Classification of Brain MRI Images Using Deep Learning

Suraj Prakash Sharma
(BLENP2DSC20038)

M.Tech Data Science (IInd Semester)

Deep Learning Course Project
Amrita Vishwa Vidyapeetam, School of Engineering, Bangalore

June 7, 2021

Overview

Introduction

- The occurrence of brain tumor patients in India is steadily rising, more and more number of cases are reported each year in India across various age groups.

[1] https://cutt.ly/Wc4DaIE

Introduction

- The occurrence of brain tumor patients in India is steadily rising, more and more number of cases are reported each year in India across various age groups.
- The International Association of Cancer Registries (IARC)[1]reported that there are over 28,000 cases of brain tumours reported in India each year and more than 24,000 people reportedly i.e. 85.72% of the total reported die due to brain tumours annually. Brain tumour's are a serious condition and in most cases fatal if not detected & treated in early stages.

[1] https://cutt.ly/Wc4DaIE

Brain Tumor Types

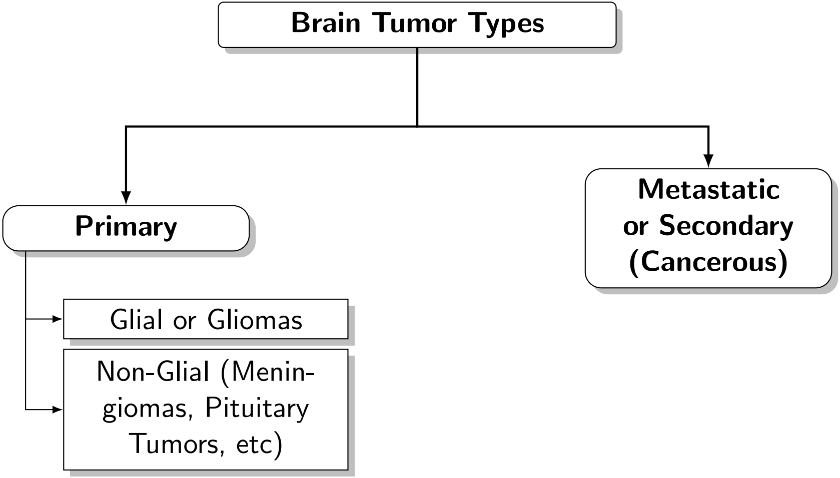

Figure: Brain Tumor Types Chart

About Dataset

Description

The dataset consists of 3,459 MRI images of the brain which belongs to four classes i.e. No Tumor, Meningioma, Glioma, Pituitary Tumor.

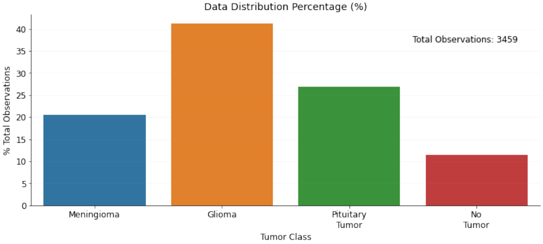

Figure: Dataset Distribution

About Dataset

Description

The dataset consists of 3,459 MRI images of the brain which belongs to four classes i.e. No Tumor, Meningioma, Glioma, Pituitary Tumor.

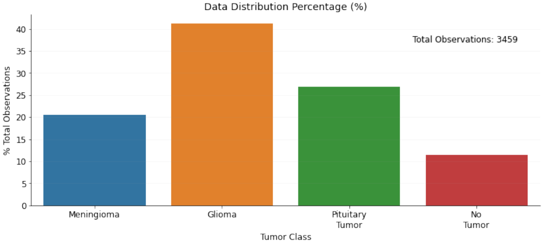

Figure: Dataset Distribution

As you can see from the distribution plot that the dataset is imbalanced i.e 41% Glioma, $\approx$ 27% Pituitary Tumor, $\approx$ 21% Meningioma, $\approx$ 12% No Tumor.

About Dataset: MRI Images

- A brain MRI is one of the most commonly performed techniques of medical imaging. It enables clinicians & doctors to focus on various parts of the brain and examine their anatomy and pathology, using different MRI sequences, such as T1w, **T2w**, or FLAIR.

