Brain Tumor Classification

Arjun Gill, Hugo Gomez, Bryan Lee, Demetrio Calimag

Table Of Contents

Project Links

Main Repository:

https://github.com/BryanL43/BrainTumorClassification

Kaggle Dataset:
https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset

Presentation Demo Repository:
https://github.com/mosguinz/sfhacks25

Project Demo Link (login with fake email & password, then navigate to “Brain Results”):
https://dashboard.guacamolerigatoni.com/

Abstract

A brain tumor is an abnormal cell growth that can occur in any part of the brain or skull. There are over 120 types of brain tumors. The 5-year survival rate for those with cancerous brain or CNS tumors is about 34% for men and 36% for women. Given the high mortality rate, early detection and treatment are critical for improving outcomes. MRI is the best method for detecting tumors, but manual analysis by radiologists is error-prone due to the complexity of brain tumors. Convolutional neural networks (CNNs) are the most common deep learning algorithm for visual learning and image recognition. In our approach, we built an image classification model using the EfficientNet CNN architecture to classify brain tumors from MRI images. The proposed approach achieved an astounding 98.63% accuracy (F1-score) on the test data. With such high accuracy, the neural network architecture can be a valuable tool in brain tumor diagnosis.

1. Introduction

When deciding on a topic for our project, our group collectively agreed that it should be meaningful—something that resonated with us on a personal level. During our discussion, one of our group members shared a deeply personal experience: a close family member had been diagnosed with an early-stage tumor that progressed slightly due to a delayed diagnosis. This experience inspired them to propose developing a model that could help identify different types of tumors quickly and accurately. The rest of the team was immediately drawn to the concept, recognizing its emotional significance and practical impact in the medical field.

After further consideration, we decided to narrow the scope of our model. Since tumors can occur throughout the body and vary significantly in type, we chose to focus specifically on brain tumors. This decision allowed us to concentrate our research on deep learning and model training on a well-defined yet medically critical area. Brain tumors present unique diagnostic challenges, and developing a model to aid in their identification felt like a meaningful and impactful contribution to the field. With this focus, we aimed to design a model that could support faster diagnosis and potentially assist healthcare professionals in making informed decisions.

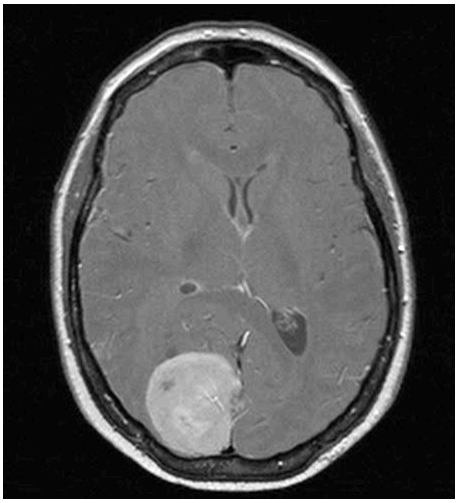

Figure 1: A sample MRI image of a brain tumor.

2. What We Set Out to Achieve

Our goal for this project was to develop an efficient and accurate EfficientNet CNN model capable of identifying different types of brain tumors using MRI scan data. We set out to create a model with at least 85% accuracy in classifying the four most common and labeled types of brain tumors: Glioma, Meningioma, Pituitary, and the absence of any tumor (no tumor). Each of these tumors presents unique characteristics in MRI scans, such as size, shape, and location. Deep learning techniques can effectively use these for classification models. An important characteristic that does not affect how the model functions or its purpose is determining malignancy.

Pituitary tumors illustrate this well, as they are typically non-cancerous. With that in mind, we designed the model to identify the type of tumor and determine whether it is present. Early diagnosis is often critical in determining a patient's treatment plan and potential outcomes. Our model and its high accuracy can be of great use to medical professionals who spend valuable time on tumor classification. We also aim to increase the model's accuracy and practical relevance by narrowing our focus to brain tumors, laying the groundwork for future expansion and purpose.

3. Methodology

Our project follows a structured setup process, beginning with the dataset, followed by image preprocessing, the transformation pipeline, and the proposed model. Initially, we incorporated an image augmentation step to address class imbalance in the original dataset. However, with the introduction of a new, balanced dataset, augmentation was no longer necessary and posed a risk of severe overfitting. In the context of medical imaging, it is generally advised to preserve the original data without transformation in order to maintain diagnostic integrity.

