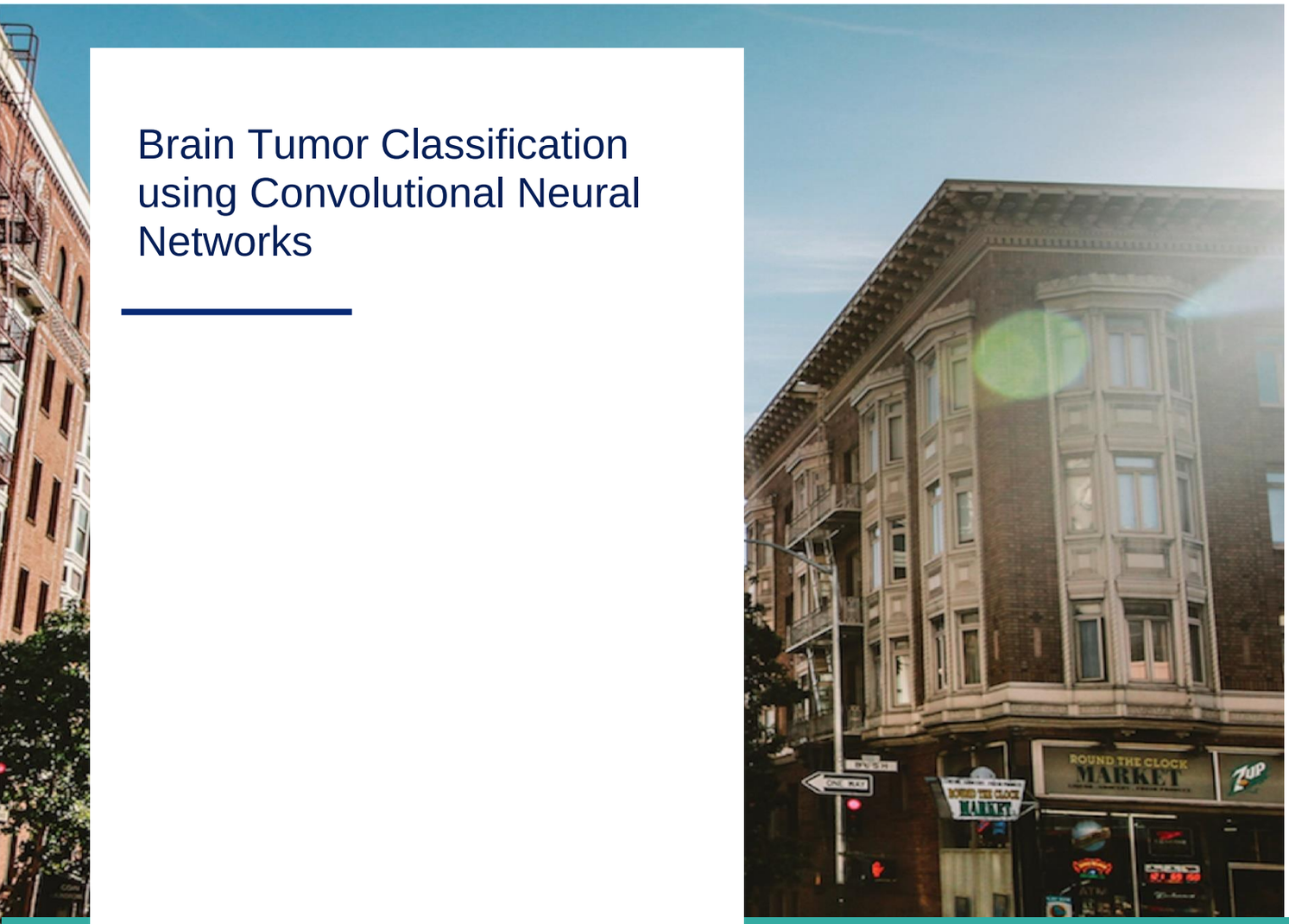
# Brain Tumor Classification using Convolutional Neural Networks




**PSN: 40022593**

LTTS

Authored by: **Adhokshaj Wategaonkar**

## *Problem Statement*

### Brain Tumor Classification using Convolutional Neural Networks (CNN)

Using the dataset of various MRI's classify the type of tumor patient has. Analyze and predict if patient is normal. Explain and elaborate on the data processing and create a function to take various classes of images as inputs and display in output the kind of tumor patient has. Experiment on various layers and hyperparameters of the model to get optimum accuracy.

The method selected is Convolutional Neural Network. Experiment with new methods if required.

## *Phase 2:*

Experiment with EfficientNet versions.

Create a UI for the models and experiments.

# *Data Set*

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

## *What is a brain tumor?*

A brain tumor is a collection, or mass, of abnormal cells in your brain. Your skull, which encloses your brain, is very rigid. Any growth inside such a restricted space can cause problems. Brain tumors can be cancerous (malignant) or noncancerous (benign). When benign or malignant tumors grow, they can cause the pressure inside your skull to increase. This can cause brain damage, and it can be life-threatening.

## *The importance of the subject.*

Early detection and classification of brain tumors is an important research domain in the field of medical imaging and accordingly helps in selecting the most convenient treatment method to save patients life therefore

## *Methods*

The application of deep learning approaches in context to improve health diagnosis is providing impactful solutions. According to the World Health Organization (WHO), proper brain tumor diagnosis involves detection, brain tumor location identification, and classification of the tumor based on malignancy, grade, and type. This experimental work in the diagnosis of brain tumors using Magnetic Resonance Imaging (MRI) involves detecting the tumor, classifying the tumor in terms of grade, type, and identification of tumor location. This method has experimented in terms of utilizing one model for classifying brain MRI on different classification tasks rather than an individual model for each classification task. The Convolutional Neural Network (CNN) based multi-task classification is equipped for the classification and detection of tumors. The identification of brain tumor location is also done using a CNN-based model by segmenting the brain tumor.

**This dataset is a combination of the following three datasets :**

**figshare**
**SARTAJ dataset**
**Br35H**

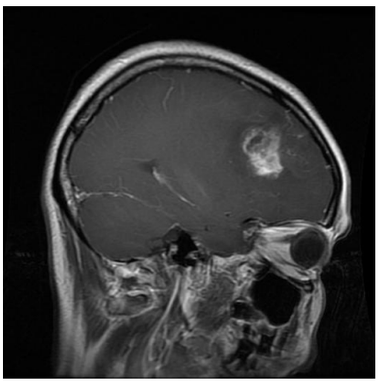
Due to this reason we couldn't add more data as this is a superset of the primary datasets found on the internet consisting of brain MRI images.