About Dataset: MRI Images

- A brain MRI is one of the most commonly performed techniques of medical imaging. It enables clinicians & doctors to focus on various parts of the brain and examine their anatomy and pathology, using different MRI sequences, such as T1w, **T2w**, or FLAIR.
- Unlike CT Scan the biggest advantage of MRI is that it uses no radiation. However, it takes longer time to be produced than CT, which is why it's not a primary imaging choice for urgent conditions.

About Dataset: MRI Images

- A brain MRI is one of the most commonly performed techniques of medical imaging. It enables clinicians & doctors to focus on various parts of the brain and examine their anatomy and pathology, using different MRI sequences, such as T1w, **T2w**, or FLAIR.
- Unlike CT Scan the biggest advantage of MRI is that it uses no radiation. However, it takes longer time to be produced than CT, which is why it's not a primary imaging choice for urgent conditions.
- Our dataset is T2w MRI sequence because it allows us to detect pathological changes in the neural tissue.
- The dataset is gathered from the following website: https://cutt.ly/jb5vcpT

Data Preprocessing: Processing .mat files

- The dataset downloaded was not in the image formats (.jpg, .jpeg, .png) rather it was in .mat file format which is a MATLAB file format used mostly to represent images.

Data Preprocessing: Processing .mat files

- The dataset downloaded was not in the image formats (.jpg, .jpeg, .png) rather it was in .mat file format which is a MATLAB file format used mostly to represent images.
- Written a python program for preprocessing .mat files using h5py library to extract important information i.e. tumor class, image data, mask data and represented it as a python dictionary object.

```python
def mat_file_to_dict(filepath: str) -> dict:
    tumor_class = {1: 'meningioma', 2: 'glioma', 3: 'pituitary_tumor'}
    tumor_data_dict = {}
    with h5py.File(filepath, mode = 'r') as image_data:
        cjdata_struct = image_data['cjdata']
        tumor_data_dict['class'] = tumor_class[int(cjdata_struct['label'][0, 0])]
        tumor_data_dict['image'] = cjdata_struct['image'][:].transpose()
        tumor_data_dict['tumor_border'] = cjdata_struct['tumorBorder'][0]
        tumor_data_dict['tumor_mask'] = cjdata_struct['tumorMask'][:].transpose()
    return tumor_data_dict
```

Figure: Function to convert .mat file to python dictionary.

Source Code (Google Colab): https://cutt.ly/yb5VhF6

Data Preprocessing: Final Dataset Glimpse

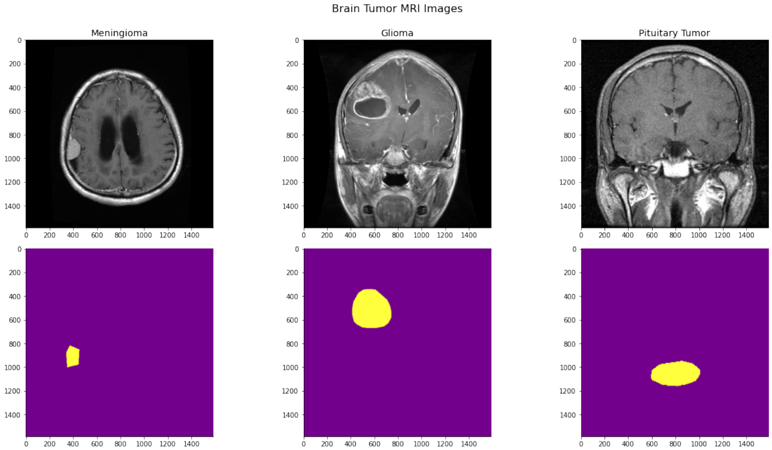

Figure: Brain Tumor MRI Images (T2w)

Final Dataset: https://cutt.ly/4b5VIuH

Training, Validation & Testing Dataset

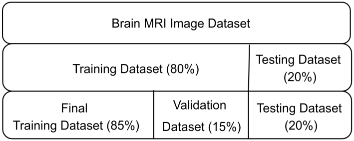

Figure: Partition % of Brain MRI Dataset.

Training, Validation & Testing Dataset

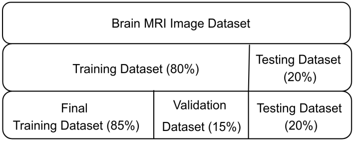

Figure: Partition % of Brain MRI Dataset.

