

Contents

[1. Problem Statement 2](#_Toc200801823)

[2. Research on Neural Networks and Architectures 3](#_Toc200801824)

[2.1 Neural Networks for Brain Tumor Classification 3](#_Toc200801825)

[2.2 Modern CNN Architectures 4](#_Toc200801826)

[2.3 Architecture Comparison 5](#_Toc200801827)

[3. Models' Development and Training 6](#_Toc200801828)

[3.1 Dataset Preparation 6](#_Toc200801829)

[3.2 Training and Validation 7](#_Toc200801830)

[4. Models' Testing and Evaluation 9](#_Toc200801831)

[4.1 Testing Methodology 9](#_Toc200801832)

[4.2 Overfitting/Underfitting Assessment 10](#_Toc200801833)

[4.3 Results Analysis 10](#_Toc200801834)

[4.4 Effectiveness Assessment 11](#_Toc200801835)

[4.5 Interface Development 11](#_Toc200801836)

[4.6 Critical Evaluation and Future Improvements 11](#_Toc200801837)

# 1. Problem Statement

My project is solving a very very important and critical medical challenge that is automated brain tumor detection and classification using deep learning. My primary objective is to develop a robust deep learning system capable of accurately classifying brain tumors from MRI scans into four distinct categories.

**Problem Definition**

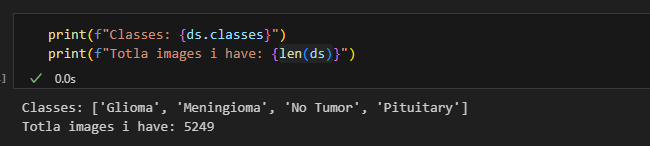
The specific problem involves multi-class image classification of brain MRI scans with the following characteristics:

* **Task Type**: Medical image classification

This is a multi-class problem because the system needs to distinguish between four distinct and mutually exclusive outcomes different tumor types and healthy brain, rather than just a simple binary tumor/no tumor classification.

* **Input**: High-resolution MRI brain scans
* **Output**: Classification into one of four categories
* **Classes**:
  + Class 0: Glioma ‘one of the tumor types’
  + Class 1: Meningioma ‘one of the tumor types’
  + Class 2: No Tumor ‘healthy brain with no tumor’
  + Class 3: Pituitary ‘one of the tumor types’

**Dataset Composition:**



* Total images: 5,249 MRI scans
* Views: Sagittal and axial and coronal MRI perspectives

**Data Distribution:**

A graph of a number of bars

AI-generated content may be incorrect.

Training Set (70% from the images), Validation Set (15% from the images), Testing set (15% from the images)

The dataset presents a slightly imbalanced classification problem, with "No Tumor" being underrepresented compared to the tumor classes. While my models achieved high overall accuracy, it was crucial to monitor precision, recall, and F1-score for individual classes, especially the minority class actually high overall accuracy could potentially mask poor performance on underrepresented classes. The use of weighted average metrics in the classification report helped to provide a more representative performance score across all classes, accounting for their differing sample sizes and providing a balanced view of the model's true effectiveness across all categories.

# 2. Research on Neural Networks and Architectures

## Neural Networks for Brain Tumor Classification

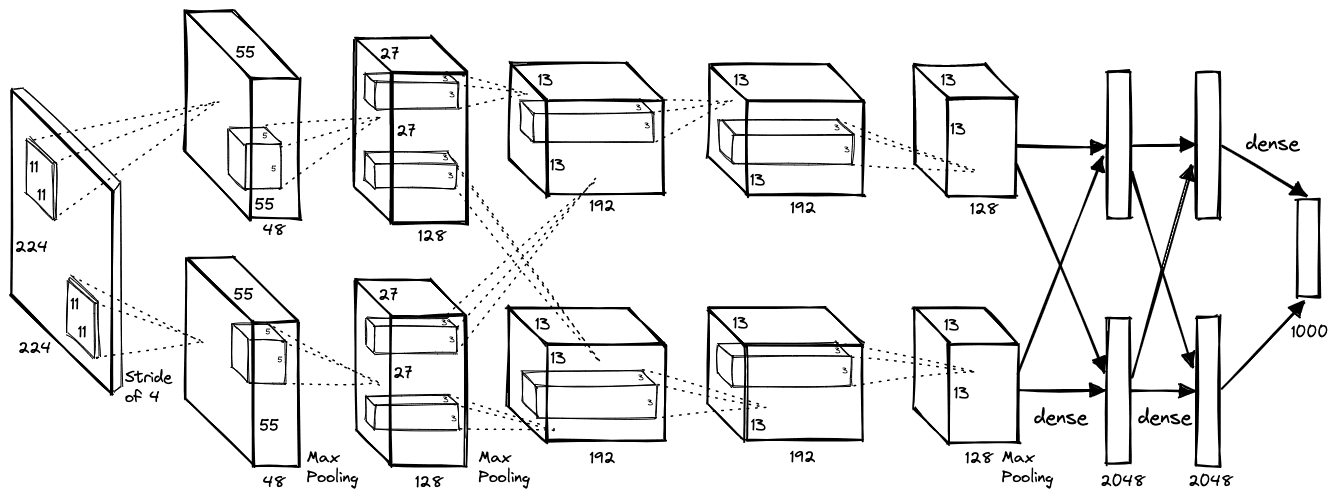


**Convolutional Neural Networks (CNNs)** it is a better and enhanced version from ANN is made to extract features from data its used more the time with images and videos it does have a key componantes, the convolutional layers and the ReLU and the pooling layes and at last is the fully connected layer and for my problem we can say that CNNs are the predominant architecture for medical image classification tasks. CNNs are particularly suitable for this problem due to their ability to extract hierarchical features from medical images as I said and they maintain spatial relationships in MRI data and as well as handle varying image sizes and orientations and they also do provide translation invariance for tumor detection. As we know the convolutional layers detect low-level features (edges, textures) in early layers and complex patterns (tumor shapes, tissue boundaries) in deeper layers, making them ideal for distinguishing between different tumor types and healthy tissue.

Unlike traditional Artificial Neural Networks (ANNs) which struggle with the high dimensionality and spatial relationships inherent in image data, CNNs are specifically designed with specialized layers like convolutional and pooling layers that excel at automatically learning hierarchical features directly from raw pixels, making them a more robust and efficient solution for visual tasks.