## *The detailed description of data is:*

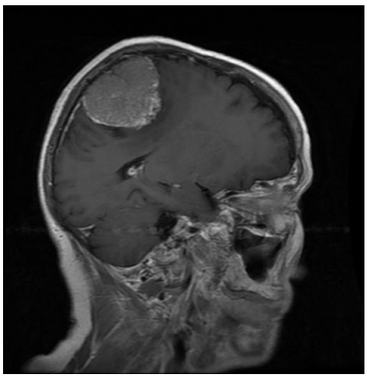
This dataset contains **7022** images of human brain MRI images which are classified into 4 classes: ***glioma*** - ***meningioma*** - ***no tumor*** and ***pituitary***.

**Training :** *glioma* ***(1321)*** – *meningioma* ***(1339) -*** *no tumor* ***(1595)*** and *pituitary* ***(1457)***.
**Testing :** *glioma* ***(300)*** – *meningioma* ***(306) -*** *no tumor* ***(405)*** and *pituitary* ***(300)***.

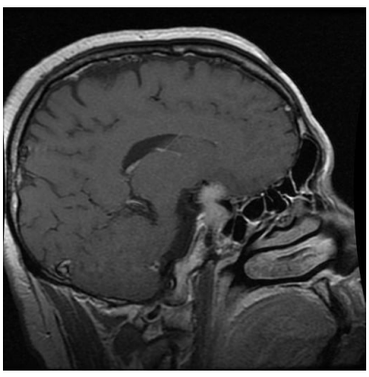
Here are some images as examples:



***Glioma*** Tumor



***Meningioma*** Tumor



***Pituitary Adenoma*** Tumor

# *Model and Algorithms*

The problem statement and dataset are the parameters on which we decide what algorithms to use or the method of solving the problem.

Thus, after analyzing the data, we have reached our conclusions and assessments of the dataset.

Clearly here, this is a simple image classification problem. We have different classes of images which we want to categorize and identify any images who fall under a specific category. Our dataset is already divided in training and testing sets. This solves a lot of preprocessing for us.

The best and time proven Neural network for solving image classification problems is **Convolutional Neural Networks** or CNN.

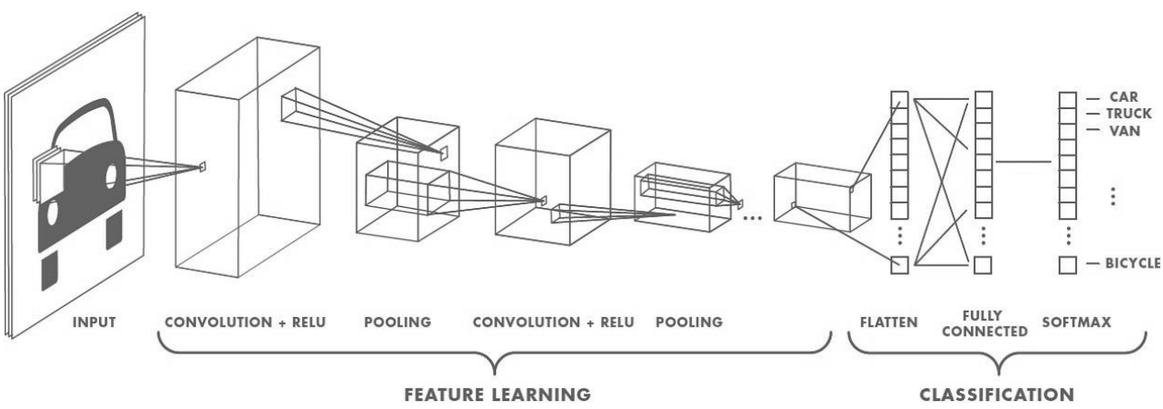
## *Convolutional Neural Network*

Convolutional Neural Networks (CNNs) are designed to map image data (or 2D multi-dimensional data) to an output variable (1 dimensional data). They have proven so effective that they are the ready to use method for any type of prediction problem involving image data as an input.

The benefit of using CNNs is their ability to develop an internal representation of a two-dimensional image. This allows the model to learn position and scale in variant structures in the data which is important when working with images.
One can effectively use Convolutional Neural Network For:

- Image data
- Classification prediction problems
- Regression prediction problems

www.poloclub.github.io/cnn-explainer/
CNN can be represented using a figure as such:



There are three types of layers in a convolutional neural network: **convolutional layer**, **pooling layer,** and **fully connected layer**. Each of these layers has different parameters that can be optimized and performs a different tack on the input data.
Different combinations of these layers are used to get effective sequential model.

- Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.

- Pooling layers are like convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.

- Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is like the output layer of an MLP.

Max pooling layer is used to reduce the dimensionality of the image. Thus, when we apply a conv2D layer after this dimensionality reduction we get a more detailed and optimized performing model. A model with a greater number of layers though increasing complexity also increases precision.

To visualize this occurrence, we can analyze it in this manner.


Here, we can see that the features are optimized after every Conv2D and MaxPool layer. The image though getting blurrier it's stand-out features are enhanced. Thus,

intuitively we can understand that how the combinations of Conv2D layer and MaxPool layer works.

We have added a flatten layer to convert all the matrices into a 1D array. Flatten layer converts the data from Conv2D and MaxPool layers in the format Dense Layers can process.

We add two hidden fully connected layers (Dense) after convolutional layers and before the output layer. The reason is that convolutional layers try to extract features in a differentiable manner, and fully connected layers try to classify the features.

Consequently, adding more layers to the dense section can empower your network's ability to classify the extracted features better. Thus, multiple dense layers.

## *Model (Phase 2):*

In phase 2 we'll specifically focus on EfficientNet for the modelling part.

After understanding how the Convolutional Layers are working with each other we tested a new concept called ***Model Scaling.***

Model scaling is the process of scaling up a base convolutional neural network to endow it with greater computational complexity and consequently more representational power. Model scaling approaches typically focus on maximizing accuracy. The conventional practice for model scaling is to arbitrarily increase the CNN depth or width, or to use larger input image resolution for training and evaluation. While these methods do improve accuracy, they usually require tedious manual tuning, and still often yield suboptimal performance. It's typically performed with pretrained models and importing them.

***EfficientNet*** is a convolutional neural network architecture and scaling method that uniformly scales all dimensions of depth/width/resolution using a compound coefficient. The compound scaling method is justified by the intuition that if the input image is bigger, then the network needs more layers to increase the receptive field and more channels to capture more fine-grained patterns on the bigger image. Generally, the models are made too wide, deep, or with a very high resolution. Increasing characteristics helps the model initially but it quickly saturates, and the model made just has more parameters and is therefore not efficient. In EfficientNet they are scaled in a more principled way i.e. gradually everything is increased.