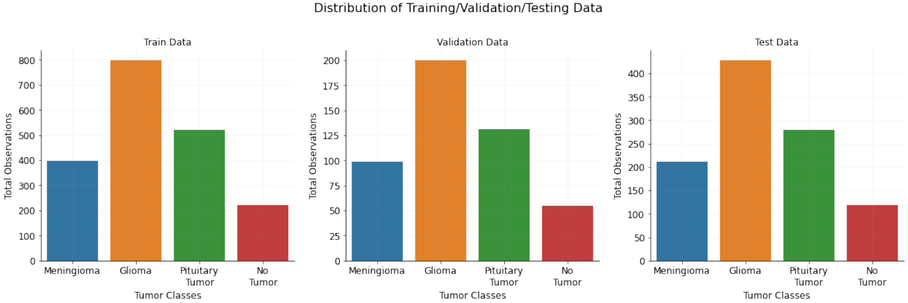

Figure: Data Distribution of Training, Validation & Testing Dataset.

Data Augmentation

- Data augmentation is a technique wherein we genenerate different variances of the same data in order to expose the model to more different types of patterns during the training process. It is one of the techniques which is used to solve the data imbalance problem in the dataset.

Data Augmentation

- Data augmentation is a technique wherein we genenerate different variances of the same data in order to expose the model to more different types of patterns during the training process. It is one of the techniques which is used to solve the data imbalance problem in the dataset.
- For this case, generated different variances of the image by altering the properties like brightness, zoom scale, shear scale, rotating by 45°, & rescaling the image by dividing it by 255.

Data Augmentation

- Data augmentation is a technique wherein we genenerate different variances of the same data in order to expose the model to more different types of patterns during the training process. It is one of the techniques which is used to solve the data imbalance problem in the dataset.
- For this case, generated different variances of the image by altering the properties like brightness, zoom scale, shear scale, rotating by 45°, & rescaling the image by dividing it by 255.
- Following slide contains the source-code written to do the data augmentation.

Brain MRI Image Dataset Augmentation Source Code

```
image_size = 128
batch_size = 32

image_datagen_kwargs = dict(rescale = 1 / 255,
                            rotation_range = 15,
                            width_shift_range = 0.1,
                            zoom_range = 0.01,
                            shear_range = 0.01,
                            brightness_range = [0.3, 1.5],
                            horizontal_flip = True,
                            vertical_flip = True)

train_image_datagen = ImageDataGenerator(**image_datagen_kwargs)
validation_image_datagen = ImageDataGenerator(**image_datagen_kwargs)
test_image_datagen = ImageDataGenerator(**image_datagen_kwargs)

train_dataset = train_image_datagen.flow_from_dataframe(train_data,
                                                        x_col = 'image_filepaths',
                                                        y_col = 'tumor_class',
                                                        seed = 42,
                                                        batch_size = batch_size,
                                                        target_size = (image_size, image_size),
                                                        color_mode = 'rgb')
validation_dataset = validation_image_datagen.flow_from_dataframe(validation_data,
                                                                  x_col = 'image_filepaths',
                                                                  y_col = 'tumor_class',
                                                                  seed = 42,
                                                                  batch_size = batch_size,
                                                                  target_size = (image_size,
                                                                                 image_size),
                                                                  color_mode = 'rgb')
test_dataset = test_image_datagen.flow_from_dataframe(test_data,
                                                      x_col = 'image_filepaths',
                                                      y_col = 'tumor_class',
                                                      seed = 42,
                                                      batch_size = batch_size,
                                                      target_size = (image_size, image_size),
                                                      color_mode = 'rgb')
```

Training Dataset Glimpse

Figure: Samples of Training Set after Data Augmentation.

Validation Dataset Glimpse

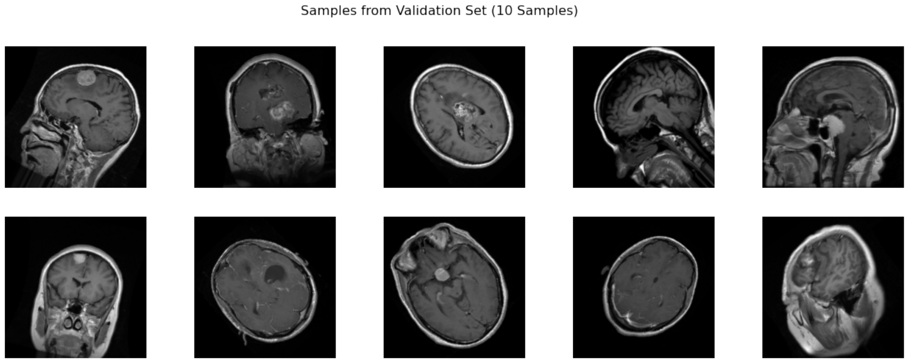

Figure: Samples of Validation Set after Data Augmentation.