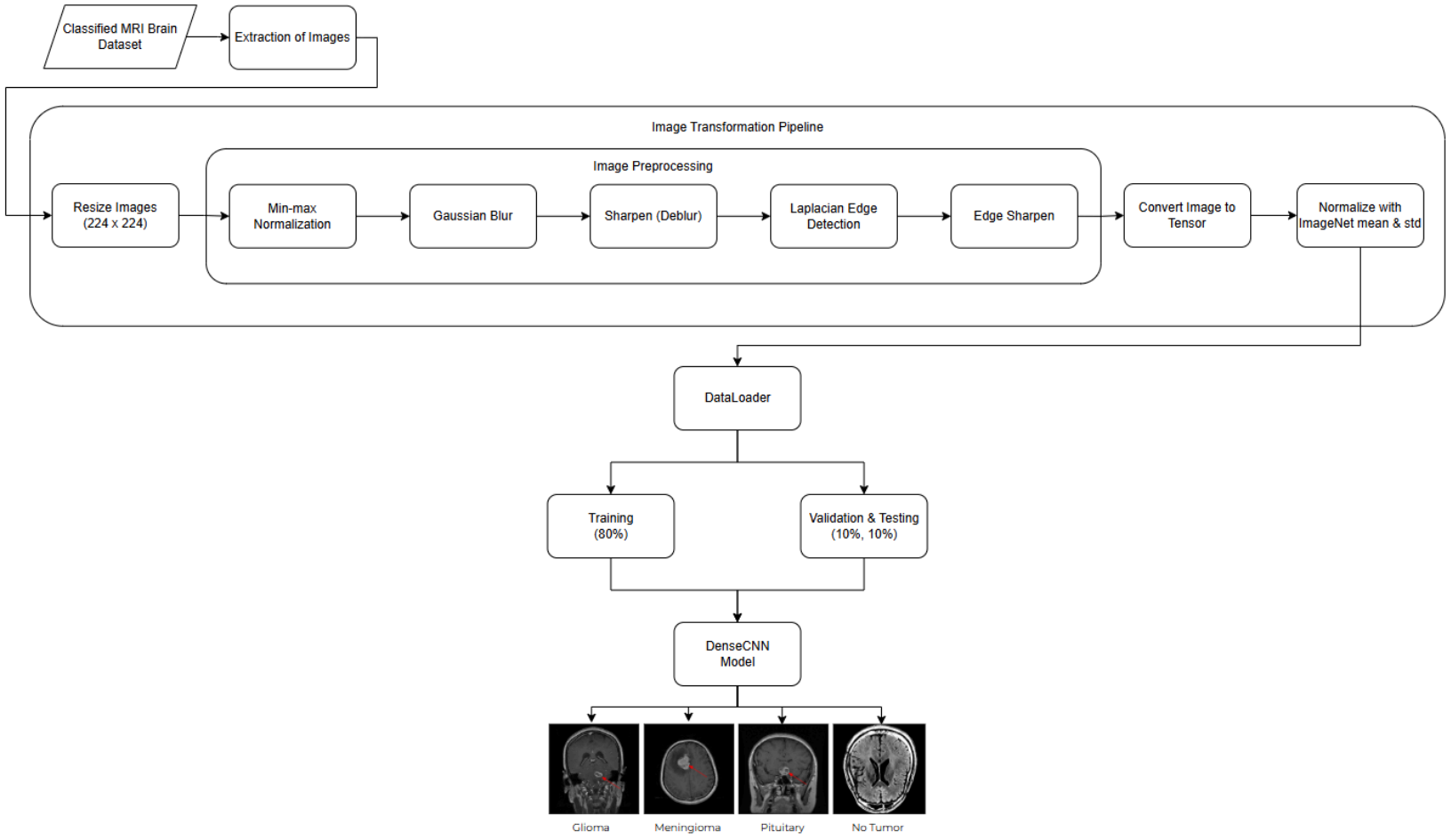

Figure 2: Overview of our methodology.

3.1. Dataset description

To begin, our chosen dataset is a publicly available collection of brain tumor MRIs from Kaggle.com. As previously mentioned, we classified the tumors into four categories, which the dataset already separates clearly: Glioma, Meningioma, Pituitary, and No Tumor (indicating the absence of any tumor). The initial dataset that we chose only had 3,264 images split in an 80-20 fashion of training and testing. The training set had 826 glioma tumors, 822 meningioma tumors, 827 pituitary tumors, and 395 healthy brain MRIs. The testing set had 115 glioma tumors, 115 meningioma tumors, 74 pituitary tumors, and 105 healthy brain MRIs. The initial dataset was noticeably imbalanced and relatively small in size, prompting us to switch to a larger (more discussed later in the paper), balanced, and more diverse collection of brain tumor MRI scans, also sourced from Kaggle.com. This new dataset contains 7,022 images and initially followed an 80-20 split for training and testing. To better align with standard deep learning practices, we created a custom script to repartition the dataset into an 80-10-10 split for training, validation, and testing. The resulting training set contains 1,321 glioma tumors, 1,339 meningioma tumors, 1,457 pituitary tumors, and 1,595 healthy brain MRIs. The validation set includes 150, 153, 150, and 202 images for each class, respectively, while the test set contains 150, 153, 150, and 203 images, respectively.

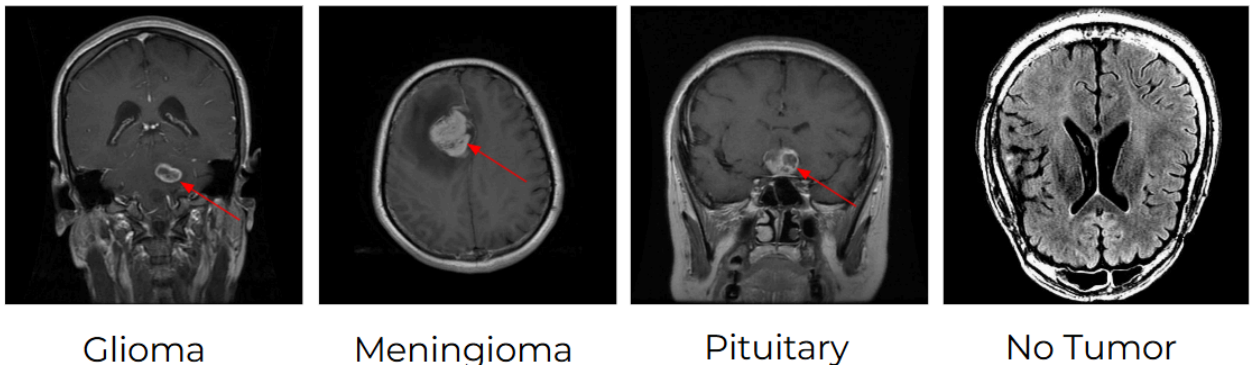

Figure 3: Dataset sample of the four classes.

3.2. Image Preprocessing, Transformation Pipeline, & Data Loading

A flaw with the MRIs is that they are of lower quality due to noise and low illumination. The patient's database from which the MRI came is unclear. These images also contain a certain amount of uncertainty. To mitigate uncertainty and improve classification, the MRIs require preprocessing before passing them into the model. The preprocessing phase consists of five steps, along with three additional critical steps in the transformation pipeline. For clarity, we group all eight steps under a single section. We perform the most crucial step—resizing images to 224×224 pixels—at the start of the transformation pipeline. The pre-trained EfficientNetB0 model expects a 224x224 resolution image. Next, we direct the resized images into our custom preprocessing pipeline, which performs the following steps in order: min-max normalization, Gaussian blur, sharpening (deblurring), Laplacian edge detection, and edge sharpening.