While scaling individual dimensions improves model performance, it was observed that balancing all dimensions of the network—width, depth, and image resolution—against the available resources would best improve overall performance.

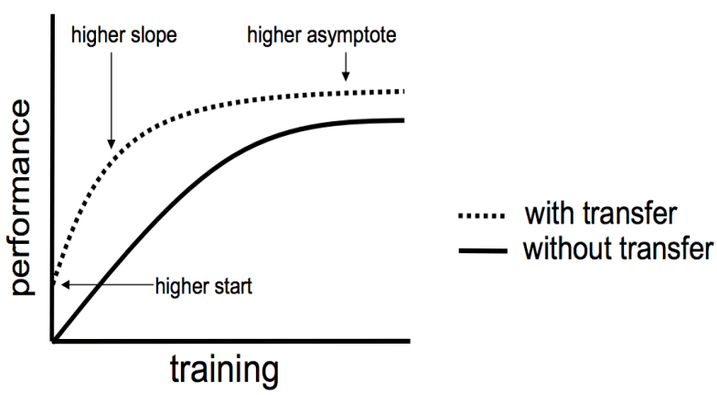
# *Transfer Learning*

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.

***Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned.***

In our case we're using transfer learning learned from models trained on the ***ImageNet*** dataset.




1. **Higher start**: The initial skill (before refining the model) on the source model is higher than it otherwise would be.

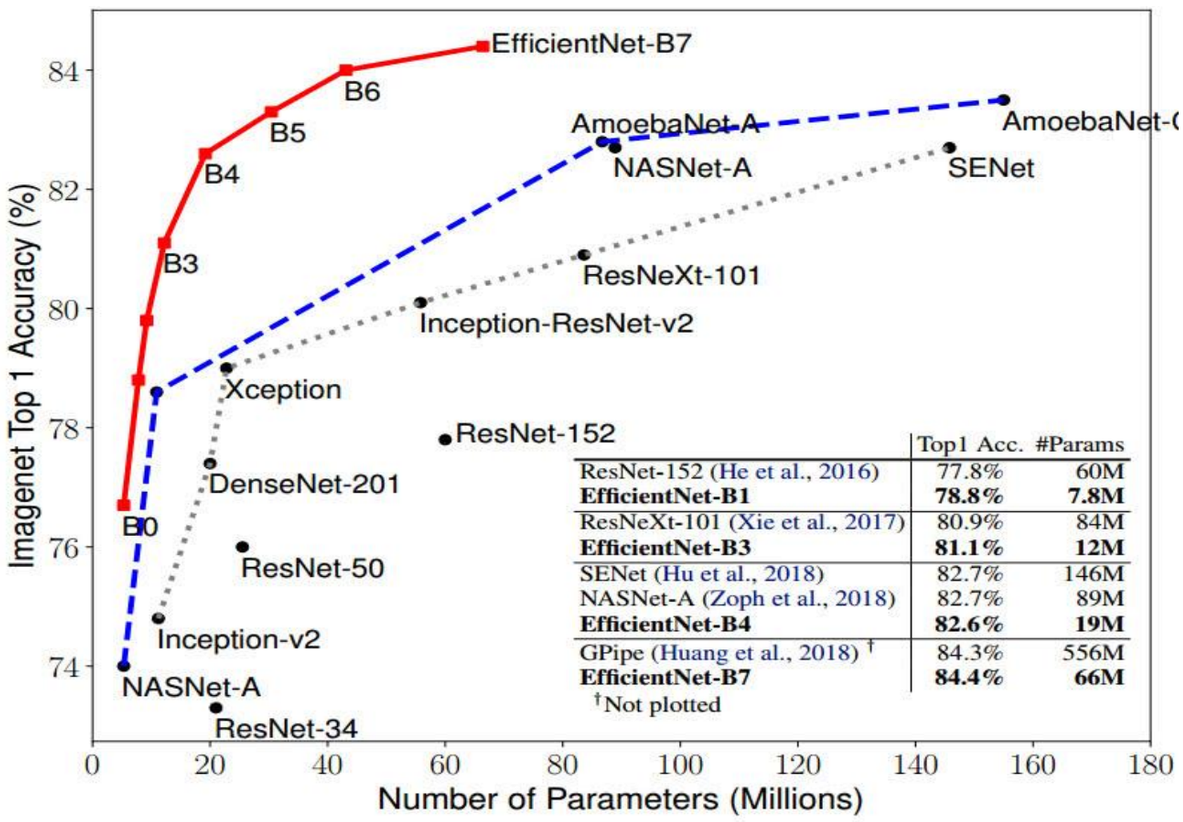
2. **Higher slope**: The rate of improvement of skill during training of the source model is steeper than it otherwise would be.

3. **Higher asymptote**: The converged skill of the trained model is better than it otherwise would be.

## *EfficientNet:*

EfficientNet is based on the baseline network developed by the neural architecture search using the AutoML MNAS framework. The network is fine-tuned for obtaining maximum accuracy but is also penalized if the network is very computationally heavy. It is also penalized for slow inference time when the network takes a lot of time to make predictions. The architecture uses a mobile inverted bottleneck convolution like MobileNet V2 but is much larger due to the increase in FLOPS. This baseline model is scaled up to obtain the family of EfficientNets.

EfficientNet family has 8 distinct networks with distinct number of layers.



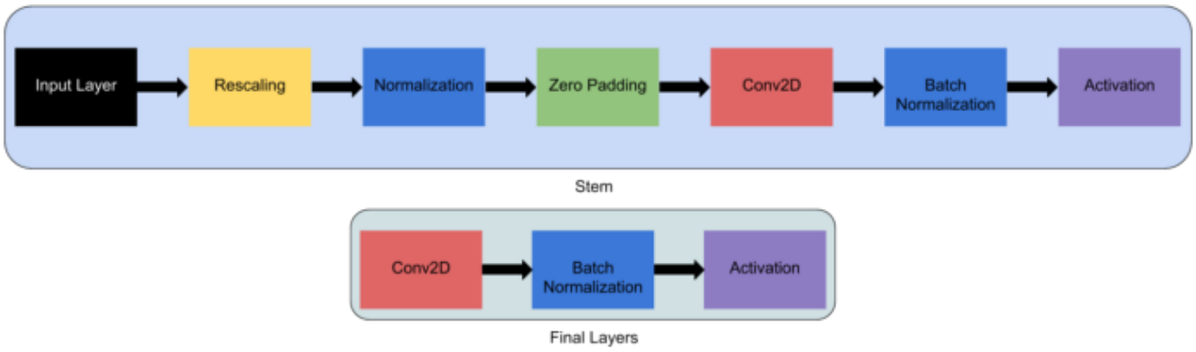

| | Top1 Acc. | #Params |
|---|---|---|
| ResNet-152 (He et al., 2016) | 77.8% | 60M |
| **EfficientNet-B1** | **78.8%** | **7.8M** |
| ResNeXt-101 (Xie et al., 2017) | 80.9% | 84M |
| **EfficientNet-B3** | **81.1%** | **12M** |
| SENet (Hu et al., 2018) | 82.7% | 146M |
| NASNet-A (Zoph et al., 2018) | 82.7% | 89M |
| **EfficientNet-B4** | **82.6%** | **19M** |
| GPipe (Huang et al., 2018) † | 84.3% | 556M |
| **EfficientNet-B7** | **84.4%** | **66M** |