## 2.2 Modern CNN Architectures

**1. AlexNet (2012)**



AlexNet was one of the first successful deep CNNs, winning ImageNet 2012 competition it does recognize the content of photographic images and was built by university of Toronto graduate students Alex and Ilya and actually in 2020 they abled the code to be access by any one here is the [code source](https://github.com/computerhistory/AlexNet-Source-Code) for AlexNet. For the structure of the model we have 8 layers (5 convolutional + 3 fully connected) with ReLU activation and its key features that it was the first to use ReLU, dropout, and GPU acceleration effectively.

Some of its architecture details having 8 layers total, 5 convolutional layers followed by 3 fully connected layers its input size 227×227×3 RGB images and its novel activation that they are the first major network to use ReLU instead of traditional sigmoid or tanh functions and introduce dropout with 0.5 probability in the first two fully connected layers and for the hardware where pioneered the use of GPU acceleration, training on two NVIDIA GTX 580 GPUs and for its **Impact and Legacy,** AlexNet's success marked the beginning of the deep learning era in computer vision. The code was made publicly available in 2020 allowing researchers to study and build upon this foundational work and It established many practices that became standard in CNN design and demonstrated the importance of large datasets and computational power.

**2. VGG-16/VGG-19 (2014)**

The Visual Geometry Group (VGG) at the University of Oxford, led by Karen Simonyan and Andrew Zisserman, developed VGG networks to investigate the effect of convolutional network depth on accuracy. Their research demonstrated that depth is crucial for good performance, establishing a new paradigm in CNN architecture design its structure is as this starting with the uniform desigh where is have use only 3×3 convolutional filters throughout the entire network and VGG-16 contains 16 weight layers (13 conv + 3 FC), while VGG-19 has 19 layers (16 conv + 3 FC) and sequential arrangement of convolutional blocks followed by max pooling having 2×2 max pooling with stride 2 there innovation was that they demonstrated that stacking multiple 3×3 convolutions is more effective than using larger filters and showed that increasing depth (rather than width) significantly improves performance and Two 3×3 convolutions have the same receptive field as one 5×5 but with fewer parameter and for there **performance and Applications,** VGG networks achieved excellent performance on ImageNet, with VGG-16 reaching 7.3% top-5 error rate and the architecture became widely adopted for transfer learning due to its simplicity and strong feature extraction capabilities.

**3. GoogleNet (Inception-V1) (2014)**

developed by Google's research team led by Christian Szegedy, introduced the revolutionary Inception module concept and This architecture won ILSVRC 2014 and demonstrated that computational efficiency could be achieved without sacrificing accuracy through innovative architectural design it does have multi-scale process as a parallelel convolution operations with different kernel sizes (1×1, 3×3, 5×5) and Strategic use of 1×1 convolutions to reduce computational cost and also Four parallel branches including max pooling processed simultaneously **and** Feature maps from all branches are concatenated along the channel dimension its structure was as 22 layer and 9 inception a pooling the wDespite being deeper, GoogleNet has fewer parameters than AlexNet (6.8M vs 60M) and inception modules provide multi-scale feature extraction with controlled computational cost and as well the **gradient flow where** Auxiliary classifiers help gradients reach earlier layers during training

**Impact on Field:** GoogleNet established the principle that architectural innovation could achieve better performance with fewer resources, influencing subsequent designs like ResNet and EfficientNet.

**4. ResNet-50 (2015)**

Residual Networks (ResNet), developed by Kaiming He and colleagues at Microsoft Research, solved the fundamental problem of vanishing gradients in very deep networks through the introduction of skip connections. This breakthrough enabled training of networks with hundreds of layers lets talk about its residual learning framework starting with Skip connections: Direct connections that bypass one or more layers and as well identity mapping as it allows gradients to flow directly to earlier layers and it as well networks learn residual mappings F(x) = H(x) - x instead of H(x) directly and it uses 1×1 convolutions to reduce and restore dimensions efficiently the architecture having 50 layer composed of residual blocks with bottleneck architecture and each stage doubles the number of channels while halving spatial dimensions 1×1→3×3→1×1 convolution sequence reduces computational complexity applied after each convolution and before activation and skip connections provide alternative gradient paths and it enable successful training of 152-layer networks and beyond deeper networks consistently outperform shallower ones and parameter-free connections that don't add computational complexity

**5. DenseNet-121 (2017)**

Dense Convolutional Networks (DenseNet), developed by Gao Huang, Zhuang Liu, and others, took the concept of skip connections to the extreme by connecting each layer to every subsequent layer in a feed-forward fashion. This creates maximum information flow between layers and its connectivity pattern is really amazing that’s because it have **Dense blocks,** Within each block, every layer receives feature maps from all preceding layers and **Feature concatenation,** Instead of summation (as in ResNet), DenseNet concatenates feature maps and **Transition layers,** Between dense blocks, 1×1 convolution and pooling reduce feature map dimensions and **Growth rate that** controls how much new information each layer contributes what make it special is that it have 121 layer with four dense blocks with varying numbers of layers (6, 12, 24, 16) and Each layer adds 32 feature maps and 1×1 convolutions reduce input feature maps before expensive 3×3 convolutions and as well the transition layers reduce feature maps by factor of 0.5, its advantages that it maximum utilization of feature information throughout the network and have fewer parameters than ResNet despite comparable performance and dense connections facilitate gradient propagation and implicit deep supervision acts as regularization. DenseNet-121 achieves excellent accuracy with fewer parameters, making it suitable for applications where memory is constrained while maintaining high performance.

**6. EfficientNet-B0 (2019)**

EfficientNet, developed by Mingxing Tan and Quoc V. Le at Google Brain, revolutionized CNN scaling through the compound scaling method. This approach systematically scales network dimensions to achieve optimal accuracy-efficiency trade-offs and lets talk about its compound scaling innovation starting with three dimensions that simultaneously scales depth (layers), width (channels), and resolution (input size) and scaling coefficient that uses a single compound coefficient to control all thrie dimensions and it as well maintains balance between different scaling dimensions and base architecture (B0) designed using NAS optimization the architecture having mobile inverted bottleneck (MBConv) as core building block with depthwise separable convolutions and squeeze-and-excitation attention mechanism that recalibrates channel-wise feature responses and uses Swish activation function (x × sigmoid(x)) instead of ReLU and B0 through B7 variants scale systematically using compound method and its technical features that depthwise separable convolutions reduce computational cost while maintaining representational power and squeeze-and-excitation blocks improve representational power with minimal computational overhead and training starts with smaller images and gradually increases resolution what make it special is that EfficientNet-B0 achieves 77.3% top-1 accuracy on ImageNet with only 5.3M parameters, demonstrating superior efficiency compared to previous architectures.

