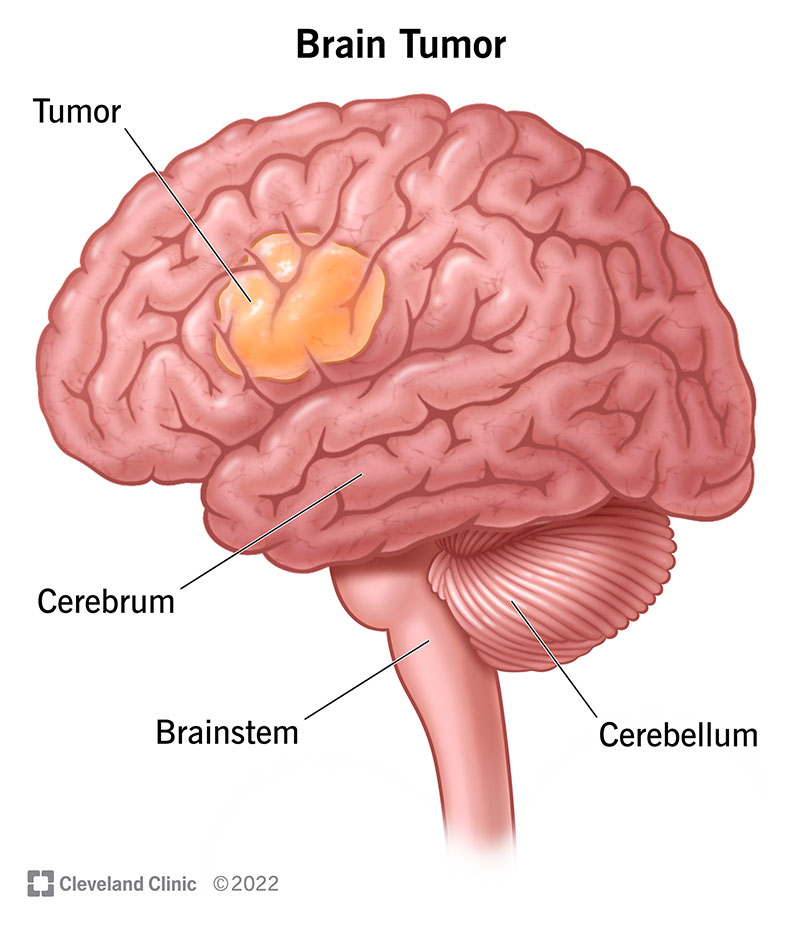
**Brain Tumor Classification in MRI Images Using Transfer Learning Techniques**

**1. Introduction**

**1.1 What Are Brain Tumors?**

A brain tumor is a cancerous or non-cancerous mass or growth of abnormal cells in the brain. Nearby locations include nerves, the pituitary gland, the pineal gland, and the membranes that cover the surface of the brain. Brain tumors that begin in the brain are called primary brain tumors. Sometimes, cancer spreads to the brain from other parts of the body. These tumors are known as secondary brain tumors, also called metastatic brain tumors.



***Figure 1.1 : Brain diagram showing tumor location and labeled parts***

**Source:**[**Cleveland Clinic | Brain Tumor Illustration**](https://my.clevelandclinic.org/health/diseases/6149-brain-cancer-brain-tumor)

**1.2 The Role of Deep Learning in Brain Tumor Detection**

Recent advancements in deep learning have revolutionized the field of medical image analysis. Convolutional Neural Networks (CNNs), in particular, have shown remarkable performance in image classification tasks, including the detection and classification of brain tumors in MRI scans.

These models can automatically learn complex features from raw image data without the need for manual feature engineering. By leveraging large datasets and transfer learning techniques, deep learning provides a scalable and efficient solution for assisting radiologists in accurate and timely tumor diagnosis.

**1.3 Objective**

The primary objective of this study is to apply transfer learning techniques to achieve the highest possible performance in classifying brain tumors from MRI images. This is accomplished by utilizing a range of pre-trained convolutional neural network (CNN) architectures, including EfficientNetV2B0, MobileNetV3Small, ResNet50, DenseNet102, InceptionV3, and Xception.

These models, originally trained on large-scale datasets such as ImageNet, provide a strong foundation for extracting deep and relevant features from medical images, even when the available dataset is limited.

By reusing and fine-tuning these models, the project aims to reduce training time and computational resources while improving classification accuracy. Moreover, this study seeks to compare the performance of these architectures in terms of accuracy, efficiency, and generalization, in order to identify the most suitable model for potential deployment in computer-aided diagnostic systems.

**1.4 Pretrained Models Overview**

#### **1.4.1 EfficientNet (Efficient Network)**

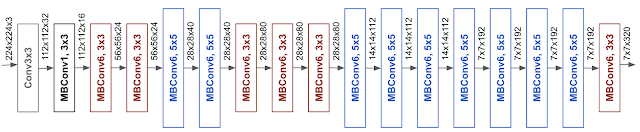
EfficientNet is a family of [convolutional neural networks (CNNs)](https://www.geeksforgeeks.org/introduction-convolution-neural-network/) that aims to achieve high performance with fewer computational resources compared to previous architectures. It was introduced by Mingxing Tan and Quoc V. Le from Google Research in their 2019 paper "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." The core idea behind EfficientNet is a new scaling method that uniformly scales all dimensions of depth, width, and resolution using a compound coefficient.

The EfficientNet-B0 network consists of:

1. **Stem**
   * Initial layer with a standard convolution followed by a [batch normalization](https://www.geeksforgeeks.org/batch-normalization-implementation-in-pytorch/) and a ReLU6 activation.
   * Convolution with 32 filters, kernel size 3x3, stride 2.
2. **Body**
   * Consists of a series of MBConv blocks with different configurations.
   * Each block includes depthwise separable convolutions and squeeze-and-excitation layers.
   * Example configuration for MBConv block:
     + Expansion ratio: The factor by which the input channels are expanded.
     + Kernel size: Size of the convolutional filter.
     + Stride: The stride length for convolution.
     + SE ratio: Ratio for squeeze-and-excitation.
3. **Head**
   * Includes a final convolutional block, followed by a global average pooling layer.
   * A fully connected layer with a softmax activation function for classification.

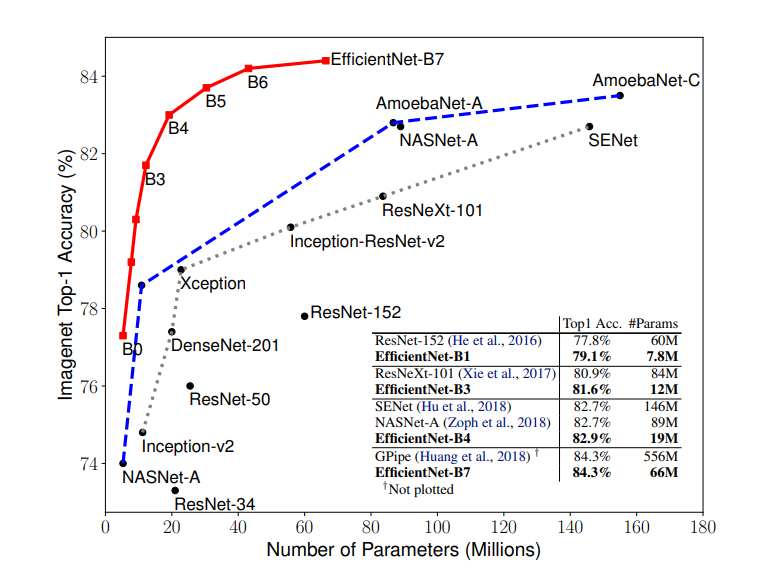
**EfficientNet-B0 Detailed Architecture**

EfficientNet uses a technique called compound coefficient to scale up models in a simple but effective manner. Instead of randomly scaling up width, depth, or resolution, compound scaling uniformly scales each dimension with a certain fixed[set](https://www.geeksforgeeks.org/set-in-cpp-stl/) of scaling coefficients. Using this scaling method and AutoML, the authors of EfficientNet developed seven models of various dimensions, which surpassed the state-of-the-art accuracy of most convolutional neural networks, and with much better efficiency.