Testing Dataset Glimpse

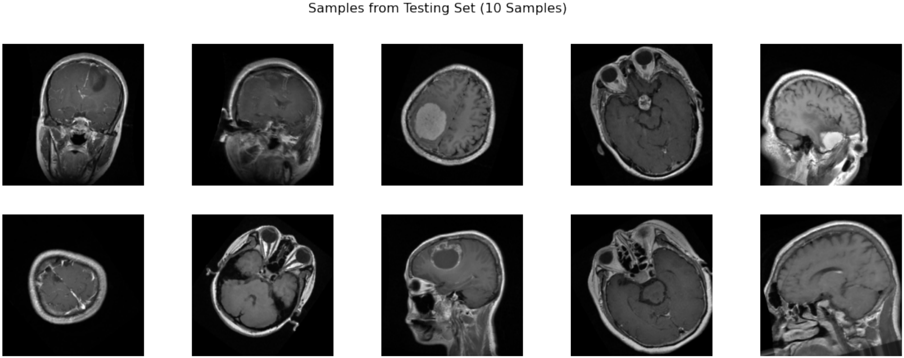

Figure: Samples of Testing Set after Data Augmentation.

Model Description: Multi-Layer Perceptron Based Architecture

- The input to the MLP will be a 1-D vector representation of the image of dimension $(128, 128, 3)$ i.e. $49,152$ neurons in the input layer.

Model Description: Multi-Layer Perceptron Based Architecture

- The input to the MLP will be a 1-D vector representation of the image of dimension $(128, 128, 3)$ i.e. $49, 152$ neurons in the input layer.
- Total number of hidden layers is 3 with ReLU activation function.

Model Description: Multi-Layer Perceptron Based Architecture

- The input to the MLP will be a 1-D vector representation of the image of dimension $(128, 128, 3)$ i.e. $49, 152$ neurons in the input layer.
- Total number of hidden layers is 3 with ReLU activation function.
- Total number of neurons in the output layer is 4 as there are 4 classes with a softmax activation function.

Architecture of MLP Used

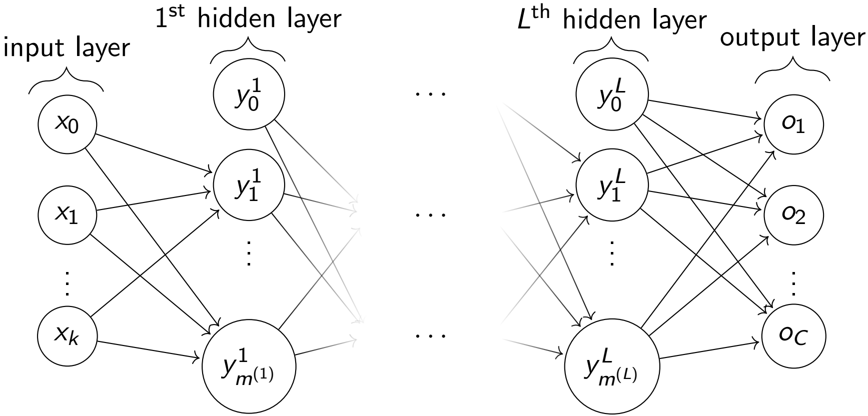

Figure: Multi-Layer Perceptron Model with $k = 49,152$ input units and $C = 4$ output units. The l^{th} hidden layer contains $m^{(l)}$ hidden units.