†Not plotted

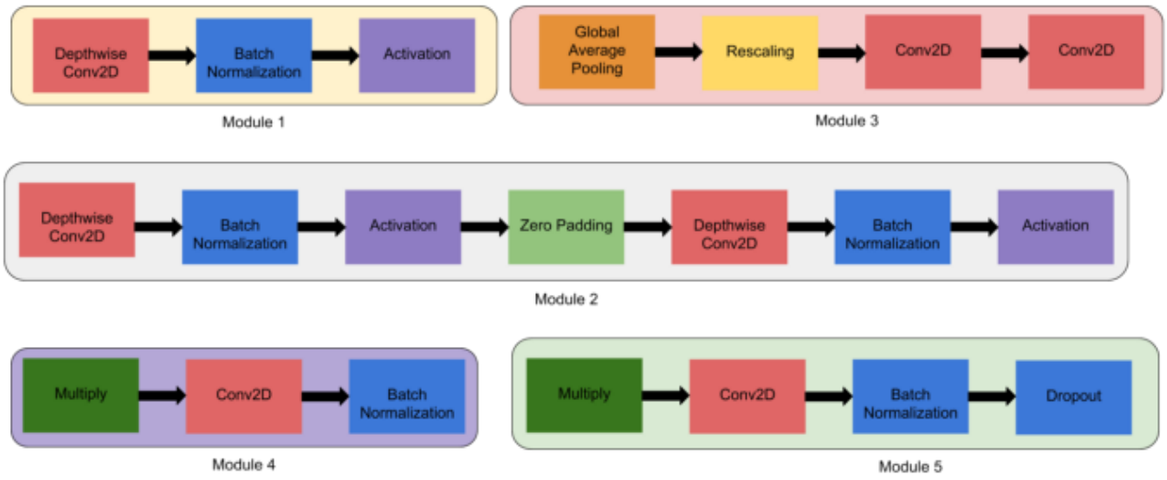
This is a comparison between models trained on ImageNet dataset and we can easily assess that EfficientNet is a superior model.
It outperforms all previous CNN architectures on most benchmarking datasets. The compound scaling method can also be used to efficiently scale other CNN architectures as well. It allows EfficientNet models to be scaled in such a way that it achieves state-of-the-art accuracy with an order of magnitude fewer parameters and FLOPS, on ImageNet and other commonly used transfer learning datasets.

EfficientNet returns a Keras image classification model, optionally loaded with weights pre-trained on ImageNet.

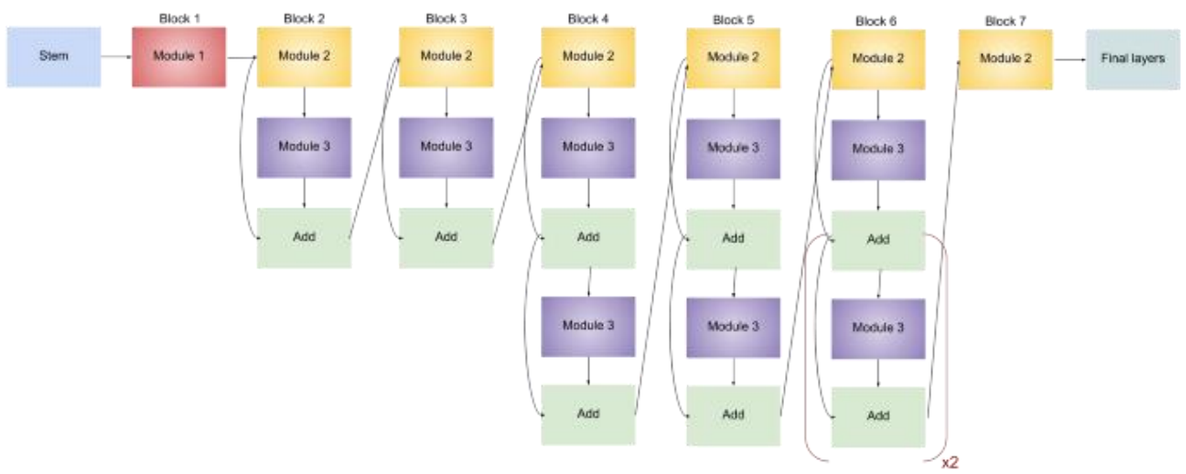
The first thing is any network is its stem after which all the experimenting with the architecture starts. The basic stem architecture is supposed to look like this:




In EfficientNet the same concept is used.
If you count the total number of layers in EfficientNet-B0 the total is 237. all these layers can be made from 5 modules shown below and the stem above.




EfficientNet-B0 is also build on the same foundation. If visualized EfficientNet-B0 looks like this:



# *Experiments and Implementations*

## *Experiments*

- Our experiments will be mainly based on testing the various Efficient Networks in the family. Also we will try to understand the accuracy and loss graphs.

- We could've played around with activation functions, Kernels & Units but a little research showed us that the most optimum and widely used Activation Function for Images Training is ReLu. Also, the optimum Kernel size for our model is 3 which is a tried and tested information for a dataset of this kind. Our output layer should have 4 outcomes and thus the Units are set to 4 in last dense layer.

| Index | Model | Number of Layers | Number of Epochs | Train accuracy | Validation accuracy | Validation Loss |
|---|---|---|---|---|---|---|
| 1. | Sequential 1 | 7 | 20 | 67.52 | 79.33 | 56.14 |
| 2. | Sequential 2 | 7 | 60 | 89.84 | 90.16 | 21.91 |
| 3. | Sequential 3 | 10 | 20 | 82.44 | 82.68 | 38.47 |
| 4 | Sequential 4 | 10 | 60 | 90.81 | 92.07 | 20.04 |
| 5. | EfficientNet-B0 | 237 | 20 | 99.74 | 99.69 | 1.74 |
| 6. | EfficientNet-B1 | 237-813 | 20 | 99.72 | 99.77 | 1.39 |
| 7. | EfficientNet-B2 | 237-813 | 20 | 99.61 | 99.52 | 1.41 |
| 8. | EfficientNet-B3 | 237-813 | 10 | 99.86 | 99.77 | 1.12 |
| 9. | EfficientNet-B4 | 237-813 | 10 | 99.74 | 98.63 | 5.32 |
| 10. | EfficientNet-B5 | 237-813 | 10 | 99.40 | 94.36 | 16.32 |
| 11. | EfficientNet-B6 | 237-813 | 10 | NA | NA | NA |
| 12 | EfficientNet-B7 | 813 | 10 | NA | NA | NA |