***Figure 1.2: EfficientNet Architecture***

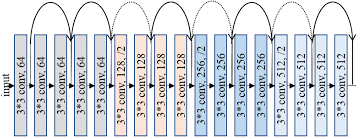
As you can see from the performance graph, EfficientNet uses fewer parameters and achieves very high accuracy.



***Figure 1.3: Model Size vs. ImageNet Accuracy***

#### **1.4.2 ResNet50 (Residual Network)**

Kaiming He et al. proposed a residual learning framework that has made the training of deep neural networks easier and faster. One of the most successful residual networks that has achieved great results on the ImageNet dataset is ResNet-50 which is summarized in Table 1. However, ResNet50 has a complex architecture which makes it hard to implement on mobile devices. **2**



***Figure 1.3: ResNet Architecture***

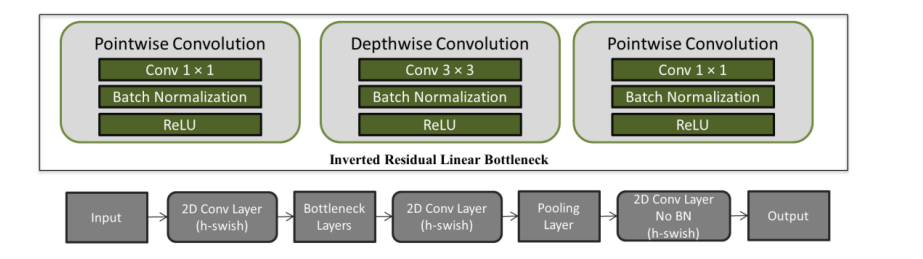
#### **1.4.3 MobileNet (Mobile Network)**

The MobileNet models are based on depth-wise separable convolution instead of a standard convolution. Standard convolution performs the channel and spatial wise computation in one step while the depth-wise separable convolution splits the process into two steps: Point-wise convolution and depth-wise convolution. The depthwise convolution applies a filter to each channel of the image input then the pointwise convolution applies a 1 × 1 convolution to combine the outputs of the former layer. This leads to a drastic reduction in the computational power required and the model complexity.

The three versions of the MobileNet models has been improved ever since they were developed in 2017. The main purpose of the MobileNets was to implement a light CNN model on mobile devices with a reduced model size ( < 10 MB ) and a reduced number of parameters.

MobileNetV3 was introduced in 2019 and it was developed by dropping complex layers and using H-swish function instead of standard ReLU to further increase the network efficiency and accuracy. Figure 1 summarizes the architecture of MobileNetV3.

However, in some cases MobileNet models must be trained for a large number of epochs to achieve a good accuracy. **2**



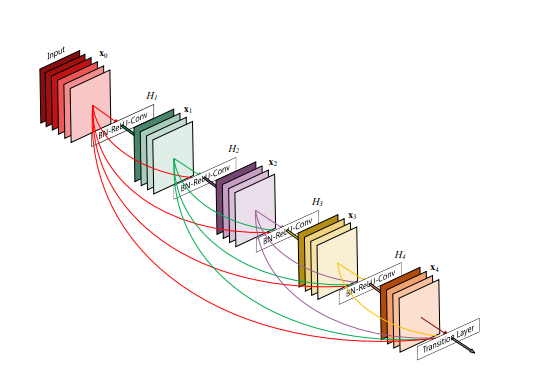
***Figure 1.4: MobileNet Architecture***

**2 Deep Residual Learning for Image Recognition" by Kaiming He et al. (2015)**

**Searching for MobileNetV3" by Howard, Sandler, Chu, et al. (2019)**

#### **1.4.4 DenseNet121 ( Dense Network )**

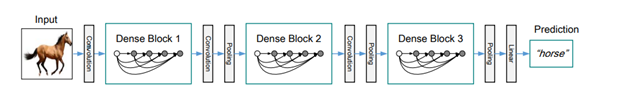
DenseNet-121 is a convolutional neural network architecture designed for image classification and related tasks such as segmentation. What sets DenseNet-121 apart is its distinctive connectivity pattern known as dense connectivity, where each layer is directly connected to all subsequent layers in a feed-forward fashion. This approach enables more effective feature reuse and enhances gradient flow, which helps in training very deep networks by mitigating the vanishing gradient problem. Additionally, DenseNet-121 is more parameter-efficient than traditional CNNs, as it avoids learning redundant features, each layer receives a collective input from all preceding layers.



***Figure 1.5: A 5-layer dense block with growth rate of k =4.***

***Each layer takes all preceding feature-maps as input***

The architecture also incorporates bottleneck layers, which reduce the number of feature maps before applying convolution operations, thereby decreasing computational complexity. Furthermore, it employs transition layers with compression to reduce the spatial dimensions and the number of feature maps, improving the overall efficiency of the network. These design choices make DenseNet-121 a powerful and resource-efficient model for various deep learning applications. **3**



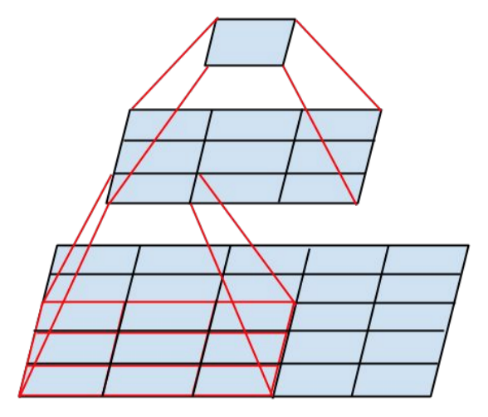
***Figure 1.6 : A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.***

**3 Huang, Gao, et al. "Densely Connected Convolutional Networks." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4700–4708.**

#### **1.4.5 InceptionV3 Network**

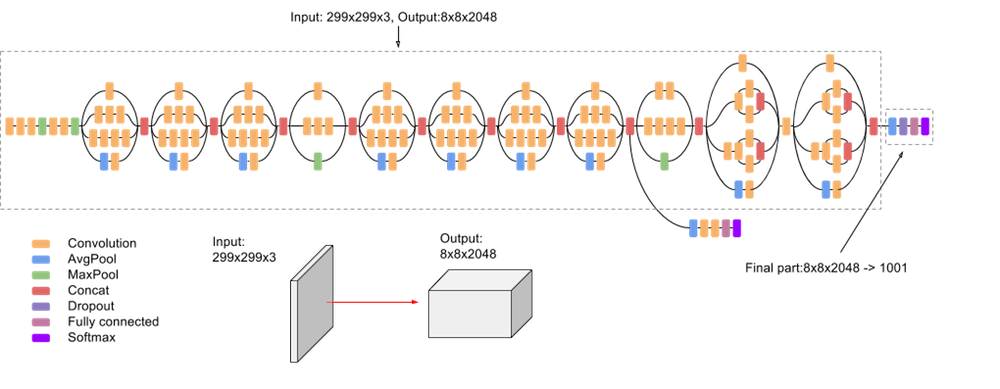
Inception v3 is a convolutional neural network architecture designed to optimize computational efficiency by refining and extending the original Inception (GoogLeNet) models. Proposed in the 2015 paper "Rethinking the Inception Architecture for Computer Vision" by Christian Szegedy and colleagues, Inception v3 offers significant improvements in parameter reduction and memory usage when compared to earlier deep networks such as VGGNet. However, adapting Inception-based models for different tasks requires careful balancing to avoid compromising their computational advantages.

To address this, Inception v3 introduces a range of architectural innovations aimed at making the model more flexible and efficient. These include factorized convolutions, which replace larger convolutions (e.g., 5×5) with consecutive smaller ones (e.g., two 3×3 convolutions), thereby reducing parameter count and improving training speed. The architecture also employs asymmetric convolutions, such as replacing a 3×3 filter with a 1×3 followed by a 3×1 convolution, which preserves the receptive field while lowering computational cost.