Min-max normalization is performed at the start to convert the image pixel values to a range of [0, 1]. The normalization function is defined as follows:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Where X' is the normalized image, X is the 2-dimensional matrix that represents the pixels as float32 type, X_{min} is the $min(image)$, and X_{max} is the $max(image)$. By performing this normalization, we can improve numerical stability in the following mathematical operations.

The next step is to apply a Gaussian filter with a radius of one blur via the “ImageFilter” function. The blur helps with the preceding step of sharpening.

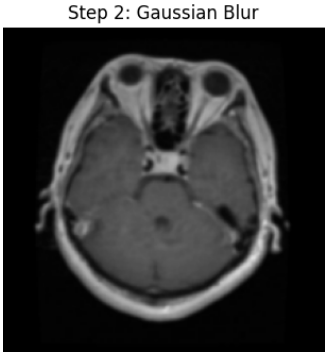

Figure 4: The applied Gaussian blur to an MRI.

The sharpening or deblurring step enhances the lines of the MRI and gives it a sharper appearance, helping to distinguish features better. The sharpening filter uses the following thresholding to clarify the finer details while still maintaining a natural look:

$$Sharpen\ Image = Normalized\ Image + threshold * (Normalized\ image - Blurred\ image)$$

The threshold is an adjustable parameter that controls the level of deblurring strength. In our case, 0.5 works best.

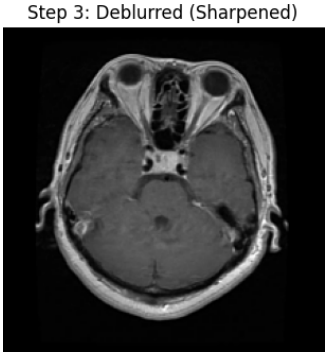

Figure 5: The applied sharpening effect to an MRI.

The preceding process is the Laplacian edge detector, which assists in identifying the edges, especially the finer ones. For simplicity, we used the "opencv" built-in Laplacian function on each R, G, and B channel with a kernel of size three. We followed it with a min-max normalization before reconstructing it as a Laplacian float threshold.

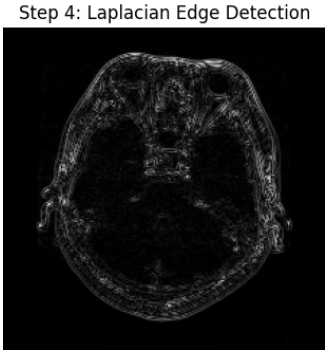

Figure 6: The applied Laplacian edge detector to an MRI.

The last preprocessing step is to apply the detected edge to the sharpened image. We will use a similar thresholding function as the sharpening step:

$$Processed\ Image = Sharpen\ Image + threshold * Laplacian$$

The threshold is an adjustable parameter that controls the level of deblurring strength. In our case, 0.5 works best. The overall objective of the Laplacian filter is to highlight the smaller tumors, especially with their finer outline.

After completing preprocessing, we convert each processed image into a tensor and normalize it using the ImageNet standard mean and standard deviation in the transformation pipeline. We then consolidate the dataset into a data loader and enable shuffling to ensure randomized batching during training.

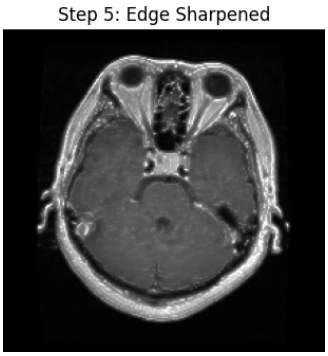

Figure 7: The final preprocessed MRI.

3.3. Model Architecture

Since we are classifying images, the neural network architecture required for the task is CNN; however, we have decided to use a pre-trained model for a more enhanced and accurate model. Our strategy involved adopting a transfer learning variant by leveraging pre-trained model weights from an extensive, general dataset and replacing the final layers with a set of custom dense final layers. We then fine-tuned these layers using brain tumor MRIs to enhance the model's ability to extract features and accurately classify brain tumor types. In a sense, this is a dense CNN architecture with a mixture of dense convolutional layers from EfficientNetB0 and the final dense classification layers.

The chosen pre-trained model, as mentioned prior, is the EfficientNetB0, a lightweight model with around 5.3 million parameters, designed for lower computational devices, such as mobile devices. Its model architecture can be summarized as follows in a sequential order: one 3x3 convolutional block with 32 channels taking in 224x224 images, one 3×3 MBConv1 block—a mobile inverted bottleneck convolution block that reverses the traditional bottleneck structure by first expanding the input channels and then compressing them—with 16 channels,

two layers of 3x3 MBConv6 blocks—difference between MBConv1 lies primarily in their expansion factor with an additional convolution block—with 24 channels, two layers of 5x5 MBConv6 blocks with 40 channels, three layers of 3x3 MBConv6 blocks with 80 channels, three 5x5 MBConv6 blocks with 112 channels, four layers of 5x5 MBConv6 blocks with 192 channels, and a 3x3 MBConv6 block with 320 channels.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	112×112	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	7×7	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 8: EfficientNetB0 model architecture table.

To elaborate further on the MBConv6 blocks, their structure is as follows: 1x1 convolution block (difference between MBConv6 and MBConv1 is that MBConv1 do not have this convolution block) with batch normalization and swish activation function—a non-linear variant of ReLU, depthwise convolutional block with batch normalization and swish activation function, squeeze optimization to convert each channel of the feature map into a single scalar value, excitation optimization—a fully connected network with sigmoid activation—that ensure the output weights are in the range of [0, 1], a 1x1 convolution block with batch normalization, and finally dropout before the output.