Model Description: AlexNet Based Architecture

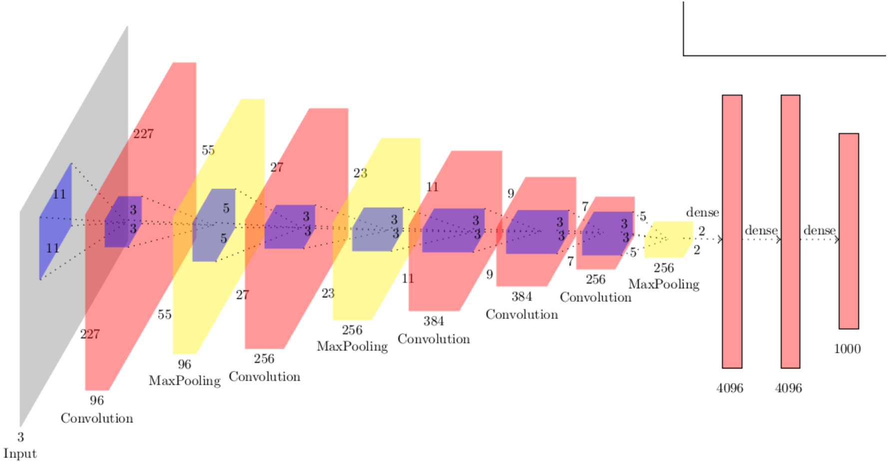

Figure: **AlexNet Architecture For ImageNet** (27.55M Parameters)

AlexNet Variant

- The variant of **AlexNet** which is used consists of 5 Convolutional Layers and 2 Dense Layers (128, 64) and the number of neurons in the output layer is 4 as there are four classes.

AlexNet Variant

- The variant of **AlexNet** which is used consists of 5 Convolutional Layers and 2 Dense Layers (128, 64) and the number of neurons in the output layer is 4 as there are four classes.
- Dimension of the input image is $(128, 128, 3)$ and thats the reason why the total number parameters in the model is significantly less than the AlexNet architecture which was trained on ImageNet benchmark dataset.

AlexNet Variant

- The variant of **AlexNet** which is used consists of 5 Convolutional Layers and 2 Dense Layers (128, 64) and the number of neurons in the output layer is 4 as there are four classes.
- Dimension of the input image is $(128, 128, 3)$ and thats the reason why the total number parameters in the model is significantly less than the AlexNet architecture which was trained on ImageNet benchmark dataset.
- Total Parameters: $3,892,420 \approx 3.9M$.

AlexNet Variant Implementation Keras

```
alexnet_cnn = Sequential()
alexnet_cnn.add(Conv2D(96,
                       kernel_size = 11,
                       strides = 4,
                       activation = 'relu',
                       input_shape = (image_size, image_size, 3),
                       name = 'Conv2D-1'))
alexnet_cnn.add(BatchNormalization(name = 'Batch-Normalization-1'))
alexnet_cnn.add(MaxPool2D(pool_size = 3, strides = 2, name = 'Max-Pooling-1'))
alexnet_cnn.add(Conv2D(256, kernel_size = 5, padding = 'same', activation = 'relu', name = 'Conv2D-2'))
alexnet_cnn.add(BatchNormalization(name = 'Batch-Normalization-2'))
alexnet_cnn.add(MaxPool2D(pool_size = 3, strides = 2, name = 'Max-Pooling-2'))
alexnet_cnn.add(Conv2D(384, kernel_size = 3, padding = 'same', activation = 'relu', name = 'Conv2D-3'))
alexnet_cnn.add(BatchNormalization(name = 'Batch-Normalization-3'))
alexnet_cnn.add(Conv2D(384, kernel_size = 3, padding = 'same', activation = 'relu', name = 'Conv2D-4'))
alexnet_cnn.add(BatchNormalization(name = 'Batch-Normalization-4'))
alexnet_cnn.add(Conv2D(256, kernel_size = 3, padding = 'same', activation = 'relu', name = 'Conv2D-5'))
alexnet_cnn.add(BatchNormalization(name = 'Batch-Normalization-5'))
alexnet_cnn.add(MaxPool2D(pool_size = 3, strides = 2, name = 'Max-Pooling-3'))
alexnet_cnn.add(Flatten(name = 'Flatten-Layer-1'))
alexnet_cnn.add(Dense(128, activation = 'relu', name = 'Hidden-Layer-1'))
alexnet_cnn.add(Dropout(rate = 0.5, name = 'Dropout-Layer-1'))
alexnet_cnn.add(Dense(64, activation = 'relu', name = 'Hidden-Layer-2'))
alexnet_cnn.add(Dropout(rate = 0.5, name = 'Dropout-Layer-2'))
alexnet_cnn.add(Dense(4, activation = 'softmax', name = 'Output-Layer'))
alexnet_cnn.compile(optimizer = 'Adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
alexnet_cnn.summary()
```