***Figure 1.7: Mini-network replacing the 5 × 5 convolutions***

Additionally, an auxiliary classifier , a lightweight CNN inserted mid-network acts as a regularizer during training by contributing to the total loss and improving gradient flow. Grid size reduction techniques are also enhanced beyond basic pooling by employing more efficient downsampling methods to preserve information while reducing spatial dimensions. Collectively, these strategies are carefully integrated into a progressive and modular architecture that balances depth, accuracy, and performance, making Inception v3 highly suitable for various computer vision tasks with minimal computational overhead. **4**



***Figure 1.8: InceptionV3 Architecture***

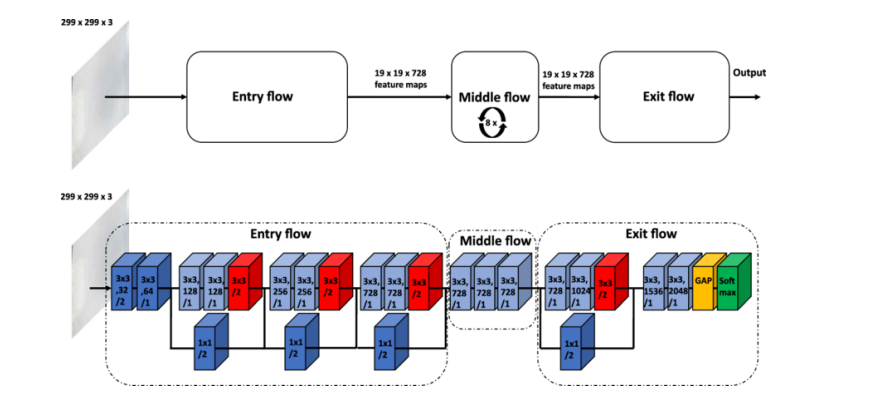
**4 Szegedy, Christian, et al. "Rethinking the Inception Architecture for Computer Vision." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2818–2826.**

#### **1.4.6 Xception Network**

Xception (Extreme Inception) is an advanced convolutional neural network architecture that extends the Inception family by replacing Inception modules with depthwise separable convolutions. This structural change significantly reduces the number of parameters while maintaining or even improving performance.

The Xception architecture is organized into three main components: the Entry Flow, the Middle Flow, and the Exit Flow. The Entry Flow serves as the initial processing stage and consists of three convolutional blocks, each integrating convolutional layers, batch normalization, ReLU activation functions, and residual (skip) connections to enhance feature propagation and gradient flow.

The Middle Flow is composed of eight identical blocks, each employing depthwise separable convolutions followed by batch normalization, ReLU activation, and residual connections, enabling efficient extraction of deeper and more abstract features. The Exit Flow contains a final block with three convolutional layers, which reduces the spatial dimensions and increases the depth of the feature maps, preparing the data for final classification. **5**



***Figure 1.9: Xception Architecture and Flows***

**5 Advanced Machine Learning Slides – Adnan Salman**

**2. Literature Review**

* Transfer learning using pretrained convolutional neural networks (CNNs) has become a prominent approach in medical image analysis. By reusing feature extractors trained on large-scale datasets like ImageNet, these models can be adapted to medical tasks with limited data, often yielding high accuracy and faster convergence.
* EfficientNet, introduced by Tan and Le (2019), brought significant improvements in both accuracy and computational efficiency through a novel compound scaling approach. In the context of medical imaging, EfficientNet has shown superior performance compared to traditional CNN architectures such as ResNet, DenseNet, Inception, and MobileNet.
* Several studies have demonstrated the effectiveness of EfficientNet in brain tumor classification using MRI images. For example, a study by Sitaula et al. (2023) conducted a comparative analysis of multiple pretrained models, including ResNet50, DenseNet121, MobileNetV2, and EfficientNetB3. The results showed that EfficientNetB3 achieved the highest accuracy (99.96%) and demonstrated better generalization, particularly on multi-class MRI brain tumor datasets.
* Another recent work by Choudhury et al. (2024) investigated the classification of Alzheimer’s disease using MRI and found that EfficientNet-B5 outperformed ResNet50 and InceptionV3 in terms of both accuracy and F1-score, while maintaining a relatively smaller number of parameters.
* In dermatology imaging, Abdalla et al. (2023) compared EfficientNetB2 with ResNet101V2, MobileNetV3, and InceptionV3. EfficientNet not only achieved the best accuracy but also converged faster and exhibited less overfitting during training, highlighting its applicability beyond brain imaging.
* In addition, Kumar et al. (2024) integrated attention mechanisms with EfficientNetV2 and evaluated its performance on brain MRI tumor classification. Their findings revealed that the hybrid EfficientNetV2-M model outperformed DenseNet121 and InceptionV3, achieving up to 99.5% accuracy with fewer trainable parameters and reduced computational cost.

**3. Dataset**

This dataset consists of a private collection of T1, contrast-enhanced T1, and T2 magnetic resonance images separated by brain tumor type. The images were collected without any type of marking or patient identification, interpreted by radiologists and provided for study purposes. The images are separated by astrocytoma, carcinoma, ependymoma, ganglioglioma, germinoma, glioblastoma, granuloma, medulloblastoma, meningioma, neurocytoma, oligodendroglioma, papilloma, schwannoma and tuberculoma.

**3.1 Dataset Description**

This dataset, published on Kaggle, is a private collection of MRI brain scans acquired using T1-weighted, contrast-enhanced T1 (T1c), and T2-weighted modalities. All images were anonymized and interpreted by certified radiologists, and no patient-identifying information is included.

The dataset is structured into 44 distinct classes, which represent subtypes of 14 primary brain tumor categories, including:

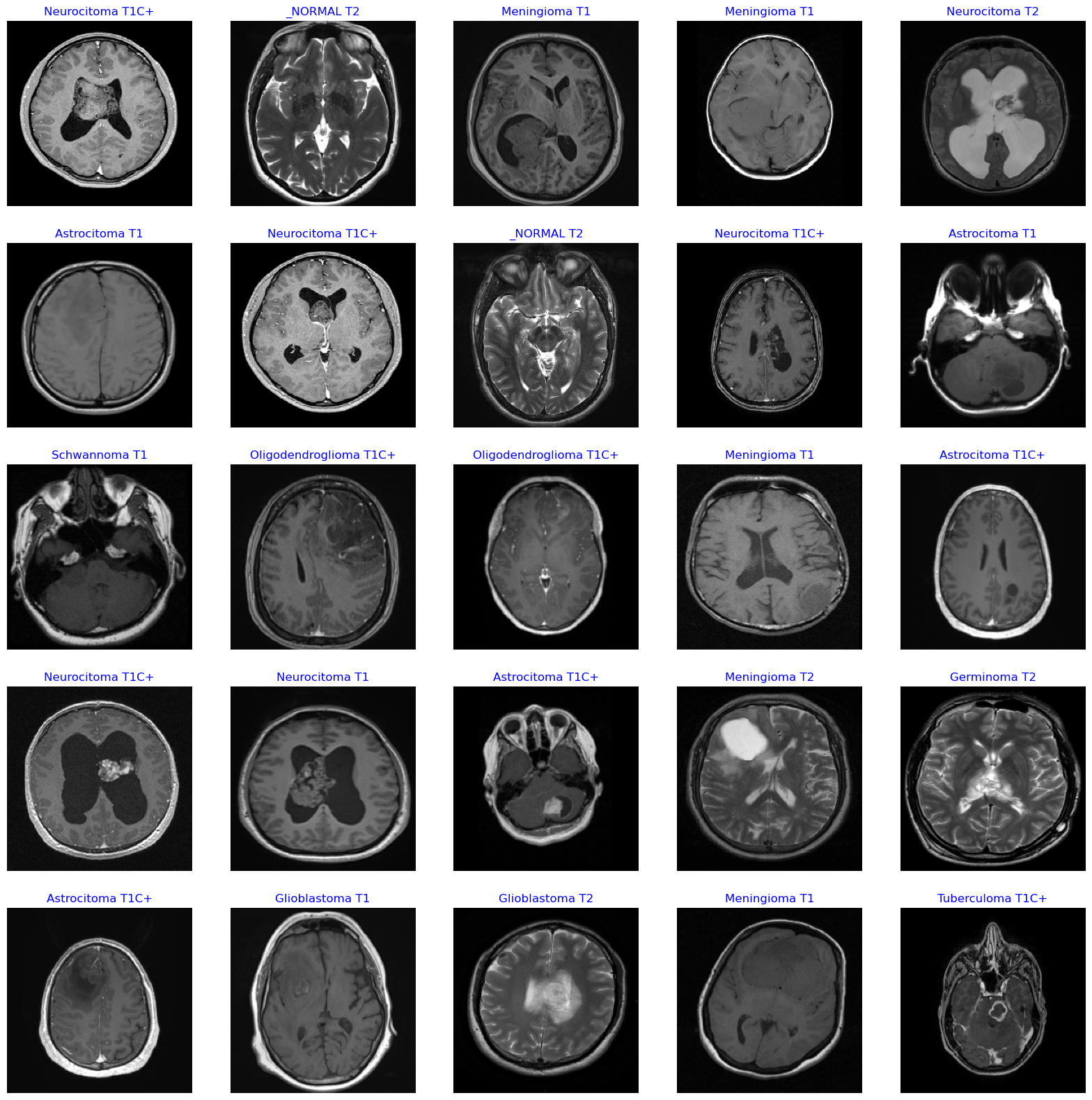
1. **Astrocytoma**
2. **Carcinoma**
3. **Ependymoma**
4. **Ganglioglioma**
5. **Germinoma**
6. **Glioblastoma**
7. **Granuloma**
8. **Medulloblastoma**
9. **Meningioma**
10. **Neurocytoma**
11. **Oligodendroglioma**
12. **Papilloma**
13. **Schwannoma**
14. **Tuberculoma**