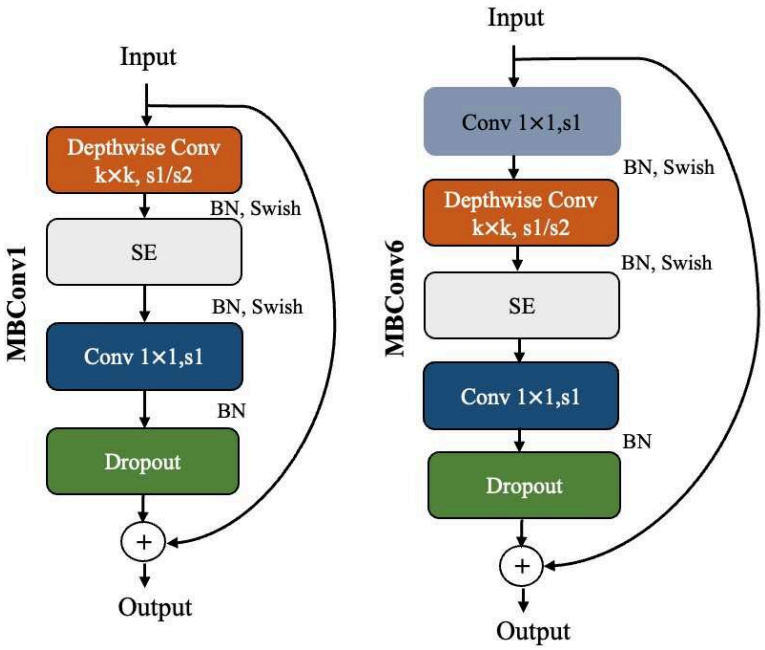

Figure 9: MBConv1 and MBConv6 architecture.

The custom fine-tuned classification layer of the dense CNN model first performs pooling, then flattens a contiguous range of dimensions into a tensor. Next, it goes through four dense layers and three dropout layers. The dropout layers help thin the network during training to help it learn new features and avoid overfitting. Each dense layer includes batch normalization to enhance convergence—an effect further discussed in the results—and a LeakyReLU activation function to mitigate the dying ReLU problem by allowing a small gradient for negative inputs, thereby preserving neuron activity and maintaining adequate gradient flow. This combination enhances the model's ability to learn complex features, which is critical for accurate tumor identification and classification. The number of neurons in the dense units and batch normalization is 720, 360, 360, and 180, respectively. The dropout values are 0.25, 0.25, and 0.5, respectively. All the LeakyReLU uses a negative slope of 0.01. Lastly, the output goes into a final dense layer that reduces the 180 features into four classes for the cases defined. The forwarding aspect follows the standard logits, where the features go through the defined dense CNN architecture.

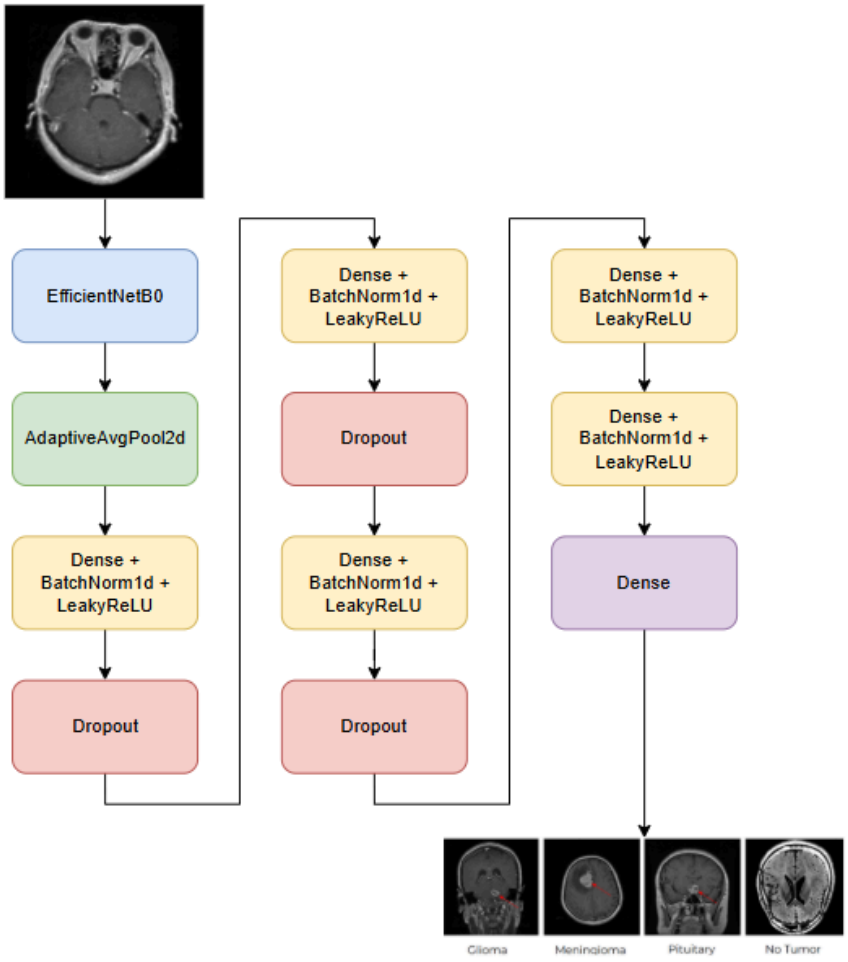

Figure 10: DenseCNN model with EfficientNetB0.

