocessing included several stages including image resizing, grayscale conversion, tensor conversion and normalization. Grayscale images were used as they provide sufficient information for feature extraction while only using a single channel. The images were then resized to 224 x 224 pixels to help with generalization. Since some of the images were smaller than 224 x 224, the images were upscaled automatically. Pixel values were converted to the tensor range of [0,1] and additionally normalized to have a mean of 0.5 and standard deviation of 0.5. This in theory makes the images less sensitive to variations in lighting or contrast. The preprocessed images were then converted to pickle files with both a training and testing dataset.

Model 1:

Convolutional Layers: 5
Pooling Type: Max Pooling
Fully connected Layers: 1
Activation Function: Relu
Batch Size: 1

Learning Rate: 0.001
Optimizer: Stochastic Gradient Descent
Loss Function: Cross Entropy
Kernel: Size=2, Stride=2, Padding=1



Model 2:

Convolutional Layers: 4
Pooling: Max Pooling
Fully connected Layers: 2
Activation Function: Relu
Batch size: 1

Drop Out: Yes
Learning Rate: 0.001
Optimizer: Stochastic Gradient Descent
Loss Function: Cross Entropy
Kernel: Size=3, Stride=1, Padding=1

Convolutional layers were added and removed from Model 1 while monitoring accuracy to find this finalized structure. All hyperparameters listed above were adjusted to find a suitable parameter. Multiple convolutional layers were used to extract local spatial patterns within the image, allowing the neural network to identify features like edges, textures, and shapes. For detecting tumors in our dataset,

multiple convolutional layers were an obvious choice. This model also uses the Adam optimizer instead of Stochastic Gradient Descent which, through our experiments, we found was able to achieve a higher accuracy and lower loss when training and during validation.

Model 2 followed a similar methodology as Model 1. The main difference between the models is the number of convolutional and fully connected layers. In Model 1 we reduced the number of fully connected layers to 1 layer in case the performance was less stable because the model was overfitting or because of the decrease in spatial awareness of the model.

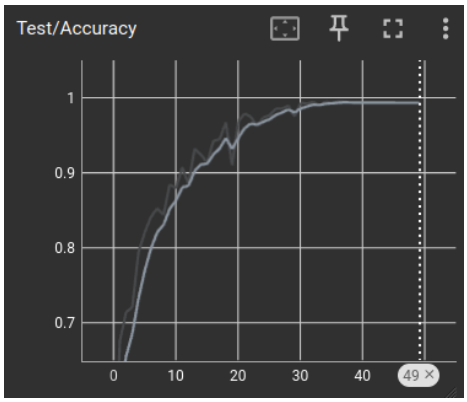
Results:

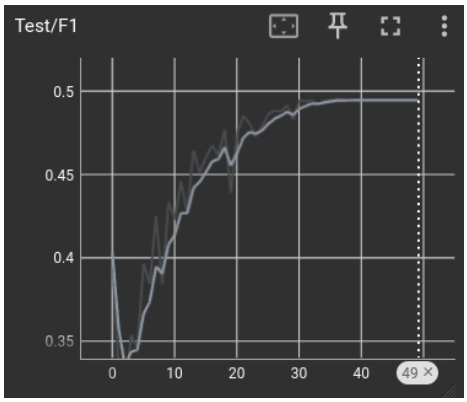
The end results of the two models were nearly identical, however upon examination, from the graphs below, it is clear that Model 2 is not consistent over the epochs.

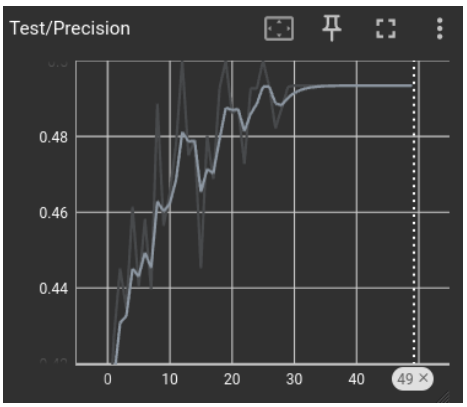
Performance Metrics	Accuracy	Recall	Precision	Specificity	F1 Score
Model 1	99.10%	48.70%	50.00%	62.40%	49.30%
Model 2	99.20%	48.70%	50.00%	62.50%	49.30%

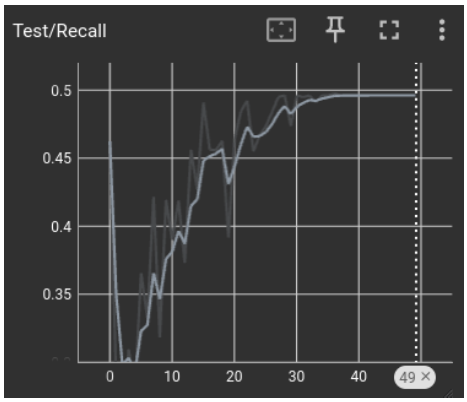
After model and hyperparameter tuning Model 1 achieved a test accuracy of 99.09% and a recall of 48.7%. Through testing it was found that learning rate, number of epochs, and optimizer are all important contributors to overall model performance. Model 2 achieved a test accuracy of 99.20% and a recall of 48.7%.