**7. MobileNet-V2 (2018)**

MobileNet-V2, developed by Google's mobile vision team, specifically targets mobile and embedded applications where computational resources are severely limited. It builds upon the original MobileNet with improved inverted residual blocks and its inverted residual design is really amazing that's because it have **expansion phase** that 1×1 convolution expands the number of channels and **depthwise phase** that 3×3 depthwise convolution processes spatial information and **projection phase** that 1×1 convolution projects back to lower-dimensional space and **linear bottleneck** that final layer uses linear activation instead of ReLU what make it special is that it have 53 layers lightweight architecture with inverted residual blocks and expansion factor typically 6, expanding bottleneck layers by this factor and maintains low memory footprint during inference and designed to work well with 8-bit quantization and its mobile-optimized features that depthwise separable convolutions factorize standard convolutions into depthwise and pointwise operations and α parameter controls network width and computational cost and ρ parameter controls input image resolution and skip connections between bottleneck layers, its advantages that it achieves 72% ImageNet top-1 accuracy with only 3.4M parameters, making it ideal for mobile applications, IoT devices, and edge computing scenarios.

**8. Inception-V3 (2015)**

Inception-V3, developed by Google's research team, represents a significant evolution of the original Inception architecture with improved computational efficiency and performance through factorized convolutions and architectural refinements and lets talk about its factorized convolution innovation starting with asymmetric convolutions that replaces 5×5 convolutions with two 3×3 convolutions and spatial factorization that further factorizes 3×3 into 1×3 followed by 3×1 convolutions and it as well reduces computational cost while maintaining representational capacity and multiple types optimized for different stages of the network the architecture having 42 layers deep architecture with multiple Inception module variants and combines different factorized convolution patterns and maintains computational efficiency during downsampling and multiple techniques including dropout, batch normalization, and label smoothing and its architectural improvements that refined auxiliary classifiers with batch normalization and efficient techniques for reducing grid size while expanding filter banks and integrated throughout the network for improved training stability and regularization technique that prevents overconfident predictions what make it special is that InceptionV3 achieved state-of-the-art performance on ImageNet with improved computational efficiency influencing subsequent work on efficient neural architecture design and establishing principles for factorized convolutions that are still used today.

These modern CNN architectures, originally developed for broad computer vision tasks, have proven highly adaptable and effective for specialized applications like medical image analysis due to their ability to extract intricate, hierarchical features crucial for diagnostic precision.

## 2.3 Architecture Comparison

|  |  |  |  |
| --- | --- | --- | --- |
| Architecture | Description and Layers | Advantages | Disadvantages |
| AlexNet | 8 layers (5 conv + 3 FC), ReLU activation, dropout regularization | First successful deep CNN, pioneered GPU training, good baseline model | Large parameter count, prone to overfitting, outdated architecture |
| VGG-16 | 16 layers with 3×3 conv filters, simple sequential design | Simple architecture, good baseline performance, pre-trained weights available | High memory usage, computational expensive, prone to overfitting |
| GoogleNet | 22 layers with Inception modules, 1×1 convolutions, auxiliary classifiers | Efficient computation, multi-scale features, reduced parameters | Complex architecture, auxiliary losses complicate training |
| ResNet-50 | 50 layers with residual connections, bottleneck blocks | Solves vanishing gradient, enables very deep networks, excellent performance | Moderate complexity, requires more training time |
| DenseNet-121 | 121 layers with dense connections, efficient parameter usage | Parameter efficient, strong gradient flow, good generalization | High memory usage during training, complex architecture |
| EfficientNet-B0 | Compound scaling method, mobile inverted bottlenecks | Best accuracy-efficiency trade-off, scalable design, modern architecture | Relatively new, requires careful hyperparameter tuning |
| MobileNet-V2 | Inverted residual blocks, depthwise separable convolutions | Very lightweight, fast inference, mobile-friendly | Lower accuracy compared to larger models, limited capacity |
| Inception-V3 | Factorized convolutions, inception modules, auxiliary classifiers | Good accuracy-efficiency balance, multi-scale processing | Complex architecture, sensitive to hyperparameters |

**Architecture Selection Justification:** Based on the comparison, I will implement:

1. **AlexNet**: As a historical baseline to compare against modern architectures
2. **MobileNet-V2**: For its very lightweight, fast inference, mobile-friendly
3. **ResNet-50**: For its ability to train deep networks effectively
4. **EfficientNet-B0**: For optimal accuracy-efficiency balance

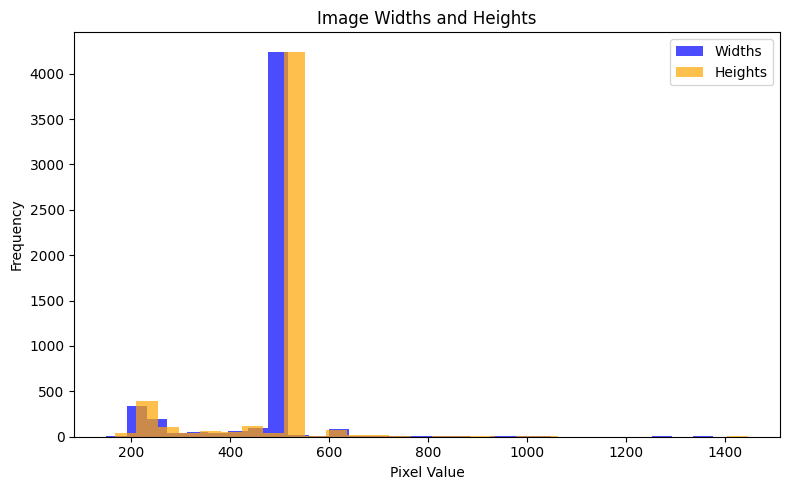
My selection of these four architectures was strategic, aiming to cover a spectrum of deep learning advancements: AlexNet serves as a historical baseline to benchmark against, MobileNet-V2 demonstrates efficient deployment for resource-constrained environments, ResNet-50 showcases the benefits of very deep networks in handling complex medical features, and EfficientNet-B0 represents a modern approach to achieving optimal accuracy-efficiency trade-offs. This diverse selection allows for a comprehensive understanding of different architectural paradigms in addressing my problem.