4. Experimental Setup

4.1. Hyperparameters & Training Configurations

The experiment configuration starts with the data. The MRI collection is stored locally on our machines, as it does not occupy too much space. As previously mentioned, the images are already neatly categorized in specific folders named training, validation, and test. From there, we also decided to store the model locally, as it was of reasonable size due to the lightweight nature of the EfficientNetB0 output. The history of the model is stored separately from the model to ensure that we get the complete picture of the validation loss, accuracy, and training loss. From there, we set up our hyperparameters for training, with things like our epochs set to 20, batch size

set to 64, number of workers set to 12, and initial learning rate at 0.0001. We initially set the epoch to 5 for testing purposes, then increased it for training the complete model. We configured these hyperparameters based on the specifications of the computer used for training. We ran the training locally on a machine equipped with 128GB of DDR5 RAM and an NVIDIA RTX 3060 GPU with 12GB of VRAM. An additional parameter we use is the CosineAnnealingWarmRestarts scheduler. We configured the scheduler to restart the learning rate every three epochs, with each new cycle lasting twice as long as the previous one. The scheduler helps avoid overfitting by allowing the training to explore new learning rates and diversify what it has learned. It also helps with generalization with the alternation between high and low learning rates, allowing for more complex feature learning, which is essential for distinguishing tumor patterns with MRI scans. We performed the training using the PyTorch library.

```
# Hyperparameters for training
training_root = "./DataSet/Training";
validation_root = "./DataSet/Validation";
model_path = "./model/DenseCNN_Brain_Tumor.pth";
history_path = "./model/DenseCNN_Brain_Tumor_history.pth";
batch_size = 64;
num_workers = 12;
scheduler_T_0 = 3;
scheduler_T_mult = 2;
learning_rate = 0.0001;
num_epochs = 20; # 20 for full training
```

Figure 11: Training hyperparameters.

```
# CosineAnnealingWarmRestarts for model to explore more
scheduler = torch.optim.lr_scheduler.CosineAnnealingWarmRestarts(
    optimizer,
    T_0=scheduler_T_0, # First restart happens after 3 epochs
    T_mult=scheduler_T_mult, # Restart occurs after each T_0 * T_mult epochs
    eta_min=1e-6 # Min learning rate
);
```

Figure 12: Scheduler.

4.2. Training Loop

With our training loop, we are using Cross-Entropy for our loss function and our optimizer AdamW with a weight decay of 0.01. Initially, we used a log loss function with a softmax output layer for the dense CNN model; however, we are not doing multi-label classification, so it yielded incorrect values. On the other hand, Cross-Entropy was effective for multi-class classification, as it penalizes incorrect predictions more heavily than other losses. AdamW optimizer helped with adaptive learning rates and improved weight decay handling. For each epoch, the training loop consists of a forward pass and backpropagation, which allows us to generate predictions and compute the loss, along with gradient computation to minimize the loss and update the weights as we train the model. We are also freeing memory per batch as we are dealing with thousands of images, and we wanted to ensure that we do not run out of memory. From there, we compute the validation loss and accuracy, store them in the history, and compare them with the training metrics to check for overfitting and assess overall accuracy. For every epoch, we are validating the model's performance with the prior epoch or checkpoint to ensure that we acquire the best model at the end and avoid the standard deviations that occur frequently during training. Each epoch took around three minutes to train on our configurations.

```
for epoch in range(epochs):
    model.train();
    total_loss, correct, total = 0, 0, 0;

    for images, labels in data_loader:
        images, labels = images.to(device), labels.to(device);

        # Forward pass
        outputs = model(images);
        loss = criterion(outputs, labels);

        # Backpropagation
        optimizer.zero_grad();
        loss.backward();
        optimizer.step();

        # Collect statistics
        total_loss += loss.item();
        _, predicted = torch.max(outputs, 1);
        correct += (predicted == labels).sum().item();
        total += labels.size(0);

        # Free memory per batch
        del images, labels, outputs, loss, predicted;
        torch.cuda.empty_cache();

    # Validation
    model.eval();
```

Figure 13: Training Loop.

5. Results and Discussion

5.1. Initial results

In our initial model, we saw that it performed well but still had room for improvement, specifically with the glioma and meningioma tumors. We anticipated a slightly higher F1-score, given that the base EfficientNetB0 model alone achieves a top-1 accuracy of 77.1% and a top-5 accuracy of 93.3%, as reported by Mingxing Tan and Quoc V. Le in their paper *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. Some factors explaining why it was underperforming were due to overfitting, in which the model was memorizing the training data rather than learning. This memorization causes the model to struggle with unseen images, especially with glioma and meningioma tumors. Another factor was that our initial implementation with early stopping always triggered early and incorrectly. The early stopping assumes that the model was not improving, and with the patience set at three epochs, it rarely saw improvement within three epochs of the best model. Another, less obvious issue stemmed from the size of the training dataset. As previously discussed, we initially used a smaller set in our first approach but replaced it with a larger one in the next phase. We will discuss the upgrades and patches further in the next approach.

The classification report—using the built-in sklearn metrics library, which still uses the standard model evaluation metric equations—in Figure 15 gives us an in-depth look at how our model performed. We are primarily focusing on the F1-score, which combines both precision and recall metrics. It is essential to know both of these since precision measures how many of the predicted positive cases are correct. In contrast, recall measures how many of the actual positive cases the model correctly identifies. As we see, the glioma and meningioma are the lowest of the 4, which drops our average to 96.8%. These are still good results and allow us to enhance our model.

Metric	Formula
True positive rate, recall	$\frac{TP}{TP+FN}$
False positive rate	$\frac{FP}{FP+TN}$
Precision	$\frac{TP}{TP+FP}$
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
F-measure	$\frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$

Figure 14: Standard model evaluation metric equations.

```
Using device: cuda
Evaluation results on test set:
Loss: 0.0881
Accuracy: 0.9680

Classification Report:
              precision    recall  f1-score   support

      glioma     0.9074    0.9800    0.9423       150
  meningioma     0.9718    0.9020    0.9356       153
     notumor     0.9951    1.0000    0.9975       203
   pituitary     0.9932    0.9800    0.9866       150

    accuracy                         0.9680       656
   macro avg     0.9669    0.9655    0.9655       656
weighted avg     0.9692    0.9680    0.9680       656
```

Figure 15: Classification report on initial results.

As shown in the confusion matrix in Figure 16, the model most frequently misclassifies images from the meningioma and glioma classes. In contrast, it performs nearly perfectly when identifying pituitary tumors and cases without tumors.