Each of these main tumor types is further subdivided into histological or anatomical subcategories, resulting in a total of 44 fine-grained labels. The dataset poses a multi-class classification task with 44 classes, requiring the model to distinguish between closely related tumor variants.

**3.2 Image and Data Characteristics**

* **Image type**: 2D grayscale MRI slices
* **Total Number of Samples**: 4,478 images
* **Modalities included:** T1, T1c (contrast-enhanced), T2
* **Image format**: JPG / JPEG
* **Typical resolution:** Various (resized during preprocessing)
* **Total number of classes:** 44
* **Task type**: Multi-class classification
* **Imbalance:** The dataset is class-imbalanced; some tumor subclasses are significantly underrepresented

In this project, the 44-class version of the dataset was used to train and evaluate multiple deep CNN architectures (EfficientNetV2B0, ResNet50, DenseNet121, MobileNetV3Small, InceptionV3, and Xception), fine-tuned via transfer learning. Given the relatively low volume of data and the highly granular, imbalanced nature of the classes, transfer learning was essential to leverage pretrained feature representations and improve generalization. Additionally, data augmentation and stratified splitting were employed to increase data variability, mitigate overfitting, and ensure a balanced representation of all classes during training and evaluation, thereby enhancing model robustness and reliability.



***Figure 3.1: Samples from Dataset***

**For more information check the following:**

* [Kaggle | Brain Tumor MRI Images 44 Classes](https://www.kaggle.com/datasets/fernando2rad/brain-tumor-mri-images-44c?sort=published)
* [Dataset Author | Fernando Feltrin](https://www.kaggle.com/fernando2rad)

**4. Methodology**

**4.1 Hardware and Environment**

#### **4.1.1 Libraries and Tools**

**The project leverages a set of essential Python libraries categorized as follows:**

1. **Deep Learning Framework**  
   • TensorFlow / Keras: Core framework for building, training, and fine-tuning convolutional neural networks (CNNs), including support for:  
       – Model architecture and layers  
       – Optimizers and loss functions  
       – Data augmentation (ImageDataGenerator)  
       – Pretrained models: EfficientNetV2, ResNet50, DenseNet, MobileNetV3, Xception
2. **Data Processing and Analysis**  
   • NumPy, Pandas: For numerical computations and structured data manipulation  
   • OpenCV (cv2), PIL: For image reading, resizing, and preprocessing  
   • Scikit-learn: For train-test splitting, label encoding, and evaluation metrics
3. **Visualization**  
   • Matplotlib, Seaborn: To visualize dataset distributions, training performance, and model evaluation
4. **Web Deployment**  
   • Streamlit: Used to deploy the trained model via an interactive web application that accepts images, displays predictions, and overlays visual explanations
5. **Utilities**  
   • OS, Shutil, Pathlib, Glob, Zipfile: For file management, directory traversal, and dataset handling  
   • Warnings: Suppresses irrelevant runtime warnings for cleaner output

#### **4.1.2 Computing Resources**

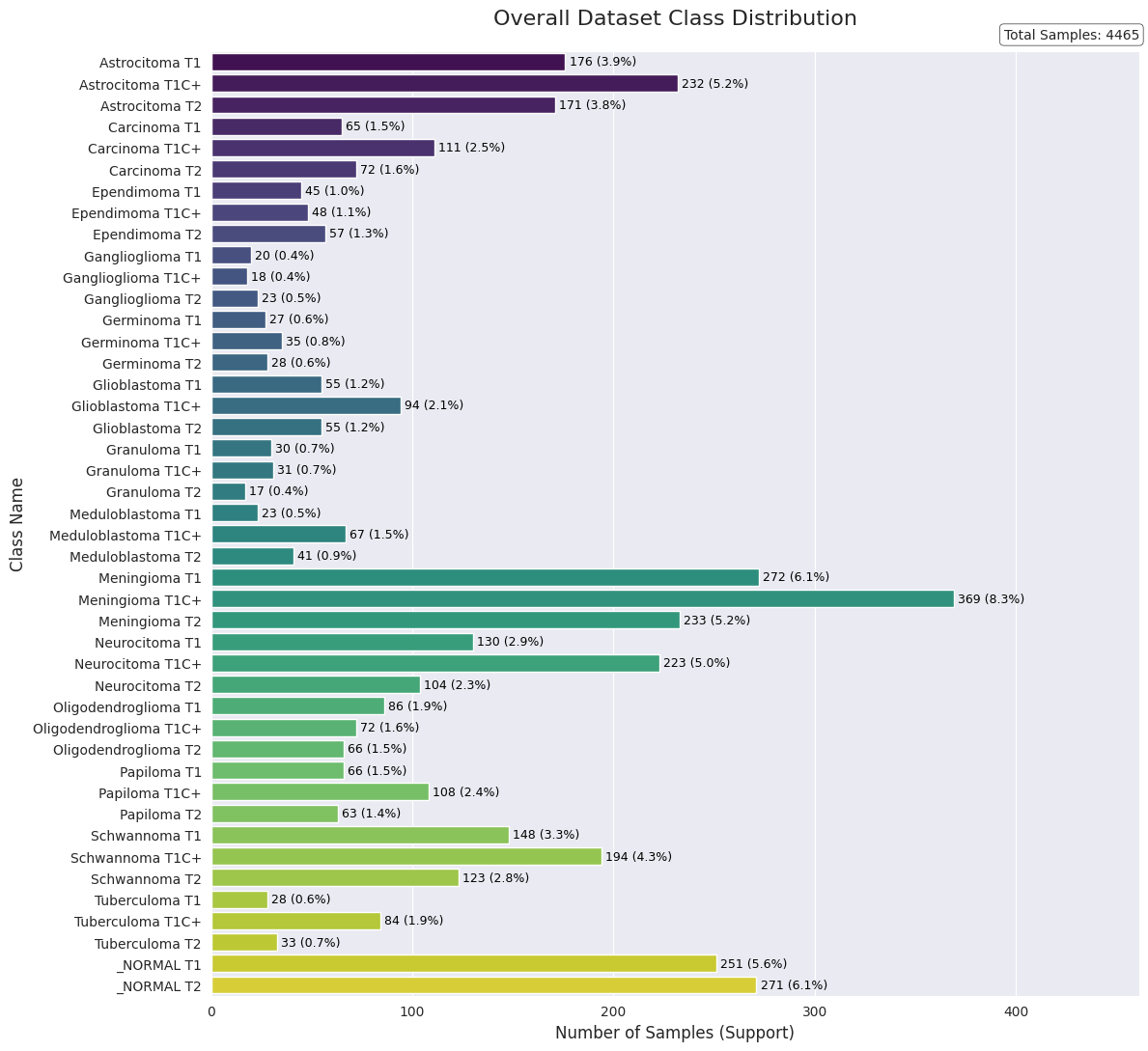
The entire training process was conducted on Google Colab, utilizing NVIDIA Tesla T4 GPUs:

* **Platform**: Google Colab
* **GPU**: NVIDIA Tesla T4 (16 GB VRAM)
* **Memory**: 16 GB RAM
* **CPU**: Intel Xeon (2 cores, 2.20 GHz typical)
* **Parallelism**: Multiple Colab sessions were run in parallel for multiple pretrained models and across the models themselves using different tuning to speed up the process of obtaining results.

**4.2 Data Preprocessing**

#### **4.2.1 Data Split**

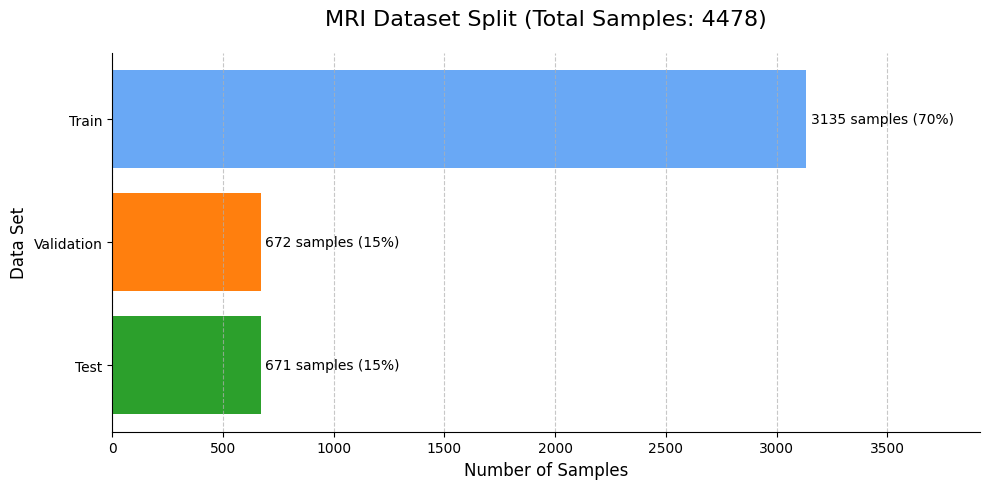
The raw dataset contains 44 subfolders, each corresponding to a distinct tumor subtype. These subfolders were parsed to extract image file paths and labels, which were then numericallyencodedusingLabelEncoder to prepare them for supervised learning.



***Figure 4.1: Overall Dataset Class Distribution***

Following this, the entire dataset was split into three parts:

* **70% for training** (3,125 samples),
* **15% for validation** (670 samples),
* **15% for testing** (670 samples).



***Figure 4.2: MRI Stratified Data split Chart***

This splitting was performed using stratified sampling, meaning the relative distribution of the 44 classes was preserved across all three subsets. This is especially important in medical datasets where class imbalance can severely affect learning and evaluation.

#### **4.2.2 Image Resizing and Formatting**

The MRI images varied in resolution, so all images were resized to a standard input shape of **(224, 224, 3)**. This ensures compatibility with most pretrained CNN architectures, which expect 3-channel input. Grayscale images were implicitly converted by replicating the single channel into three.

#### **4.2.3 Dataset Pipeline Construction**

To streamline training and evaluation, **a TensorFlow Dataset pipeline** was implemented using a helper function prepare\_dataset() which:

* Takes as input the images and their labels (X\_train, y\_train)
* Applies model-specificpreprocessing via a function like (tf.keras.applications.efficientnet\_v2.preprocess\_input)
* Includes options for shuffling, data augmentation, batching, and prefetching

This function returns train\_ds, val\_ds, and test\_ds objects , then optimized datasets of TensorFlow tf.data.Dataset pipelines will be ready for model training and evaluation.

#### **4.2.4** **Data Augmentation Strategy**

To combat overfitting and enrich the training data, a data augmentation layer was defined using tf.keras.Sequential. This augmentation is only applied to the training data and includes:

* RandomFlip("horizontal"): flips images horizontally
* RandomRotation(0.1): randomly rotates the image by ±10%
* RandomZoom(0.1): zooms in/out up to 10%
* RandomContrast(0.1): adjusts image contrast by ±10%

These augmentations introduce variation and improve the model’s generalization to unseen MRI data, which is especially important given the limited dataset size and class imbalance.

**4.3 Experimental Design and Model Selection**

The initial step involved a strategic selection of pre-trained models. The architecture, original training data (ImageNet), and complexity of each model were analyzed to assess its suitability for the brain tumor dataset. Given the dataset's relatively small size (~4,478 samples), moderately-sized architectures like EfficientNetV2B0 and MobileNetV3Small were prioritized over larger, more complex models to mitigate the inherent risk of overfitting from the outset.

This was followed by a systematic, multi-phase methodology designed to identify the optimal configuration for each selected model.

**4.4 Training and Fine-Tuning Strategy**

The core of the methodology was an iterative training and optimization process, divided into distinct phases for each model.

**Phase 1: Baseline Training**  
Each model first underwent baseline training with its entire pre-trained backbone frozen. A simple classification head, typically consisting of a Global Average Pooling layer followed by dropout and a final dense layer (Figure 4.3), was attached and trained. This initial step established a performance baseline and provided critical insights into the model's out-of-the-box suitability for the task.



***Figure 4.3: Basic Classification Head Architecture***

**Phase 2: Iterative Refinement and Fine-Tuning**  
Based on the baseline performance, a series of iterative experiments were conducted, guided by the following principles:

* **Addressing Overfitting:** If a model exhibited significant overfitting (a large gap between training and validation accuracy), the strategy shifted to targeted fine-tuning. This involved incrementally unfreezing the top layers of the backbone while simultaneously designing a more robust and heavily regularized classification head , like the one in Figure 4.4. This included adding more dense layers, increasing L2 regularization strength, and tuning dropout rates. If the training is stable and strong but there is still a gap between the validation accuracy and the training accuracy , for example, a 5% - 20% percent difference in favor of the training accuracy , it is sufficient to focus on a deep fine-tuning of the classification head network with more dropout layers and bigger L2 regularization penalties in addition to refreezing a very few layers, for example, about 5-30 layers ( if needed) , to address the gap as much as possible and get closer to the ideal performance.
* **Addressing Slow Convergence:** Conversely, if a model learned too slowly or stagnated, two approaches were tested: increasing the learning rate to accelerate convergence or unfreezing additional layers to increase the model's learning capacity. A standard learning rate of 2e-3 was used for fine-tuning to avoid damaging pre-trained weights, but it was adjusted when necessary.
* **Systematic Layer Unfreezing:** The process of unfreezing layers was methodical. Small blocks of layers were unfrozen incrementally, and performance was evaluated after each change. If unfreezing a large portion of the network led to severe overfitting, the process was reversed by gradually re-freezing layers to identify the optimal trade-off point between model capacity and generalization.

Training was conducted for up to 100 epochs, with early stopping configured using a patience of 10 epochs on the validation loss, min delta of 0.005, restoring the best weights upon completion.