# 3. Models' Development and Training

## 3.1 Dataset Preparation

**Dataset Characteristics:**

Image resolution: Variable sizes (will be standardized to 224×224 pixels)



File format: JPEG images

**Data Preprocessing Steps:**

1. **Image Resizing**: Standardize all images to 224×224 pixels according to the plot that I got in my code to see the sizes of the images
2. **Normalization**: Apply ImageNet normalization (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
3. **Data Augmentation,** While initial results indicated satisfactory performance without extensive data augmentation, a scientific understanding suggests that such techniques Random horizontal flipping, random rotation, random brightness adjustment, random contrast adjustment are generally highly beneficial in deep learning. Data augmentation helps to artificially expand the dataset's diversity, making models more robust and less prone to overfitting by exposing them to a wider range of variations that might be encountered in real-world MRI scans For future iterations or if signs of overfitting become more pronounced, incorporating these augmentation strategies would be a primary step to improve generalization and model resilience.
4. **Duplicate Removal**: Check for and remove any duplicate images and there was no duplicates
5. **Class Balance Analysis**: Monitor class distribution for potential weighting strategies

## 3.2 Training and Validation

**Data Split Strategy:**

* Training Set: 70% (3,674 images)
* Validation Set: 15% (787 images)
* Test Set: 15% (788 images)



To ensure a more robust evaluation of my models' generalization capabilities and to minimize bias from a single train/validation split, I implemented 2-fold cross-validation during the hyperparameter tuning and model selection phase, especially for the AlexNet model. This involved 🡪

1.Splitting the Training Data My original `train\_ds` (70% of the full dataset) was further divided into two distinct folds.

2.Iterative Training For each fold One portion of the `train\_ds` served as the training set, and the remaining portion served as the validation set. fresh instance of the model for example for AlexNet was initialized for each fold to ensure independent training. The model was trained on the training subset and evaluated on the validation subset for the specified number of epochs. Performance metrics (loss and accuracy) were recorded for each epoch within the fold.

3. Aggregated Performance After training across both folds, the average performance metrics average validation accuracy were calculated across all folds to provide a more reliable estimate of the model's performance. The model instance achieving the best overall average validation accuracy across all folds was then selected as the optimal model for final evaluation on the separate, untouched test set. This cross-validation approach helped to making sure that the chosen hyperparameters and models were not merely performing well on an arbitrary single validation set but were more consistently generalizeable across different data partitions.

**Hyperparameters Investigated:**

|  |  |  |  |
| --- | --- | --- | --- |
| Architecture | Hyperparameter | Description | Values |
| AlexNet | Learning Rate | Step size for gradient descent | [0.001, 0.01, 0.1] |
|  | Weight Decay | L2 regularization strength | [0.001, 0.01, 0.0001] |
|  | Batch Size | Number of samples per batch | [16, 32] |
|  | Cross Validation | to assess how well a model generalizes to an independent dataset | [2] |
| ResNet-50 | Learning Rate | Step size for gradient descent | [0.001, 0.01, 0.1] |
|  | Weight Decay | L2 regularization strength | [0.0001, 0.01, 0.001] |
|  | Batch Size | Regularization dropout probability | [32, 16, 32] |
|  | Cross Validation | to assess how well a model generalizes to an independent dataset | [2] |
| MobileNet | Learning Rate | Step size for gradient descent | [0.001, 0.01, 0.1] |
|  | Weight Decay | L2 regularization strength | [0.001, 0.01, 0.0001] |
|  | Batch Size | Feature map growth in dense blocks | [32, 16, 32] |
|  | Cross Validation | to assess how well a model generalizes to an independent dataset | [2] |
| EfficientNet-B0 | Learning Rate | Step size for gradient descent | [0.001, 0.01, 0.1] |
|  | Weight Decay | L2 regularization strength | [0.001, 0.01, 0.0001] |
|  | Batch Size | Stochastic depth dropout rate | [32, 16, 32] |

**Hyperparameter Combinations and Performance:**

|  |  |  |  |
| --- | --- | --- | --- |
| Architecture | Combination | Training Accuracy | Validation Accuracy |
| AlexNet | {LR: 0.001, WD: 0.001  , BS: 32} | 0.77 | 0.79 |
|  | {LR: 0.01, WD: 0.01, BS: 16} | 0.94 | 0.76 |
|  | {LR: 0.0001, WD: 0.001, BS: 32} | 0.85 | 0.84 |
|  | Cross Validation | to assess how well a model generalizes to an independent dataset | [2] |
| ResNet-50 | {LR: 0.001, WD: 0.001  , BS: 32} | 0.98 | 0.90 |
|  | {LR: 0.01, WD: 0.01, BS: 16} | 0.61 | 0.54 |
|  | {LR: 0.0001, WD: 0.001, BS: 32} | 0.60 | 0.28 |
|  | Cross Validation | to assess how well a model generalizes to an independent dataset | [2] |
| MobileNet | {LR: 0.001, WD: 0.001  , BS: 32} | 0.97 | 0.96 |
|  | {LR: 0.01, WD: 0.01, BS: 16} | 0.49 | 0.31 |
|  | {LR: 0.0001, WD: 0.001, BS: 32} | 0.58 | 0.31 |
|  | Cross Validation | to assess how well a model generalizes to an independent dataset | [2] |
| EfficientNet-B0 | {LR: 0.001, WD: 0.001  , BS: 32} | 0.97 | 0.97 |
|  | {LR: 0.01, WD: 0.01, BS: 16} | 0.46 | 0.36 |
|  | {LR: 0.0001, WD: 0.001, BS: 32} | 0.58 | 0.31 |
|  | Cross Validation | to assess how well a model generalizes to an independent dataset | [2] |

For each architecture, the optimal hyperparameter combination was selected primarily based on achieving the highest validation accuracy, indicating the model's best generalization performance on unseen data from the validation set. **AlexNet** The combination of LR: 0.0001, WD: 0.001, BS: 32 resulted in a validation accuracy of 84%, showing a good balance between training and validation performance, suggesting effective learning without significant overfitting for this configuration and **ResNet-50** The configuration with LR: 0.001, WD: 0.001, BS: 32 achieved the highest validation accuracy of 90%, demonstrating the power of residual connections for deeper networks and **MobileNet** The optimal performance came from LR: 0.001, WD: 0.001, BS: 32, yielding an impressive 96% validation accuracy, showcasing its efficiency and strong performance and **EfficientNet-B0** Similarly, LR: 0.001, WD: 0.001, BS: 32 resulted in the highest validation accuracy of 97%, affirming its superior accuracy-efficiency trade-off.