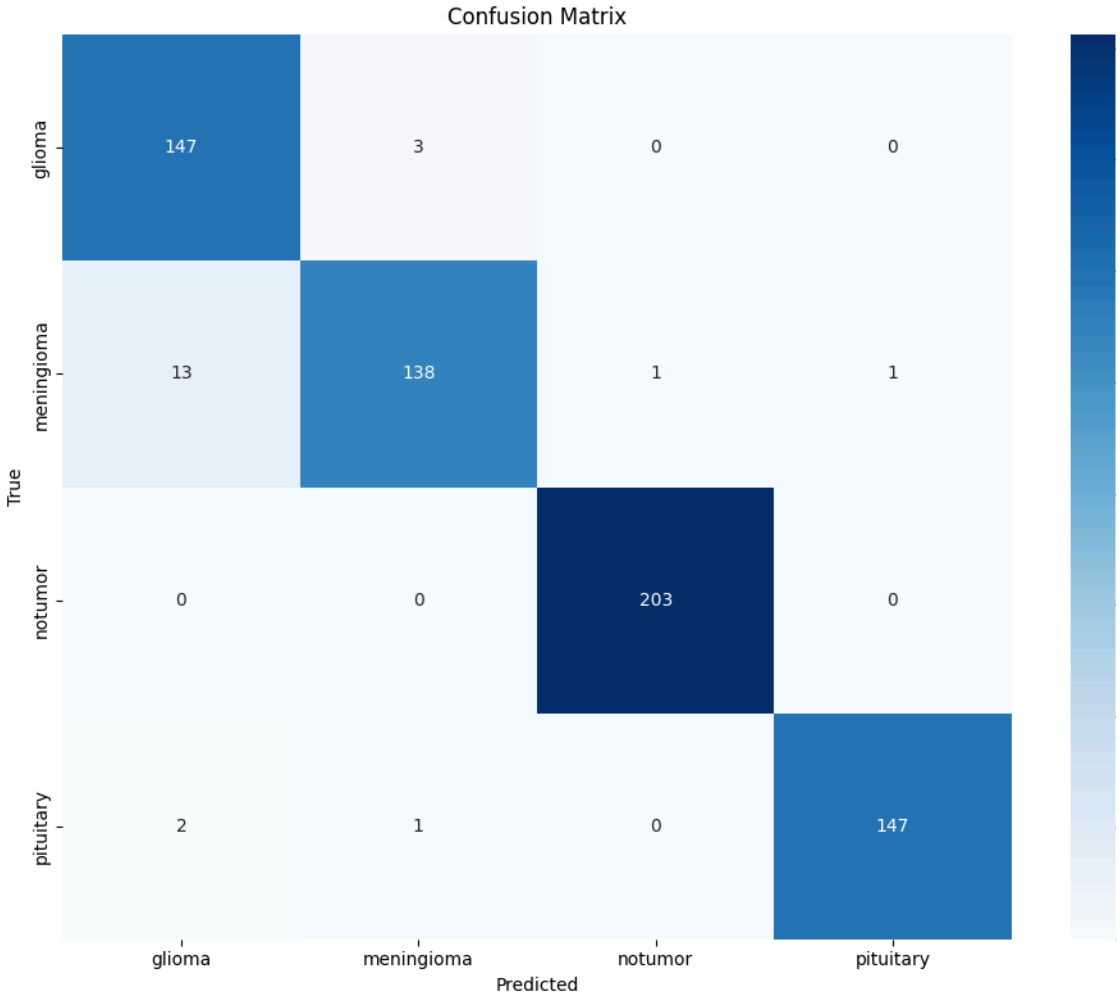

Figure 16: Confusion matrix on initial results.

As shown in the training loss curve in Figure 17, both the training and validation losses decrease rapidly before leveling off, indicating that the model is effectively learning and stabilizing. Since we use cross-entropy loss to quantify prediction error, a lower loss value suggests that the model's output probabilities are becoming more aligned with the actual class labels. This reduction in error contributes directly to improved classification performance, shown in the higher evaluation metrics such as the F1 score. However, we still need to recognize the flaws pointed out in the initial training.

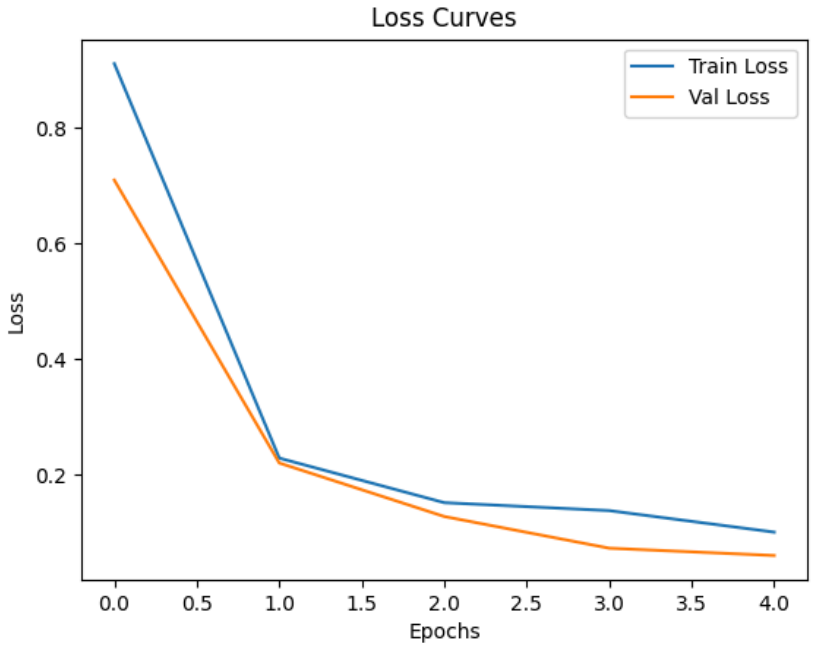

Figure 17: Loss curve graph on initial results.