***Figure 4.4: Denser & Regularized Head Architecture***

**4.5 Evaluation**

The performance evaluation for each experiment primarily relies on the test set accuracy & test loss as the main criterion.  
If an experiment ranks among the top three in performance, a more detailed evaluation is conducted, focusing on the macro averaged F1-score and micro averaged F1-score across all categories, including assessments at the individual category level. Samples with the lowest F1-scores are carefully analyzed to determine whether issues such as data sparsity or low sample counts are contributing to poor category performance.

Experiments exhibiting a significant discrepancy such as high validation loss despite low training loss, or low validation accuracy despite high training accuracy are allowed a maximum of 20 epochs of early stopping. If the gap persists without noticeable improvement or convergence, training is halted early. Subsequently, further adjustments and fine-tuning are applied, targeting the pre-trained model layers, and in case of overfitting there will be an extensive fine-tuning of the classification head layers, for example adding more dense layers , l2 regularizations, more dropout layers with different tuned values , alongside optimization of other hyperparameters, then a new experiment is conducted.

**5. Results**

**Table 5.1 (DenseNet, Inception, Xception, ResNet)**

**Consolidated Summary of Models Experimentation**

The following experiments were conducted to evaluate the performance of larger, more complex CNN architectures.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model Architecture | Experiment Description | Configuration Details | Training Accuracy | Validation Accuracy | Validation Loss | Outcome / Observation |
| **ResNet50V2** | **Fine-Tuned (Best Result)** | Unfrozen top 75 layers, Deeper Head, LR=0.001 | 97.48% | **27.31%** | 3.362 | Best validation accuracy achieved, but **catastrophic overfitting**. |
|  | Frozen Base + Deeper Head | Frozen Base, D(512)->D(256)->D(44), LR=0.002 | 58.27% | 23.88% | 5.394 | Severe overfitting. The model memorized the training set. |
|  | Base Model (Simple Head) | Frozen Base, Simple Head, LR=0.01 | ~60% | ~20% | ~6.5 | Failed to converge. Unstable training with high validation loss. |
| **Xception** | **Fine-Tuned (Best Result)** | Unfrozen top 12 layers, Deeper Head | 62.93% | **24.63%** | 3.432 | The best result for Xception, but still extremely poor and overfit. |
|  | Frozen Base + Deeper Head + Kernel Init | Frozen Base, D(512)->D(256)->D(44), Glorot Init | 80.10% | 21.64% | 5.512 | Severe overfitting. Kernel initialization did not solve the generalization issue. |
|  | Frozen Base + Deeper Head | Frozen Base, D(512)->D(256)->D(44) | 42.70% | 18.51% | 3.283 | Failed to learn meaningful features. |
|  | Base Model (Simple Head) | Frozen Base, Simple Head, LR=0.01 | 52.65% | 17.01% | 7.135 | Failed to converge, very high validation loss. |
| **InceptionV3** | Frozen Base + Deeper Head + Kernel Init | Frozen Base, D(512)->D(256)->D(44), Glorot Init | 68.95% | **21.79%** | 5.100 | Severe overfitting. |
|  | Fine-Tuned (Top 61 Layers) | Unfrozen top 61 layers, Deeper Head | 61.52% | 18.66% | 4.808 | Fine-tuning did not improve performance and resulted in overfitting. |
|  | Frozen Base + Deeper Head | Frozen Base, D(512)->D(256)->D(44) | 40.51% | 15.07% | 3.679 | Failed to learn. |
|  | Fine-Tuned (Top 11 Layers) | Unfrozen top 11 layers, Deeper Head | 43.68% | 11.49% | 4.556 | Minimal fine-tuning was ineffective. |
| **DenseNet121** | Frozen Base + Deeper Head + Kernel Init | Frozen Base, D(512)->D(256)->D(44), Glorot Init | 78.66% | **19.40%** | 8.380 | Severe overfitting with extremely high validation loss. |
|  | Fine-Tuned (Top 12 Layers) | Unfrozen top 12 layers, Deeper Head | 54.82% | 18.36% | 3.850 | Minimal fine-tuning was ineffective. |
|  | Base Model (Simple Head) | Frozen Base, Simple Head, LR=0.01 | 40.49% | 17.91% | 30.608 | Failed. The model diverged completely, as shown by the massive loss. |
|  | Frozen Base + Deeper Head | Frozen Base, D(512)->D(256)->D(44), LR=0.002 | 48.99% | 15.37% | 5.953 | Failed to learn. |

*Where D(x) denotes a Dense layer with x neurons and GAP is Global Average Pooling.*

**Table 5.2 (MobileNetV3Small)**

**Summary of MobileNetV3-Small Experiments**

The following experiments were conducted to evaluate the effectiveness of transfer learning and fine-tuning strategies using the lightweight MobileNetV3Small architecture pre-trained on ImageNet. The results are presented in descending order of test accuracy.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Rank | Test Accuracy | Training Accuracy | Val Accuracy | Fine-Tuning Strategy | Classifier Head Architecture | Key Observation / Learning Rate (LR) |
| 1 | 89.70% | 97.81% | 90.45% | Unfrozen top 30 layers (Base layer 127 onwards) | GAP → Drop → D(44) (Simple) | Best performing model. Less aggressive fine-tuning worked best.  Slight Overfitting |
| 2 | 88.06% | 98.84% | 88.36% | Unfrozen top 57 layers | GAP → Drop → D(44) (Simple) | Good performance, but slightly less than unfreezing 30 layers. |
| 3 | 79.40% | 96.64% | 58.51% | Unfrozen top 57 layers | Deeper Head (D(512)→...) | Severe Overfitting. A deeper head caused performance to collapse. |
| 4 | 55.97% | 56.88% | 54.63% | Frozen (No Fine-Tuning) | GAP → Drop → D(44) (Simple) | LR = 0.002. Poor baseline performance. |
| 5 | N/A | 72.26% | 73.58% | Frozen (No Fine-Tuning) | GAP → Drop → D(44) (Simple) | LR = 0.1. Best performing frozen model. |
| 6 | N/A | 60.62% | 67.46% | Frozen (No Fine-Tuning) | Drop(0.5) → D(512, l2=0.005) → D(256, l2=0.0025) → Drop(0.1) → D(44, l2=0.0005) | A complex head on a frozen model is not effective (this is the "Final Model 1" architecture from EffNetV2B0, applied here to MobileNetV3Small). |
| 7 | N/A | 60.53% | 62.39% | Frozen (No Fine-Tuning) | GAP → Drop → D(44) (Simple) | LR = 0.001. |
| 8 | N/A | 58.90% | 57.46% | Frozen (No Fine-Tuning) | GAP → Drop → D(44) (Simple) | LR = 0.003. |
| 9 | N/A | Highly variable (19.80%-70.78%) | < 50% | Various (30-100 layers) | Deeper Regularized Heads (D(512)→...) | Failed Experiments. All attempts to use a deeper head with fine-tuning resulted in catastrophic overfitting. |

*Where D(x) denotes a Dense layer with x neurons and GAP is Global Average Pooling.*

*N/A: Test set evaluation was not performed as the model's validation performance was not competitive enough to be considered a final candidate.*

**Table 5.3 (EfficientNetV2B0)**

**Summary of EfficientNetV2B0 Fine-Tuning Experiments**

**Learning Rate = 0.002, Early Stopping with patience of 10, min\_delta = 0.005**

The following experiments were conducted to evaluate the impact of unfreezing layers in the EfficientNetV2-B0 architecture (pre-trained on ImageNet) and modifying the classifier head design. Results are reported in descending order of test accuracy.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Rank | Test Accuracy | Training Accuracy | Val Accuracy | Fine-Tuning Strategy | Classifier Head Architecture | Experiment Name |
| 1 | 93.88% | 96.46% | 94.03% | Unfrozen top 170 layers | GAP → Drop → D(512) → Drop → D(256) → D(44) | Deeper Head & Maximum Fine-Tuning |
| 2 | 90.45% | 98.85% | 92.24% | Unfrozen top 170 layers | GAP → Drop → D(44) | Simple Head & Maximum Fine-Tuning |
| 3 | 90.00% | 96.34% | 90.75% | Unfrozen top 170 layers | (Assumed Deeper Head - Early run) | Deeper Head (Early Run) |
| 4 | 87.16% | 97.28% | 89.40% | Unfrozen top 70 layers | GAP → Drop → D(44) | Simple Head & Partial Fine-Tuning |
| 5 | N/A | 58.08% | 56.72% | Frozen (0 layers unfrozen)  LR = 0.002 | GAP → Drop → D(44) | Base Model (No Fine-Tuning) |