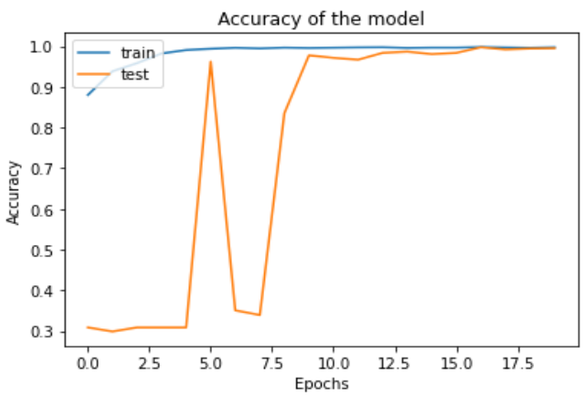
# *Implementations Phase 2*

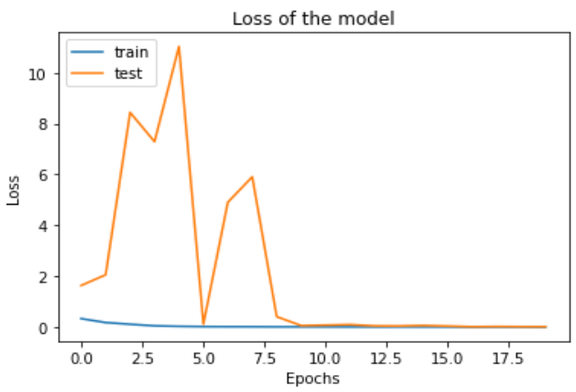
- EfficientNet-B0:

Thus, we have implemented the first of 8 EfficientNet Models i.e., ***EfficientNet-B0.***

Since it contains 237 layers, we cannot for practical reasons display all the layers here. **Refer code*

We trained it for 20 epochs.
We received a validation accuracy of ***99.69*** using EfficientNet-B0.
The recorded loss is ***1.74***

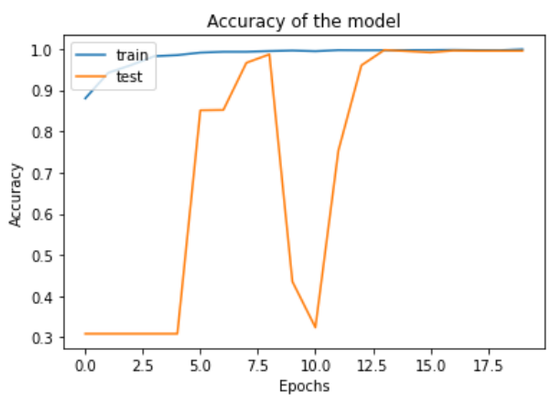


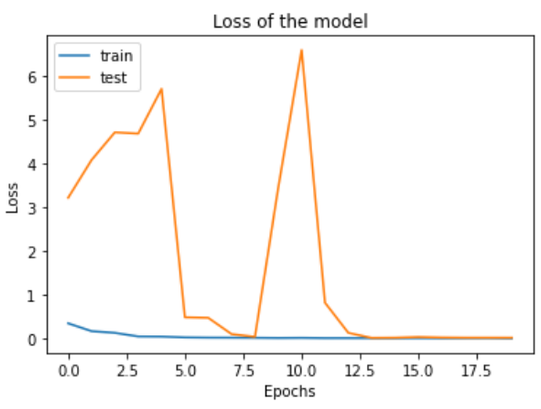




- EfficientNet-B1:

Thus, we have implemented the second EfficientNet Models i.e., ***EfficientNet-B1.***
Since it contains 237-813 layers, we cannot for practical reasons display all the layers here. **Refer code*

We trained it for 20 epochs.
We received a validation accuracy of 99.77.

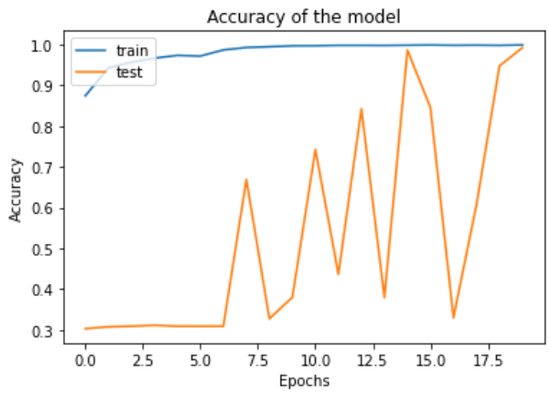


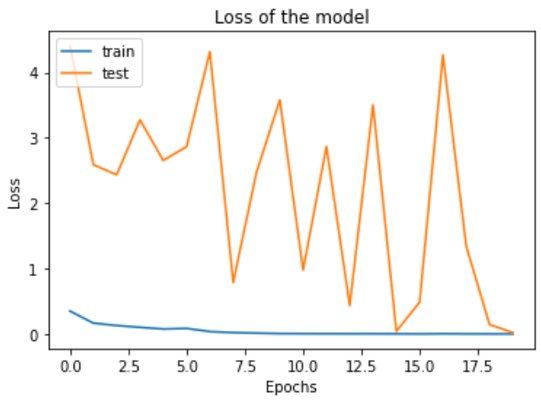



➢ EfficientNet-B2:

Thus, we have implemented the third EfficientNet Models i.e., ***EfficientNet-B2.***
Since it contains 237-813 layers, we cannot for practical reasons display all the layers here. **Refer code*

We trained it for 20 epochs.
We received a validation accuracy of 99.52.