5.2. Improved results

In our improved model, we incorporated batch normalization and replaced the ReLU activation function with Leaky ReLU, as detailed in the model architecture. Batch normalization contributes to more stable and efficient convergence of both training and validation losses, as observed around epoch 8 in Figure 20. The Leaky ReLU, as already discussed, helps the model learn complex features better. As mentioned countless times, switching from the 3,264-image dataset to 7,022 images was the most straightforward and effective way of reducing overfitting. With these improvements, our classification reports in Figure 18 revealed an astounding F1-score of 98.63% accuracy, a 1.83% increase from the initial model. It may not seem like much, but improving it to get closer to 100% is challenging, and any 0.01% increase matters. Where we saw our most significant improvements were in the glioma and meningioma tumors, where the rates are 97.99% for glioma with a 3.76% increase, and meningioma is 97.09% with a 3.53% increase.

```
Loss: 0.0592
Accuracy: 0.9863

Classification Report:
              precision    recall  f1-score   support

      glioma     0.9865    0.9733    0.9799       150
  meningioma     0.9615    0.9804    0.9709       153
     notumor     1.0000    0.9951    0.9975       203
   pituitary     0.9933    0.9933    0.9933       150

    accuracy                         0.9863       656
   macro avg     0.9853    0.9855    0.9854       656
weighted avg     0.9864    0.9863    0.9863       656
```

Figure 18: Classification report on improved model.

Our confusion matrix in Figure 19 shows improvements in classifying glioma and meningioma tumors, and remains roughly the same. The model also did slightly better in classifying the pituitary and healthy brain cases. The changes did make a slight impact on improving the accuracy.

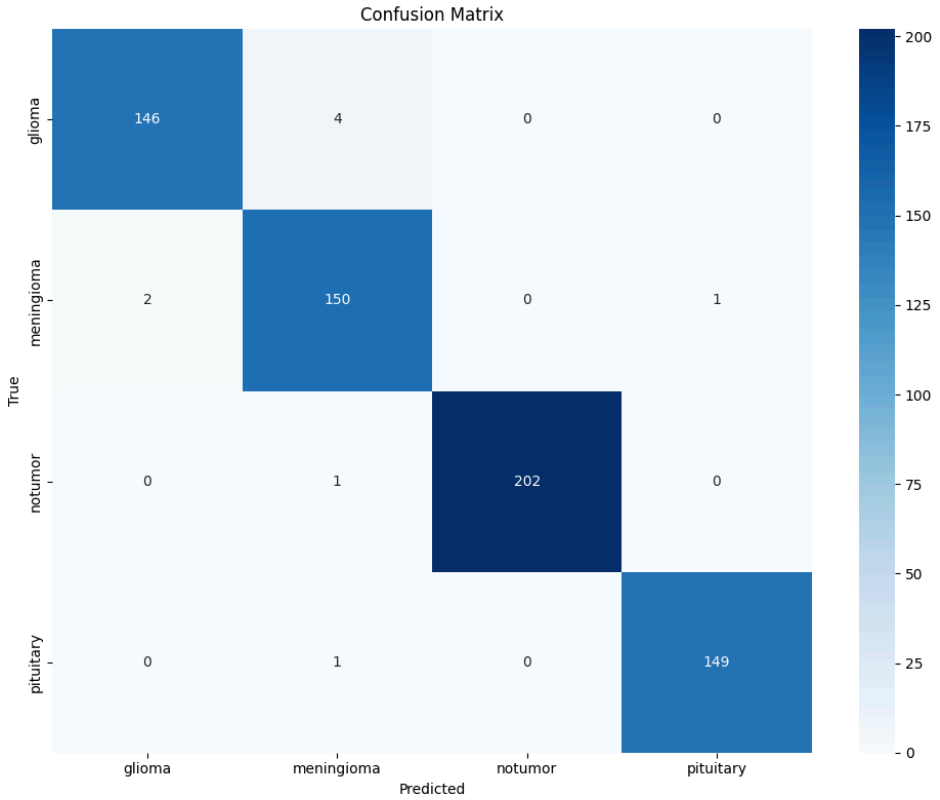

Figure 19: Confusion matrix on improved model.

Our final graph in Figure 20 illustrates the training and validation loss, accuracy, and learning rate adjustments over 20 epochs as managed by the scheduler. Throughout this period, the model showed steady improvement, aligning with our expectations. Both training and validation loss decreased before plateauing, revealing a relatively smooth convergence. Notably, the model achieved 100% training accuracy by the eleventh epoch. The downside of this is that it indicates that the model is overfitting on the training images.