*Where D(x) denotes a Dense layer with x neurons and GAP is Global Average Pooling.*

*N/A: Test set evaluation was not performed as the model's validation performance was not competitive enough to be considered a final candidate.*

**Table 5.4 (Advanced Fine-Tuning of EfficientNetV2B0)**

**Summary of Advanced Fine-Tuning Experiments on EfficientNetV2-B0**

**Learning Rate = 0.002 , Early Stopping with patience of 10 , min\_delta = 0.005**

The following experiments were conducted to evaluate the impact of advanced fine-tuning strategies applied to the EfficientNetV2-B0 architecture pre-trained on ImageNet. Unless otherwise specified as Frozen, all models were fine-tuned starting from layer 100 of the base model.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Rank | Test Accuracy | Training Accuracy | Val Accuracy | Classifier Head Architecture & Regularization | Experiment Name |
| 1 | 94.48% | 97.10% | 95.46% | Drop(0.5) → D(512, l2=0.005) → D(256, l2=0.0025) → Drop(0.1) → D(44, l2=0.0005) | Final Model 1 (Best) |
| 2 | 93.58% | 95.71% | 92.69% | Drop(0.5) → D(512) → Drop(0.3) → D(256) → Drop(0.1) → D(128) → Drop(0.05) → D(64) → D(44) with mixed L2 (0.001–0.005) | Mixed Regularization 1 |
| 3 | 93.43% | 95.96% | 93.58% | Drop(0.6) → D(768, l2=0.008) → D(256, l2=0.004) → Drop(0.2) → D(44, l2=0.002) | Final Model 2 |
| 4 | 92.69% | 97.10% | 92.54% | Drop(0.5) → D(512, l2=0.1) → Drop(0.3) → D(256, l2=0.05) → Drop(0.1) → D(128, l2=0.001) → D(44, l2=0.0005) | Heavy L2 Regularization |
| 5 | 92.39% | 96.63% | 94.48% | Drop(0.5) → D(768) → Drop(0.3) → D(512) → Drop(0.1) → D(256) → Drop(0.05) → D(128) → D(44) | Wider Head |
| 6 | 91.79% | 95.73% | 92.99% | Drop(0.5) → D(768, l2=0.006) → D(348, l2=0.003) → D(192, l2=0.001) → Drop(0.25) → D(44, l2=0.0006) | Final Model 4 (Deeper) |
| 7 | 91.49% | 94.39% | 90.90% | Architecture not specified (variation with less regularization, frozen base model) | Less Regularized |
| 8 | 91.34% | 95.58% | 91.94% | Drop(0.6) → D(512, l2=0.006) → Drop(0.3) → D(256, l2=0.003) → Drop(0.15) → D(44, l2=0.0015) | Final Model 3 |
| 9 | 89.25% | 93.46% | 90.60% | D(512) → Dropout → D(256) → D(128) → Dropout → D(44) (Initial dropout layers) | Initial Model |
| 10 | 88.36% | N/A\* | 90.75% | Same as "Mixed Regularization 1" but with a frozen base model | Mixed Reg. (Frozen) |
| 11 | 86.27% | 89.81% | 85.07% | D(512) → ... → D(44) with a uniform l2=0.005 on all dense layers | Uniform L2 (0.005) |
| 12 | N/A | 67.51% | 68.21% | Same architecture as "Final Model 1" but with a fully Frozen base model and LR = 0.002 | Base Model (Frozen) |
| 13 | 92.99% | 97.77% | 93.43% | Same architecture as "Final Model 1" but with a fully Frozen base model and LR = 0.02 | Base Model (Frozen) |

*Where D(x) denotes a Dense layer with x neurons and GAP is Global Average Pooling.*

*N/A: Test set evaluation was not performed as the model's validation performance was not competitive enough to be considered a final candidate.*

**6. Discussion**

**6.1 (DenseNet, Inception, Xception, ResNet) Experimentation**

Across the board, these models demonstrated a significant inability to generalize from the training data, resulting in severe overfitting and poor performance on the validation set. These experiments conclusively show that the **DenseNet121, InceptionV3, Xception, and ResNet50V2** architectures are not suitable for this specific dataset and task under the tested conditions.

* **Universal Failure**: No experiment yielded a validation accuracy above 28%, which is far below acceptable thresholds for practical deployment.
* **Overfitting as the Core Problem**: The dominant failure mode was persistent overfitting, where the models achieved high training accuracy but consistently failed to generalize, as reflected by low validation accuracy and high validation loss.
* **Ineffective Mitigation Attempts**: Standard techniques such as deeper classifier heads, dropout, L2 regularization, and custom kernel initializations were extensively applied. In addition to that, the classification heads were gradually made more complex and heavily regularized. These efforts included fine-tuning dropout rates, increasing regularization strength, and carefully adjusting activation layers.

Further, comprehensive experiments were conducted using fully frozen backbones (except for the classification head), as well as partially unfrozen models unfreezing half, a quarter, or specific portions of the network (layer-wise). Despite these efforts, which were combined with advanced regularization strategies and progressive unfreezing, none of the models showed meaningful improvement. In fact, some configurations resulted in complete training stagnation, with symptoms of gradient vanishing and no convergence.

These consistent failures strongly suggest that the limitation lies not in tuning or training strategies, but in the architecture's capacity relative to the dataset. Heavier architectures with high representational power tend to overfit small or unbalanced datasets. Therefore, lightweight and modern architectures like EfficientNetV2 and MobileNetV3 demonstrate better suitability for this task, offering more stable training behavior and better generalization. Alternatively, expanding and balancing the dataset may also help enable more complex architectures to learn robust patterns.

**6.2 MobileNetV3Small Experimentation**

Experiments with MobileNetV3Small for brain tumor classification revealed that optimal performance is achieved through a delicate balance of fine-tuning depth and classifier head simplicity. While a frozen backbone proved inadequate for feature extraction, yielding low accuracies (e.g., max 73.58% validation), fine-tuning significantly improved results.

Crucially, allowing only the top 30 layers of the backbone to adapt achieved the highest test accuracy of 89.70% (with 97.81% training and 90.45% validation accuracy), outperforming deeper fine-tuning (top 57 layers at 88.06%) due to reduced overfitting. A critical insight was the model's extreme sensitivity to classifier head complexity: while a simple GlobalAveragePooling → Dropout → Dense(44) head was highly effective, any attempt at deeper or more intricate heads consistently led to catastrophic overfitting, evidenced by high training accuracies (e.g., 96.64%) paired with severely degraded validation and test performance (e.g., 58.51% validation, 79.40% test) and widespread failure to generalize across numerous configurations.

In essence, MobileNetV3Small's efficient architecture benefits from focused fine-tuning , streamlined output layer and higher number of epochs with higher learning rates, strongly emphasizing that for this model and dataset, simplicity in design is key to mitigating overfitting and ensuring robust generalization.

**6.3 EfficentNetV2B0 Experimentations**

Experiments with EfficientNetV2-B0 were conducted to determine the most effective strategy for brain tumor classification. The investigation began by comparing a frozen backbone against various fine-tuning depths.

Initial tests with a frozen backbone using a standard learning rate (0.002) were unpromising, yielding a validation accuracy of only 68.21% (for a complex head) and 56.72% (for a simple head). This preliminary result suggested that the base ImageNet features were insufficient and that fine-tuning was necessary.