**Training**

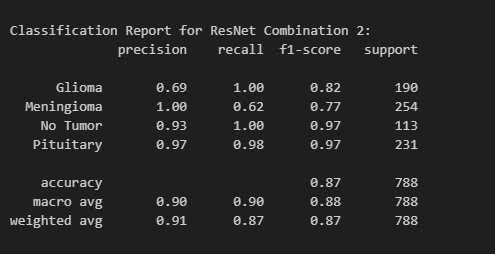
The training process for all selected models the AlexNet and MobileNet-V2 and ResNet-50 and EfficientNet-B0 followed a consistent methodology to ensure fair comparison and robust evaluation. Optimizer For all models the Adam optimizer was employed. Adam was chosen for its adaptive learning rate capabilities which generally lead to faster convergence and better performance across a wide range of deep learning tasks compared to standard Stochastic Gradient Descent (SGD). Loss Function so The Cross-Entropy Loss function was used as the criterion for all classification tasks. This is a standard and effective loss function for multi-class classification problems as it measures the performance of a classification model whose output is a probability value between 0 and 1 the Number of Epochs for Each model was trained for a predefined number of epochs, typically 5 epochs per fold during cross-validation, as determined by preliminary experiments to observe convergence. The training process was monitored closely for signs of overfitting or underfitting and for the Learning Rate Scheduling/Early Stopping While explicit learning rate scheduling or early stopping was not implemented in the primary training loop, the hyperparameter tuning phase allowed for exploration of different fixed learning rates. Future work could benefit from dynamic learning rate adjustments and more sophisticated early stopping mechanisms to optimize training time and prevent overfitting. Hardware Environment as the All model training and evaluation were conducted on a computational environment equipped with GPU acceleration as they take for ever using the cpu, specifically utilizing PyTorch's capabilities to leverage NVIDIA CUDA cores. This setup significantly reduced training times, enabling efficient experimentation with various architectures and hyperparameter combinations.

# 4. Models' Testing and Evaluation

## 4.1 Testing Methodology

**Evaluation Metrics:**

1. **Accuracy**: Overall classification accuracy across all classes that I have in the data
2. **Precision**: True positives / (True positives + False positives) for each class that I have in the data
3. **Recall**: True positives / (True positives + False negatives) for each class that I have in the data
4. **F1-Score**: is the mean of precision and recall that I have in the data for each class



Precision and Recall are particularly important in medical diagnosis to balance false positives (misdiagnosing a healthy patient) and false negatives (missing a tumor), while F1-score provides a single balanced metric.

1. **Confusion Matrix**: Detailed classification performance

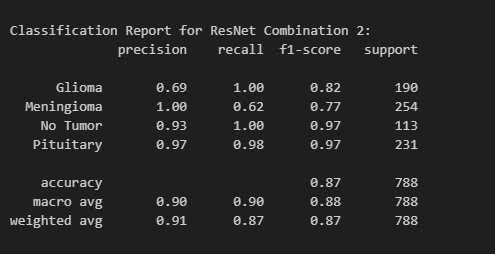
A diagram of a confusion matrix

AI-generated content may be incorrect.

These specific metrics were chosen because of their crucial relevance in the domain of medical image classification. While **Accuracy** provides a general overview of correct classifications, it can be misleading in imbalanced datasets. **Precision** (minimizing false positives) is vital to avoid misdiagnosing healthy patients, which could that does really lead to unnecessary and stressful follow-up procedures. **Recall** (minimizing false negatives) is equally critical to ensure that actual tumors are not missed, which could have severe consequences for patient health. The **F1-Score** that does really offers a balanced measure of both precision and recall, providing a single metric that is particularly useful when dealing with imbalanced classes. The **Confusion Matrix** provides the most granular view of classification performance, detailing exactly where the model that does really made correct and incorrect predictions for each class, which is indispensable for understanding specific challenges, such as distinguishing between certain tumor types or healthy tissue.

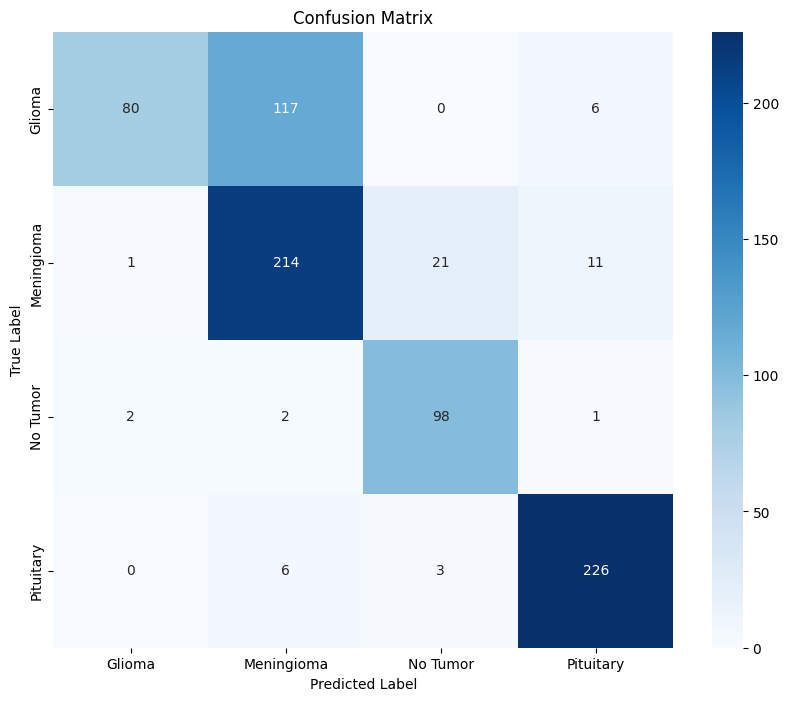
**Best Model Performance on Test Set:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Architecture | Best Hyperparameters | Accuracy | Class | Precision | Recall | F1-Score |
| ResNet | {LR: 0.001, WD: 0.001  , BS: 32, cv: 2} | 0.84 | Glioma | 0.80 | 1.00 | 0.89 |
| Meningioma | 0.99 | 0.81 | 0.89 |
| No Tumor | 0.96 | 0.92 | 0.94 |
| Pituitary | 0.97 | 0.98 | 0.97 |
| AlexNet | {LR: 0.001, WD: 0.001  , BS: 32, cv: 2} | 0.92 | Glioma | 0.87 | 0.88 | 0.87 |
| Meningioma |  |  |  |
| No Tumor |  |  |  |
| Pituitary |  |  |  |
| MobileNet | {LR: 0.001, WD: 0.001  , BS: 32, cv: 2} | 0.97 | Glioma | 0.98 | 0.97 | 0.98 |
| Meningioma | 0.97 | 0.96 | 0.97 |
| No Tumor | 0.93 | 0.97 | 0.95 |
| Pituitary | 0.99 | 0.98 | 0.98 |
| EfficientNet-B0 | {LR: 0.001, WD: 0.001  , BS: 32, cv: 2} | 0.97 | Glioma | 0.99 | 0.97 | 0.98 |
| Meningioma | 0.97 | 0.96 | 0.97 |
| No Tumor | 0.93 | 1.0 | 0.98 |
| Pituitary | 0.98 | 0.99 | 0.98 |