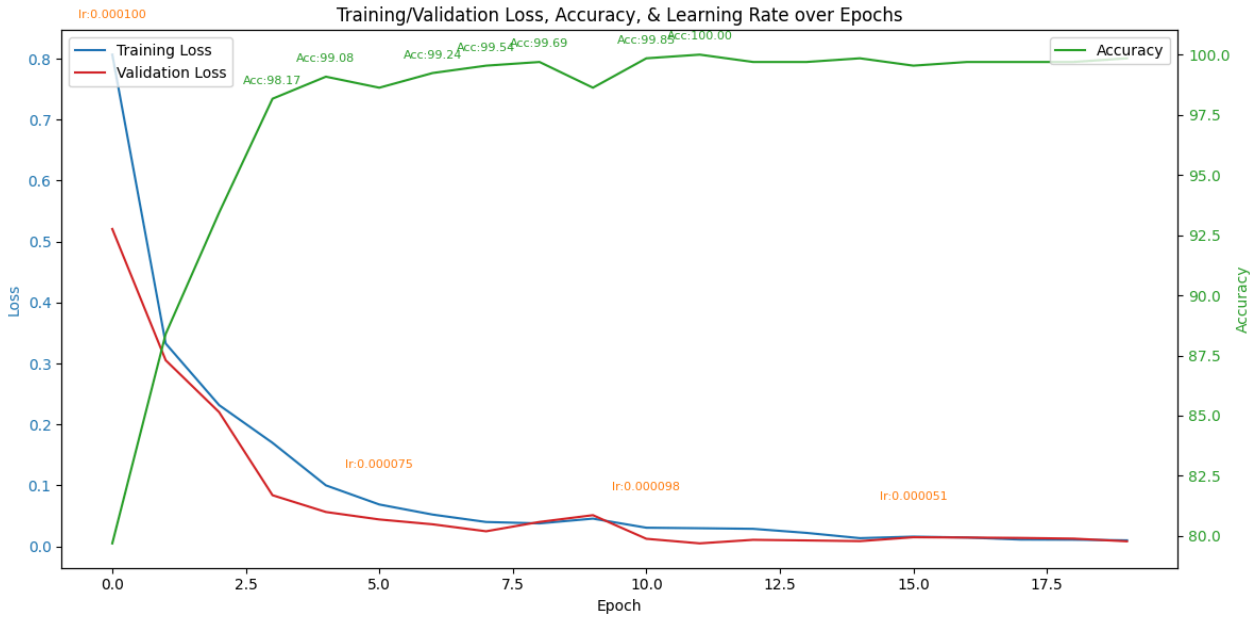

Figure 20: Training/Validation Loss Graph

5.3. Discussion

In the final round of training, we observed signs of overfitting, indicating that the model had likely memorized the training data. Further analysis revealed that when tested against the original testing set (old dataset) from our initial results, the model achieved only a modest F1-score of 74%, as shown in Figure 21. To address this overfitting, a key recommendation for future work is to introduce more training data to the DenseCNN model. Given the strong performance of EfficientNetB0, architectural modifications may risk degrading accuracy, unless implemented as part of a sequential multi-model framework, such as combining a U-Net for image segmentation with a DenseCNN for classification. Additionally, we suggest merging the two datasets and carefully removing duplicate MRIs to construct a larger and more diverse dataset, which would enable the model to learn richer feature representations. As a final note,

future work could explore using a higher-capacity model such as EfficientNetB7, which contains 66 million parameters compared to the relatively lightweight 5.3 million in EfficientNetB0. According to benchmarks reported by Tan and Le, EfficientNetB7 achieves a significantly higher top-1 accuracy of 84.3% and a top-5 accuracy of 97%, offering the potential for improved classification performance.

```
Using device: cuda
Evaluation results on test set:
Loss: 1.2284
Accuracy: 0.7411

Classification Report:
                  precision    recall  f1-score   support

    glioma_tumor     1.0000    0.2000    0.3333       100
meningioma_tumor     0.6354    1.0000    0.7770       115
        no_tumor     0.7447    1.0000    0.8537       105
 pituitary_tumor     1.0000    0.7027    0.8254        74

        accuracy                         0.7411       394
       macro avg     0.8450    0.7257    0.6974       394
    weighted avg     0.8255    0.7411    0.6939       394
```

Figure 21: Classification report on further overfitting investigation.

6. Conclusion

Throughout this project, we explored various approaches. One of our initial considerations was selecting the appropriate model architecture. U-Net emerged as a strong candidate due to its widespread application in the biomedical field. However, we ultimately ruled it out because of the absence of publicly available datasets containing labeled ground truth segmentation masks, rather than full MRI scans, which made its implementation unfeasible. Another thing we experimented with was data augmentation. The main reason for us using data augmentation was to give us more data with a smaller dataset, but along with that, the results we had with it were underwhelming. In the context of medical imaging, overly aggressive augmentation can distort critical anatomical features, leading the model to learn unrealistic or incorrect patterns, which ultimately hinders performance. That is why we opt to use a large dataset. While our implementation of DenseCNN focused on brain tumor classification, its lightweight and versatile architecture makes it well-suited for a wide range of image classification tasks beyond the medical domain.

References

Herath, S. (2024, January 15). Efficientnetb0 architecture-block 2. Medium. https://medium.com/image-processing-with-python/efficientnetb0-architecture-block-2-b00cc690e891.

Laplacian filter. Laplacian Filter - an overview | ScienceDirect Topics. (n.d.). https://www.sciencedirect.com/topics/engineering/laplacian-filter.

Nayak, D. R., Padhy, N., Mallick, P. K., Zymbler, M., & Kumar, S. (2022). Brain Tumor Classification Using Dense Efficient-Net. Axioms, 11(1), 34. https://doi.org/10.3390/axioms11010034.

Nayak, D.R.; Padhy, N.; Swain, B.K. Brain Tumor Detection and Extraction using Type-2 Fuzzy with Morphology. Int. J. Emerg. Technol. 2020, 11, 840–844. https://www.researchtrend.net/ijet/pdf/Brain%20Tumor%20Detection%20and%20Extraction%20using%20Type%20%20Fuzzy%20with%20Morphology%20Dillip%20Ranjan%20Nayakj1.pdf.

Ngoc, Hoang & Nguyen Vinh, Nghi & Lê, Nhi & Nguyen, Nam & Le, Thu & Dinh Nguyen, Vinh. (2024). Enhancing Semantic Scene Segmentation for Indoor Autonomous Systems Using Advanced Attention-Supported Improved UNet. 10.21203/rs.3.rs-4587262/v1. https://www.researchgate.net/publication/381931873_Enhancing_Semantic_Scene_Segmentation_for_Indoor_Autonomous_Systems_Using_Advanced_Attention-Supported_Improved_UNet. (Image citation)

Riza, Jaohar & Barman, Shohag & Ridita, Sadia & Mahmud, Zaeed & Bhattacharya, Aditi. (2024). PhytoCare: A hybrid approach for identifying Rice, Potato, and Corn diseases. https://www.researchgate.net/publication/378395574_PhytoCare_A_hybrid_approach_for_identifying_Rice_Potato_and_Corn_diseases. (Image citation)

Tan, M., & Le, Q.V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. ArXiv, abs/1905.11946. https://arxiv.org/pdf/1905.11946.