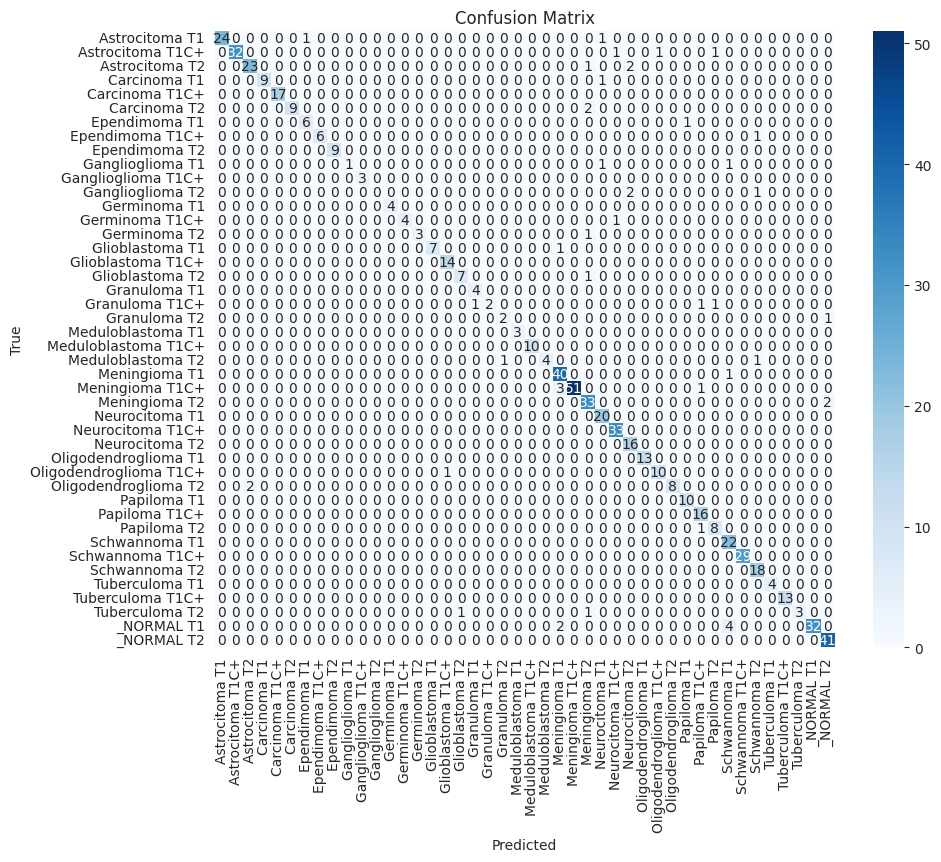
Consequently, fine-tuning was explored. Unfreezing just the top 70 layers boosted test accuracy to 87.16%. Deeper fine-tuning (unfreezing the top 170 layers) proved even more effective. With a simple head, this strategy achieved 90.45%, but performance peaked at 93.88% when paired with a more complex, deeper classification head, establishing deep fine-tuning as a highly successful approach.

However, a pivotal later experiment revisited the frozen backbone strategy, this time with a significantly higher learning rate of 0.02. This single change dramatically altered the outcome: the frozen model achieved an excellent 92.99% test accuracy. This crucial finding demonstrated that the pre-trained features are indeed powerful, but require the classification head to be trained aggressively to adapt to them.

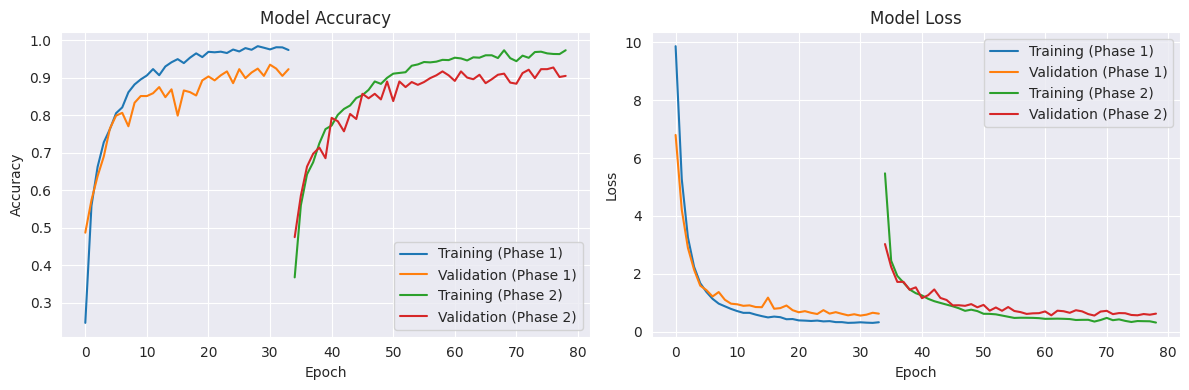
Further advanced experiments focused on optimizing the fine-tuned model's head architecture. The best overall performance (94.48%) was achieved by "Final Model 1," which employed deep fine-tuning combined with a carefully designed head using progressive L2 regularization.

In conclusion, the experiments revealed two distinct and effective strategies for leveraging EfficientNetV2-B0:

1. Maximum Accuracy via Fine-Tuning: The highest performance is achieved through deep fine-tuning of the backbone, paired with a carefully designed, regularized classification head and a moderate learning rate.
2. High Efficiency via a Frozen Backbone: Excellent, near-top-tier results can be achieved far more efficiently by keeping the backbone frozen and training only the classification head with a significantly higher learning rate.



***Figure 6.1: Test Set Confusion Matrix of Best EfficientNetV2B0 Fine-Tuned Experiment***



***Figure 6.2: Learning Curves Between Best Frozen Backbone Experiment vs Best Fine-Tuned Experiment (EfficinetNetV2B0)***

**7. Conclusion**

1. **Big Models (ResNet, Xception, Inception, DenseNet) Failed:**
   * These very complex models, even with fine-tuning, consistently overfitted badly.
   * They tried to memorize the training data instead of learning real patterns, so they performed terribly on new images.
   * They likely need a huge amount of perfectly balanced data, which wasn't available, to work well.
2. **MobileNetV3Small was Better, but Not Best:**
   * This smaller model was faster to train and more efficient.
   * It performed much better than the "big failed models."
   * It worked best when only a small part of its original structure was fine-tuned, and its classification head was kept very simple.
   * However, it still struggled with some overfitting and couldn't reach the highest accuracy levels.
3. **EfficientNetV2-B0 was the Champion:**
   * This model was the strongest performer and the best choice for this project, demonstrating stable and reliable training.
   * Its success stemmed from its architectural flexibility, which revealed two distinct and effective strategies:
     + **Maximum Accuracy via Fine-Tuning:** The absolute highest performance (**94.48%**) was achieved through deep fine-tuning. This required a **moderate learning rate** to carefully adapt the unfrozen layers without corrupting their valuable pre-trained weights.
     + **High Efficiency via a Frozen Backbone:** Crucially, a highly efficient alternative was discovered. By keeping the backbone completely frozen, a **high learning rate (0.02)** was essential for the new classification head to converge properly and learn from the fixed features, achieving an excellent **92.99%** accuracy.
   * This demonstrates that while fine-tuning can extract the best possible performance, the model's pre-trained features are powerful enough to yield outstanding results on their own if the head is trained appropriately. The learning rate proved to be the key to this distinction: a high rate is essential to unlock the potential of a frozen backbone, while a moderate rate is required to safely adapt the delicate weights during fine-tuning.

**8. Future Work**

* Explore Full EfficientNet Family: Evaluate larger EfficientNet models (B1-B7) with refined fine-tuning for potentially higher accuracy.
* Optimize MobileNet Further: Apply advanced fine-tuning to MobileNetV3 and other versions to maximize performance while retaining efficiency.
* Investigate Vision Transformers (ViTs): Explore Vision Transformer architectures for their demonstrated strength in MRI medical image analysis.
* Granular Fine-Tuning/Layer Modification: Precisely modify or unfreeze specific internal layers in models prone to overfitting for finer control ( like ResNet50V2 , DenseNet102 )

**9. Appendices**

* Source Code
* PPT Presentation

**By :**

* **Ahmad Hudhud - https://github.com/AhmadHudhud83**