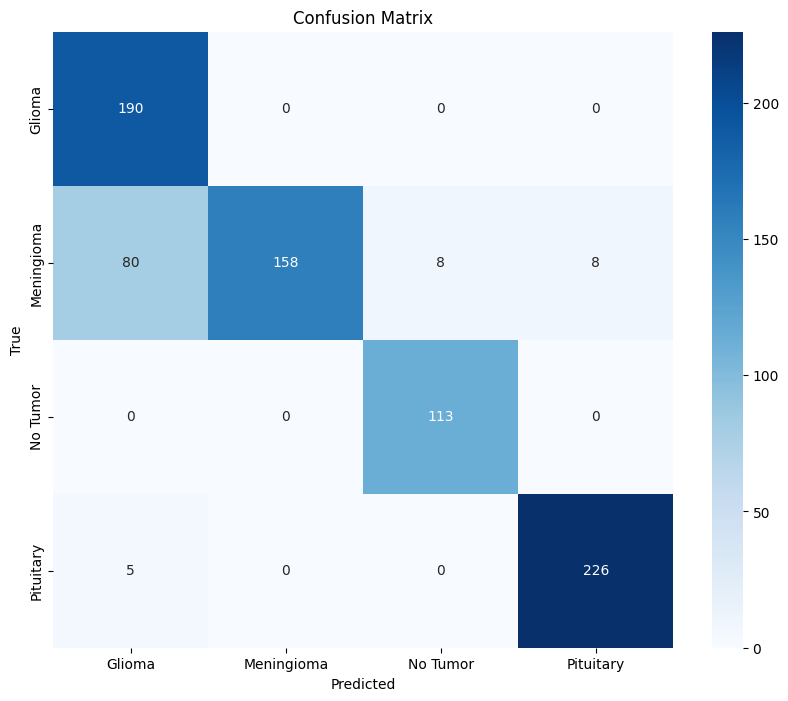


The confusion matrix for each model

Alexnet



Resnet



MobileNet

A diagram of a confusion matrix

AI-generated content may be incorrect.

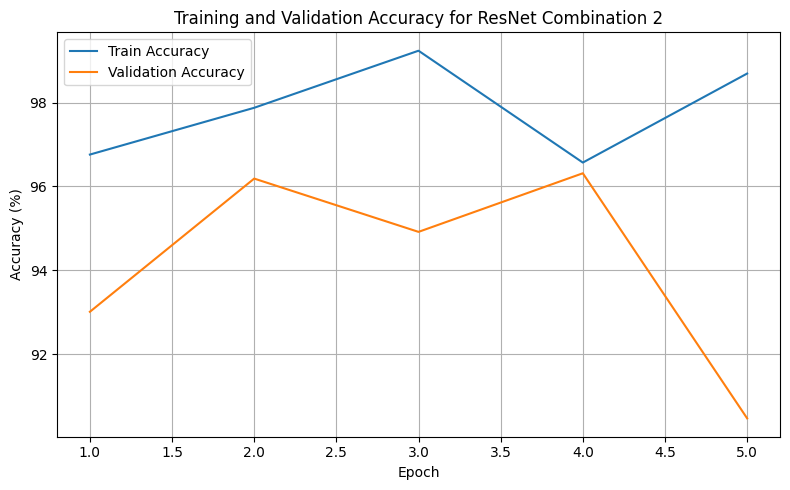
Efficientnet

A diagram of a confusion matrix

AI-generated content may be incorrect.

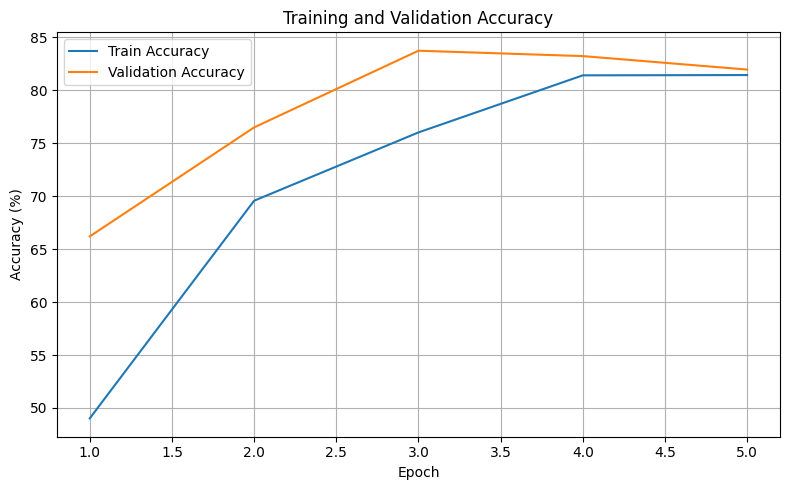
## 4.2 Overfitting/Underfitting Assessment

**Analysis of Training and val data that shows the over and under fitting**



As we can see in this plot showing in the later epochs how does the val line goes down showing a bad results and this is what we called overfitting