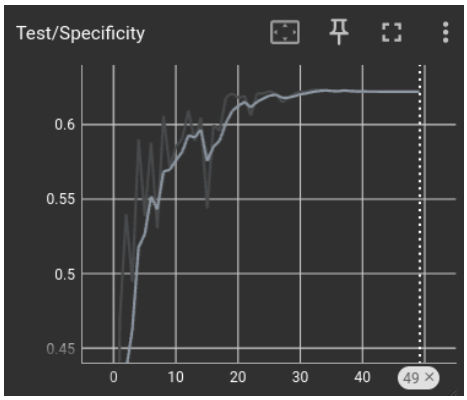
Model 1 Performance Metrics

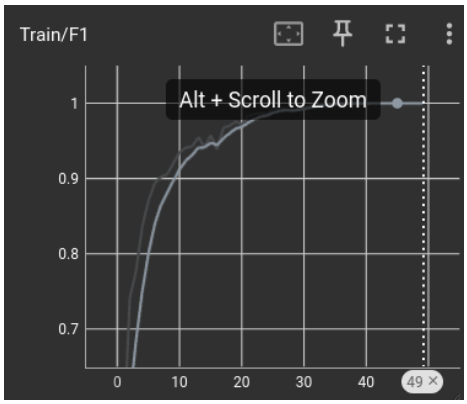


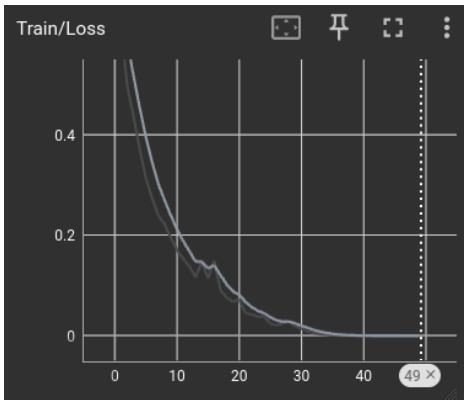


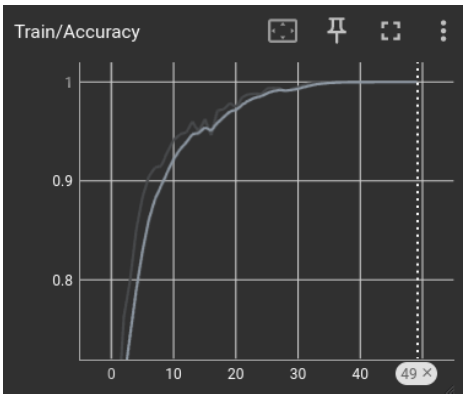


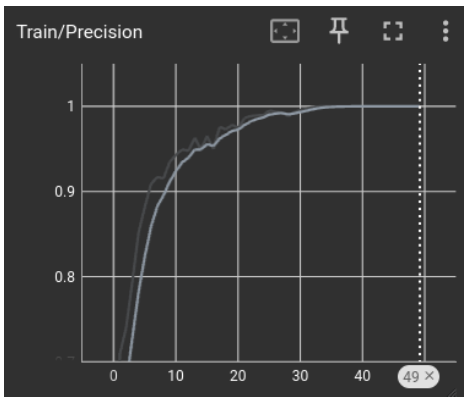


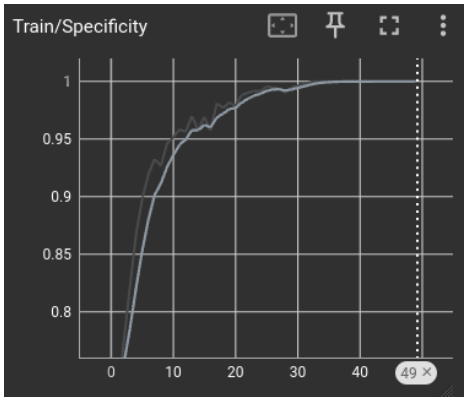


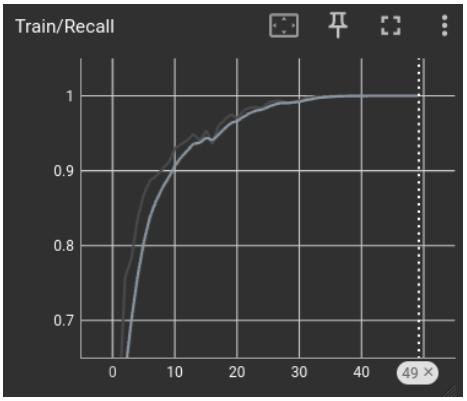




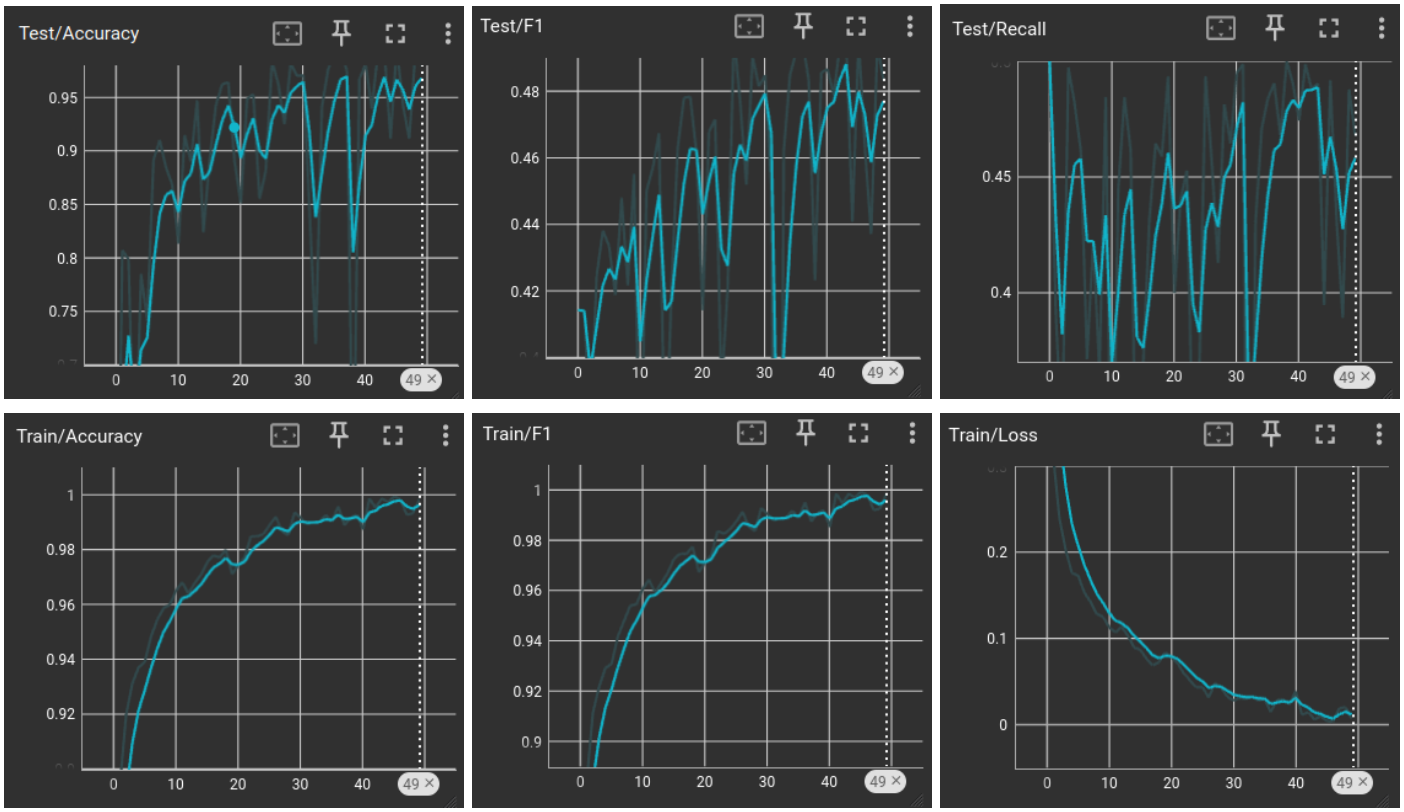








Model 2 Performance Metrics



It seems that the difference in convolutional layers, fully connected layers, and choice of training algorithm did not have a significant effect on the models. Additionally the initial learning rate did have an effect on model tuning with model 1 and model 2 having learning rates near optimal at 0.001 and 0.0001 respectively. However, moving too far outside of this range did cause instability in some cases as seen in the above Model 2 graphs. Preprocessing was a large contributor in the final results. Upscaling the images using the opencv resize function played a significant role in improving our results.

Both models performed with high accuracy and low recall. This indicates that the majority of classes are being correctly classified, but for some reason there is a large portion of false negatives that are being classified by the model. Further hyperparameter tuning was attempted to increase overall recall, but it was optimized at around 49%. Given that preprocessing had such a large effect on model performance, this is likely due to the different profile the images came in. It is possible that the model is correctly classifying a given profile and not others. As hyperparameter tuning did not effectively increase recall, further model improvement would be needed to improve this metric.

As previously mentioned, the datasets contain MRIs that were taken with top, rear and side profile. In the future, classification of the images into these categories could improve recall of the CNN, however this may also lead to the need for more images in the dataset.

Conclusion

In summary, this study used two custom CNN models to detect and classify brain tumors. The models successfully classified the images with 99 percent accuracy and 48 percent recall, with model 1 being the most stable. Model structures were adjusted and hyperparameters were tuned to achieve these results. Preprocessing was an important step in the process and it was shown that more complex models need a higher resolution to accurately classify images and further preprocessing would likely further improve recall. Overall the models performed to a moderate standard of success. These results show the potential for AI to help in the medical field. For future works we could focus on expanding these models on larger and more diverse datasets to see how well they perform.

References

Msoud Nickparvar. (2021). Brain Tumor MRI Dataset [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DSV/2645886

Github Repository

https://github.com/Nerdeee/Brain-Tumor-Classification