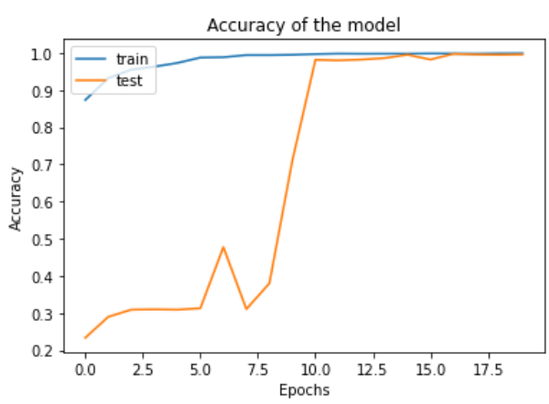


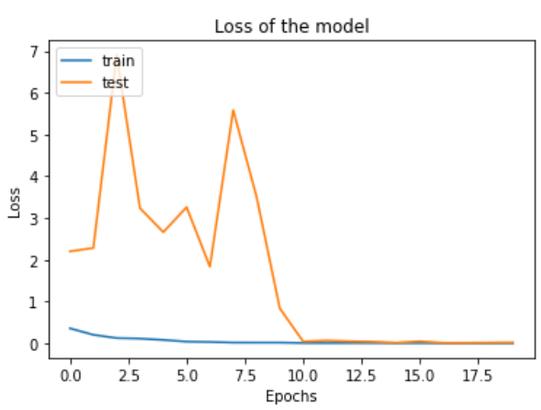




➢ EfficientNet-B3:

Thus, we have implemented the fourth EfficientNet Models i.e., ***EfficientNet-B3.***
Since it contains 237-813 layers, we cannot for practical reasons display all the layers here. **Refer code*

We trained it for 10 epochs.
We received a validation accuracy of 99.77.

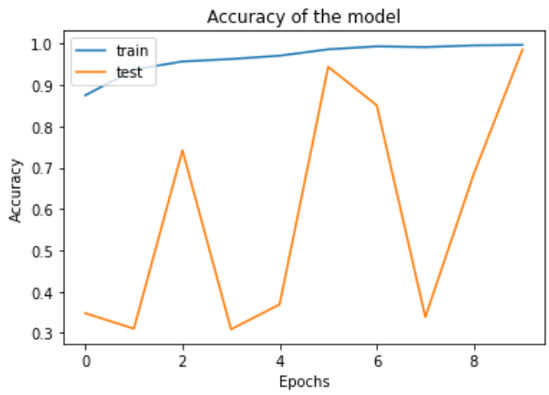


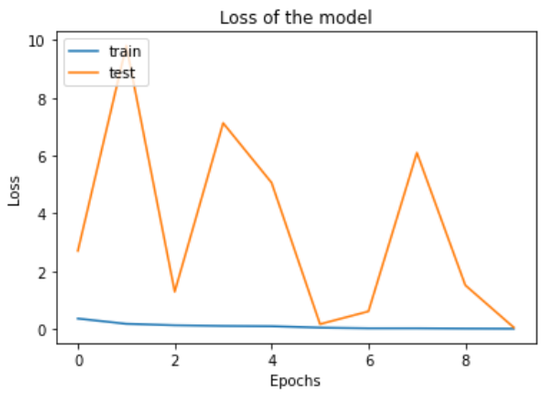



➢ EfficientNet-B4:

Thus, we have implemented the fifth EfficientNet Models i.e.,
***EfficientNet-B4.***
Since it contains 237-813 layers, we cannot for practical reasons display all the layers here. **Refer code*

We trained it for 10 epochs.
We received a validation accuracy of 98.63.

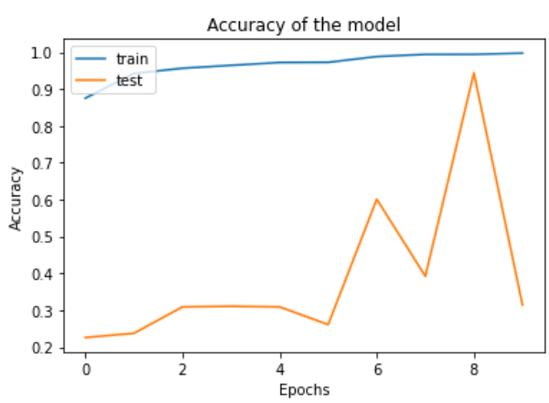


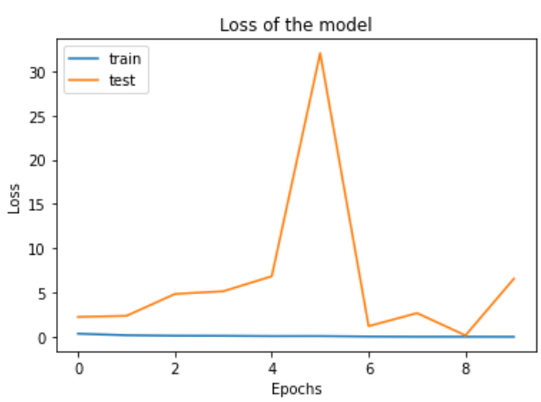




➢ EfficientNet-B5:

Thus, we have implemented the sixth EfficientNet Models i.e.,
***EfficientNet-B5.***
Since it contains 237-813 layers, we cannot for practical reasons display all the layers here. **Refer code*

We trained it for 10 epochs.
We received a validation accuracy of 94.36.
The fact remains that if we had increased the epoch number here, we could've gotten better results around 20 epochs.




Non-trainable params: 0

➢ EfficientNet-B6 & EfficientNet-B7:

We received a Resource Exhausted Error while trying to run the last two Efficient Networks. Also, we received an OOM error which essentially means Out Of Memory Error.

An OOM error occurs when your system goes out of memory. We can try reducing batch size and/or model size and see if it persists. Also, we can check whether other processes are hogging significant memory.

After a little research we reached a safe conclusion where we understood that our systems are not compatible to create tensors of the shape [32,7,7,2064]. Further research is needed to solve this error. Due to limitations by google Collaboratory we couldn't further assess the situation.

```
ResourceExhaustedError                    Traceback (most recent call last)
<ipython-input-98-bb5f0bdc29c9> in <module>()
      4                           verbose=1,
      5                           batch_size=32,
----> 6
callbacks=[tensorboard,checkpoint,reduce_lr])

1 frames
/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     53     ctx.ensure_initialized()
     54     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name,
op_name,
---> 55                                         inputs, attrs, num_outputs)
     56   except core._NotOkStatusException as e:
     57     if name is not None:

ResourceExhaustedError: Graph execution error:

OOM when allocating tensor with shape[32,7,7,2064] and type float on
/job:localhost/replica:0/task:0/device:GPU:0 by allocator GPU_0_bfc
        [[{{node model_6/block6i_activation/mul_1-0-1-TransposeNCHWToNHWC-
LayoutOptimizer}}]]
Hint: If you want to see a list of allocated tensors when OOM happens, add
report_tensor_allocations_upon_oom to RunOptions for current allocation info.
This isn't available when running in Eager mode.
 [Op:__inference_train_function_465570]
```

Regardless of that we certainly have created some of the highest accuracy models all of which have an accuracy above 95%.