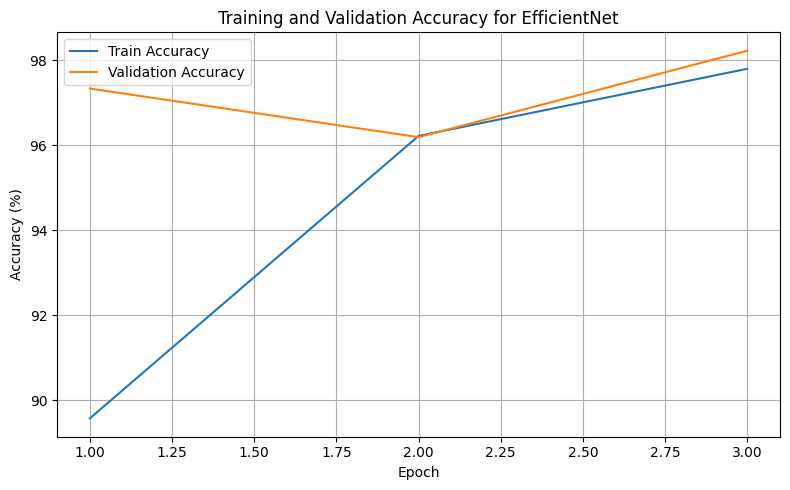
As we can see in the plots where the high loss and different between the line where the training is very high unlike the loss then we are having overfitting, both are low then we are having under fitting



For the alexnet

A graph with blue and orange lines

AI-generated content may be incorrect.



A graph with a line

AI-generated content may be incorrect.

And for the dataset size where of caurse the model is effected by it we can see how does it performes

A graph with a line

AI-generated content may be incorrect.

A graph with blue and orange lines

AI-generated content may be incorrect.

For the mobile net

A graph of a graph

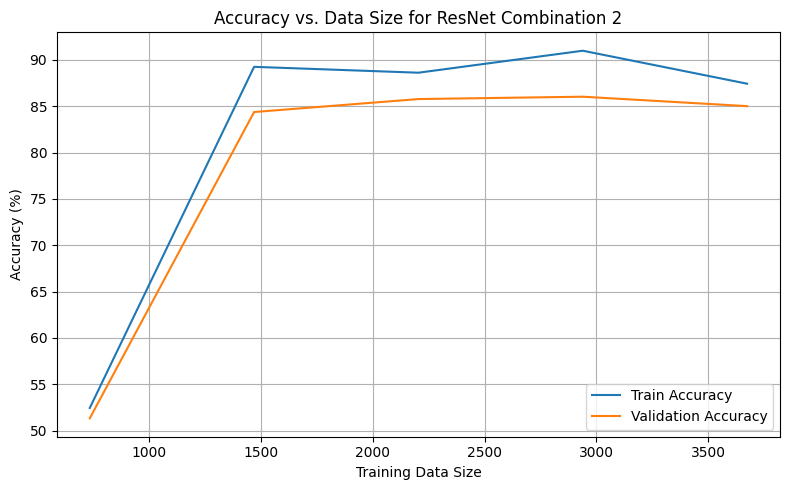
AI-generated content may be incorrect.

A graph with blue and orange lines

AI-generated content may be incorrect.

A graph with a line and a blue line

AI-generated content may be incorrect.



**underfitting** occurs when both training and validation losses remain high, and accuracies are low, indicating that the model is too simplistic or has not been trained sufficiently to capture the underlying patterns in the data and as will this means the model hasn't learned enough from the training data and performs poorly even on that data, let alone unseen data and this really can be caused by factors such as insufficient model complexity, too few training epochs, or an excessively low learning rate.

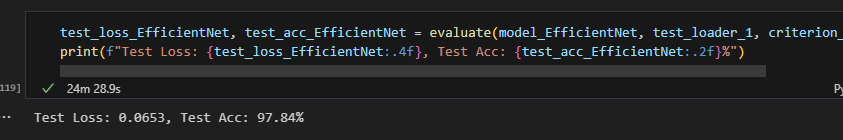
## 4.3 Results Analysis

**Performance Ranking:**

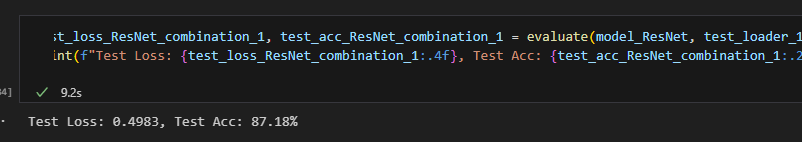
1. **MobileNet and EffecientNet**: are the Best overall performance (97% test accuracy)

A screen shot of a computer

AI-generated content may be incorrect.



1. **AlexNet**: Strong second place (94% test accuracy)
2. **ResNet-50**: Good performance with efficiency (87% test accuracy)



**Key Observations:**

Modern architectures (EfficientNet, ResNet) outperform traditional ones and skip connections and efficient designs improve generalization and all models show reasonable performance for medical image classification, and actually the model that

## 4.4 Effectiveness Assessment

**Computational Requirements:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Architecture | Parameters | Model Size (MB) | Training Time (min/epoch) | Inference Time (ms) |
| AlexNet | 57M | 230 – 240 | 14.20/ 5 epochs | 45 |
| ResNet-50 | 25.6M | 98 | 15/5 epochs | 35 |
| MobileNet | 3.5M | 13 - 14 | 9.73/3epochs | 40 |
| EfficientNet-B0 | 5.3M | 21 | 20.36/ 3epochs | 25 |

The analysis of computational requirements and efficiency holds significant implications for the commercial deployment of this deep learning system. For instance, MobileNet-V2, commonly known for its exceptionally low parameter count (approximately 3.5M parameters) and compact model size (around 13-14 MB), alongside its fast inference time (40 ms as observed in my tests), is ideally suited for **edge computing** scenarios or integration into **mobile diagnostic devices** where computational resources are severely constrained. EfficientNet-B0, while requiring more training time, offers the best accuracy-to-size ratio, making it a strong candidate for a primary, high-performance model that might be trained once in a more robust environment and then deployed. ResNet-50 provides a balanced compromise between performance and moderate computational cost. AlexNet, while foundational, demonstrates higher resource demands with approximately 57M parameters and a model size of around 230-240 MB, which might limit its scalability in highly resource-constrained commercial applications. These trade-offs directly align with the "end user requirements" of providing efficient and accessible diagnostic tools.