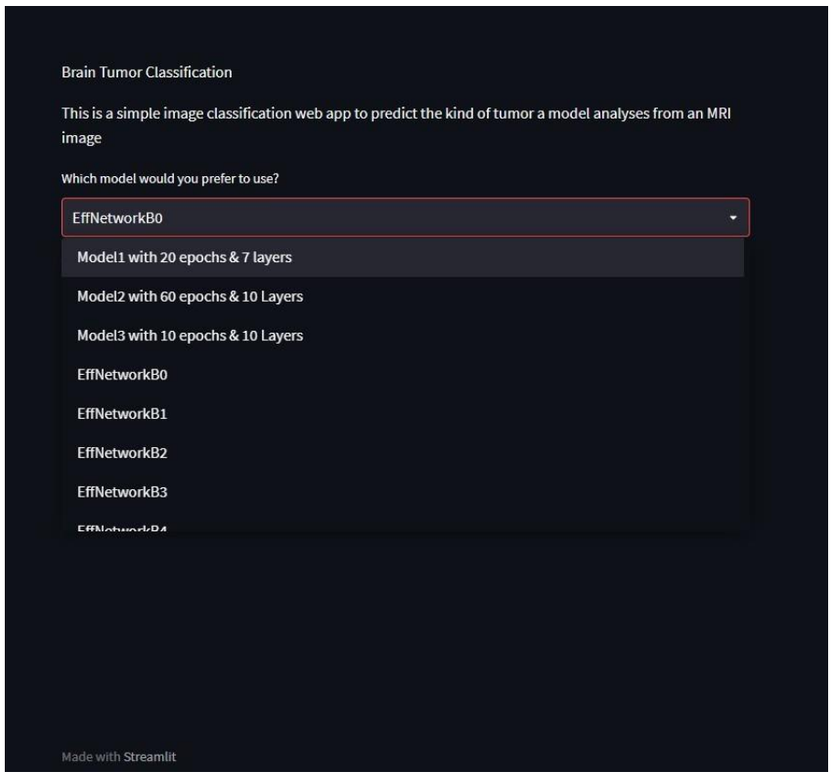
# *Outputs*

We created a simple and efficient User Interface for our various models implementing Streamlit.
The attached screenshots will properly display the implementation part.
The main features of our UI are:

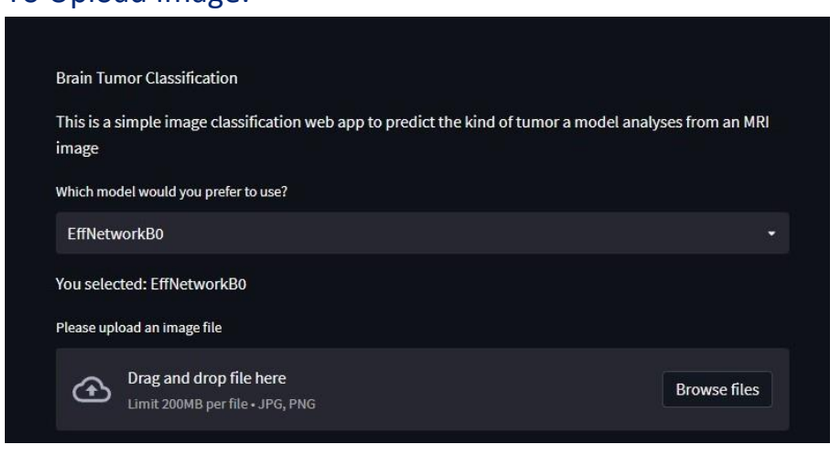
- It is clean and uncluttered.
- It is locally hosted by npx.
- We can test out all the models created till now with a single click. Changing the model will process the already uploaded image and give updated outputs.
- We can upload an image and the code will readjust its dimensions to feed into model.
- It displays the image we entered alongside the results.

The UI:

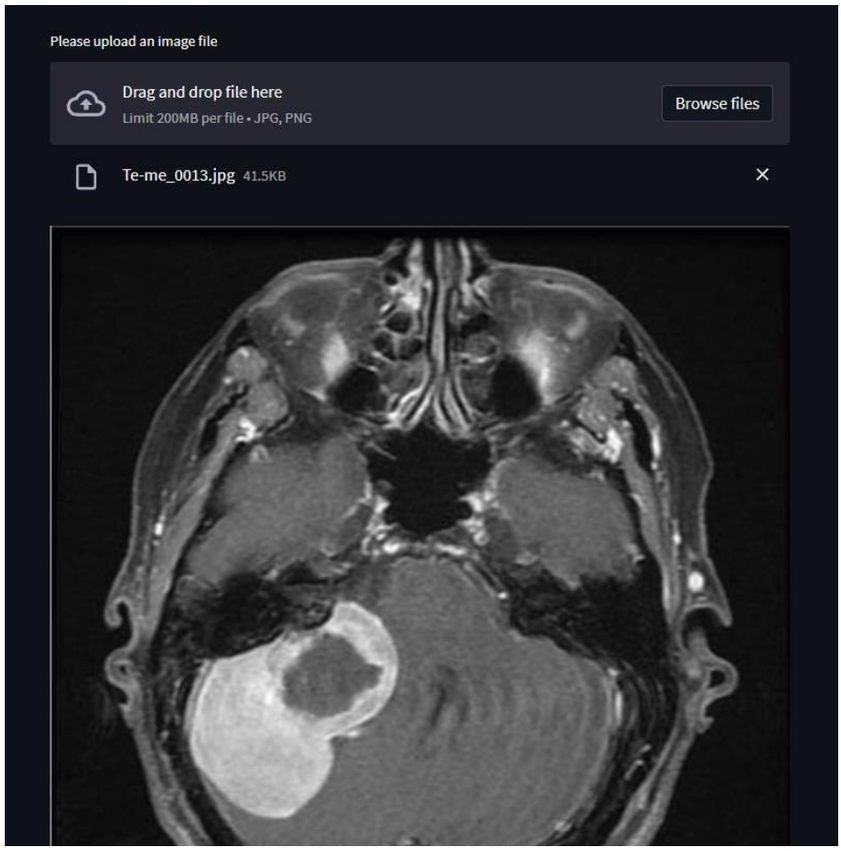
1. To select the model:



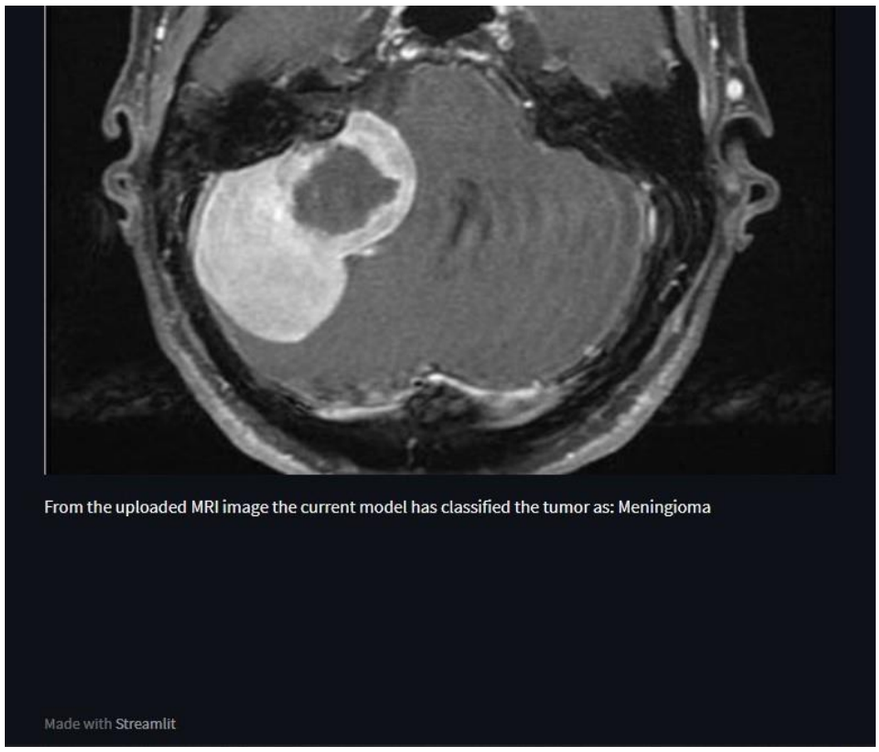

2. To Upload Image:



3. Displayed Image:




4. Displayed Result:




We can reiterate this process for every model and every image we have and still we will always get the results displayed and the UI won't ever fail.

# *Results*

1. By leveraging Model scaling we can get a very high accuracy in an image classification model.

2. Experimenting with Kernel Sizes and Activation Function influences accuracy and loss of the model, but the parameters of various problem sets have already been tested and fixed. There is an optimum way of solving every kind of problem and this problem of image classification required tried and tested method of setting Kernel size 3 and activation function as Relu for maximum results.

| ACTIVATION FUNCTION | SIGMOID | TANH | RELU |
|---|---|---|---|
| Range | 0 to 1 | -1 to 1 | 0 to Infinity |
| Vanishing Gradient Problem | Yes | Yes | No |
| Nature | Non Linear | Non Linear | Linear |
| Zero Centered Activation Function | No | Yes | No |
| Symmetric Function | No | Yes | No |
| Equation | y = 1/(1+e^(-x)) | y = tanh(x) | { xi if x >=0<br>0 if x <=0 } |
| Model Accuracy | Good | Very Good | Excellent |

3. Padding is an important part for capturing maximum features. Not enabling padding can drop accuracy score drastically.

4. Higher the number of epochs higher is the probability of the model overfitting. There must be an optimum number of epochs. We discovered this conundrum while training a variation of Sequential 4 for 60 epochs. The model initially classified every tumor as Meningioma. It overfitted.

5. A small analysis from data on the internet has shown that there are images of Meningioma floating around a lot prevalently than images of other tumors. For testing it is highly advisable to get correctly labelled images so that there is no confusion. Reliability of the dataset we used is also in question since only a doctor can correctly assess and validate the dataset if one might be inclined to check.

6. Google Collab is essentially a powerful tool for leveraging GPU and completing small projects.

7. Using the .h5 method to save the trained models is viable and most suggested for every further project.

8. As the model being introduced with callbacks the accuracy of the model is increased indicating that the callbacks prevent the overfitting of the model and provides efficient training of the model.

9. The use of ReduceLRonPlateau with less patience the model learns better with less epochs.

10. EarlyStopping stops the model's overfitting.

11. Checkpoints are an efficient way to store and load model weights to achieve better accuracy if the model had varying learning.

12. The Usage of GPU has improved the training time and results of the models. Eventually though google collab has it's limitations and lit locks our account from using more computational power.

13. Hypertuning of the models with kernel size, number of filters and number of nodes has increased the accuracy of the model.

14. Transfer Learning is an efficient way to train and develop models.

15. With the increase in the number of layers the accuracy of the models increases. Though in some circumstances the model reaches a stage where it overfits. Analyzing the accuracy and loss graphs can help in determining the optimum epoch number.

16. The experiments with the models indicate that all the models were able to predict more that 98% of the total images that has been provided to the model.

17. Images provided from the internet were also analyzed and we have seen good results with all the EfficientNet models.

# *Conclusion*

Thus, we can conclude that using Convolutional Network and Brain MRI data one can classify the Brain Tumors and detect them. Model's performance can be improved with hyperparameter tuning and with the usage of callbacks from TensorFlow package. Retraining of models makes the model become more narrowed. Usage of GPU has major influence in not only reducing the training time but also improves the mathematical calculations resulting in better accuracy. Transfer learning improves the model's accuracy by adding more layers to the model with getting affected by vanishing gradient problem. Accuracy of those models can also be improved my adding more hidden layers. Using Model Scaling and a larger dataset one might just get a perfect accuracy model for classification of images.

We were also able to create a code to test all our models and various images off the internet and give near to accurate assessment. Based on the testing process and accuracy score, we can select the EfficientNet model as the best model. A simple User Interface was developed that uploads the image and predicts the output.

# *Improvement Scope*

Using larger dataset to scale the model and getting better results on random images from the internet. Classifying the tumors in accordance with its severity and treatability.

We tried to address these issues in phase 2. Though we reached a conclusion that we either need more data or rather more precise data and a brain/neural doctor/surgeon to analyze the MRI images. Our scope lies below the required domain knowledge and is limited by the fact that we ain't no doctors.

# *References:*


1. www.towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37
2. www.towardsdatascience.com/conv2d-to-finally-understand-what-happens-in-the-forward-pass-1bbaafb0b148
3. www.towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1
4. www.distill.pub/2017/feature-visualization/
5. www.https://www.baeldung.com/cs/neural-networks-hidden-layers-criteria
6. https://www.javatpoint.com/pytorch-convolutional-neural-network
7. www.kaggle.com
8. www.medium.com
9. www.towardsdatascience.com
10. www.machinelearningmastery.com
11. https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac
12. www.towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142
13. www.ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html
14. www.aitude.com/comparison-of-sigmoid-tanh-and-relu-activation-functions/
15. www.wandb.ai/ayush-thakur/dl-question-bank/reports/Keras-Layer-Input-Explanation-With-Code-Samples--VmlldzoyMDIzMDU