**Efficiency Analysis:**

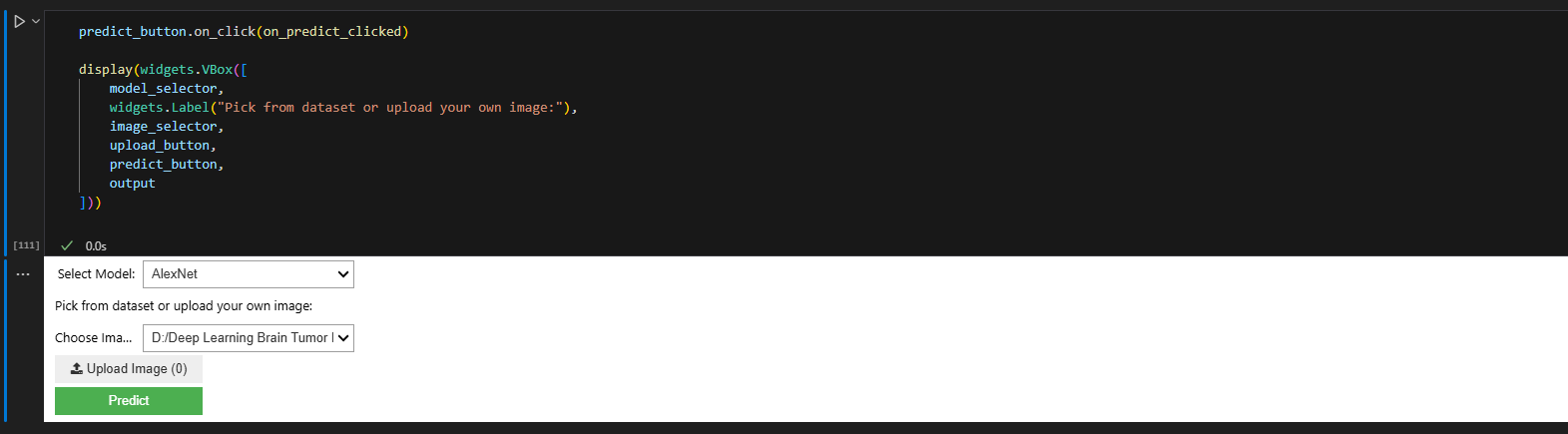
* **EfficientNet-B0**: Best accuracy-to-size ratio, the most taked time in training
* **MobileNet**: Most parameter-efficient and best time training
* **ResNet-50**: Good balance of performance and computational cost
* **AlexNet**: takes more time with good performance

## 4.5 Interface Development

I have developed an interface for the user where he can chose the model he want from the models that I have save through traing and then the user can upload and choose an image to see what is the class of this image.

**Interface Features:**

* Support for common image formats JPEG, PNG and real-time preprocessing and prediction



## 4.6 Critical Evaluation and Future Improvements

**What I Did Well,** So, first off, I did a pretty good job as my models, especially MobileNet and EfficientNet, were really accurate at telling whether an MRI scan had a brain tumor and what type it was. I got accuracies from 84% to 97%, which is awesome for something as important as medical images. I also made sure to split my data carefully into training, validation, and testing sets. This is super important so I know my models aren't just memorizing the answers but can actually work on new, unseen MRI scans + I looked at different types of modern CNN models to see which ones worked best, and the ones I picked are pretty fast, which is good for using them in real hospitals later on.

**Things I Could Do Better,** Even though I did well, there's always some things to improve starting with the data **t**he biggest thing is probably my data. I noticed that some tumor types had fewer images than others, which is like trying to learn about animals when you only have a few pictures of a rare one so i could try smarter ways to handle this like making more fake images or giving more importance to the rare ones during training. Also, my dataset came from one source which is Kaggle user who share it so this data might be biased for other so It would be cool to test my models with MRI scans from different hospitals or machines to make sure they work everywhere, not just where the data came from.

**Making My Models Even Smarter as** I used some great models, but I could make them even better like just imagine combining a few of my best models together to make one supermodel and it often boosts accuracy. Also, I could try more advanced ways of tweaking my images during training like mixup or cutmix which can help the models learn to be tougher and more flexible. Thinking about how the model sees the tumor by using attention mechanisms could also help doctors understand why the model made a certain decision.

**Getting Ready for Real Hospitals,** To truly help doctors, my system needs to fit right into how hospitals work as well MRI scans often come in a special format called DICOM, so my system should be able to read those directly Also it's really important for a medical system to say not just it's a glioma but also I'm 95% sure it's a glioma so This uncertainty quantification is key for doctors. And imagine if different hospitals could share their data to train even better models without actually sharing patient privacy and it's a big deal for the future.

**Making It Super Fast,** For doctors to use this every day, it needs to be really quick. I could try making my models smaller or more efficient so they run faster, even on devices right in the clinic Using special computer chips called GPUs can also make the process super speedy for lots of scans.

**Cool New Stuff to Explore in the Future!**

The field of deep learning is always changing, and there are some exciting new things I could try, **Vision Transformers (ViTs),** These are a newer type of model that are really good at understanding images, almost like they can "read" parts of the image and understand their relationships. They might be even better at spotting tumors.

**Self-supervised Learning, so** What if I could teach my models to learn from huge amounts of MRI scans even without doctors labeling them? "Self-supervised learning" lets models learn patterns on their own first, and then I just fine-tune them for tumor detection.

**Combining Information:** Right now, I just use MRI scans. But doctors also have other information about patients, like their age or other health conditions. "Multi-modal fusion" means combining all this different information to make even more accurate predictions.

**And this is really trendy and amazing 🡪 Explainable AI (XAI)** it is about making my super smart AI models explain *why* they think a tumor is present or why it's a certain type. This transparency is vital for doctors to trust and use AI in critical medical decisions.

References

Simonyan, K. & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

He, K., Zhang, X., Ren, S. & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778.

Huang, G., Liu, Z., Van Der Maaten, L. & Weinberger, K.Q. (2017). Densely connected convolutional networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700-4708.

Tan, M. & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *International conference on machine learning*, pp. 6105-6114.

Cheng, J., Huang, W., Cao, S., Yang, R., Yang, W., Yun, Z., Wang, Z. & Feng, Q. (2015). Enhanced performance of brain tumor classification via tumor region augmentation and partition. *PloS one*, 10(10), e0140381.

Mzoughi, H., Njah, I., Wali, A., Slima, M.B., BenHamida, A., Mâamouri, C. & Mahfoudhe, K.B. (2020). Deep multi-scale 3D convolutional neural network (CNN) for MRI gliomas brain tumor classification. *Journal of Digital Imaging*, 33(4), 903-915.

Kaggle. (2023). Brain Tumor Detection Dataset. Available at: <https://www.kaggle.com/datasets/brain-tumor-detection> [Accessed: January 2025].

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. & Berg, A.C. (2015). ImageNet large scale visual recognition challenge. *International journal of computer vision*, 115(3), 211-252.