Figure: AlexNet Implementation Source Code

Model Description: Inception V3 Based Architecture [Sze+15]

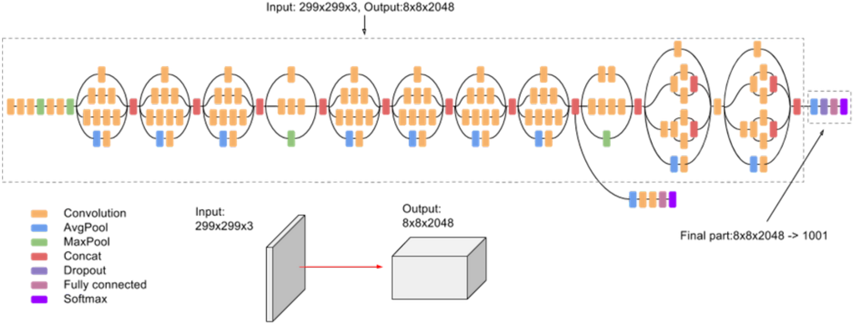

Figure: **InceptionV3 Model Architecture** (24M Parameters).

InceptionV3 Variant Implementation Keras

- The implementation model consisted of $23,905,060 \approx 23.9M$ parameters.

```
inception_v3_model = InceptionV3(include_top = False, input_shape = (image_size, image_size, 3),
pooling = 'avg')
inception_v3_model.trainable = False

inception_cnn_model = Sequential()
inception_cnn_model.add(inception_v3_model)
inception_cnn_model.add(Flatten())
inception_cnn_model.add(Dense(1024, activation = 'relu', name = 'Hidden-Layer-1'))
inception_cnn_model.add(Dense(4, activation = 'softmax', name = 'Output-Layer'))
inception_cnn_model.compile(optimizer = 'Adam',
                            loss = 'categorical_crossentropy',
                            metrics = ['accuracy'])
inception_cnn_model.summary()
```

Figure: InceptionV3 Implementation Source Code

Results: Multi-Layer Perceptron

Figure: Multi-Layer Perceptron Training Statistics.

Results: Multi-Layer Perceptron

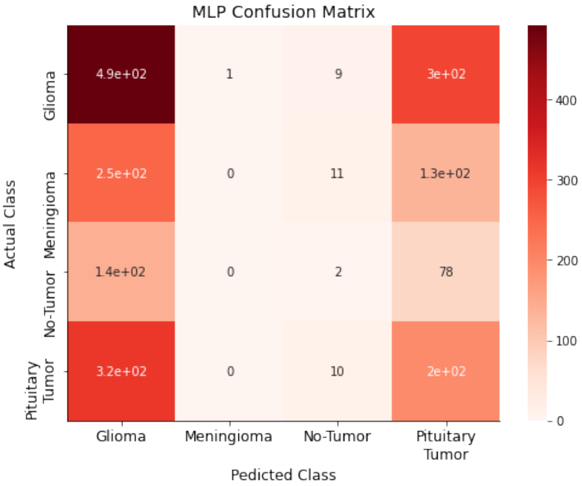

Figure: Multi-Layer Perceptron Confusion Matrix

Results: AlexNet

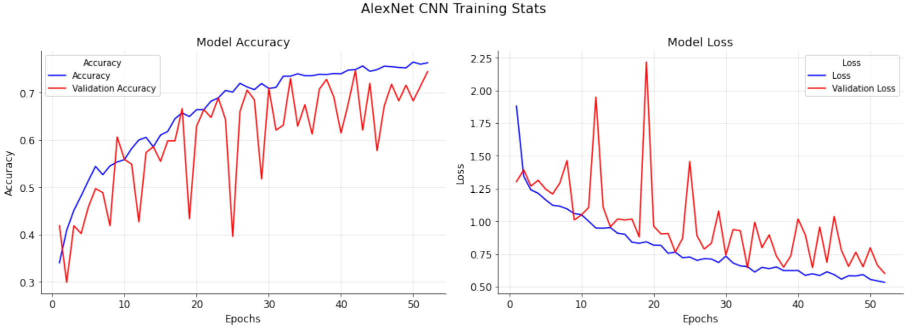

Figure: AlexNet Training Statistics

Results: AlexNet

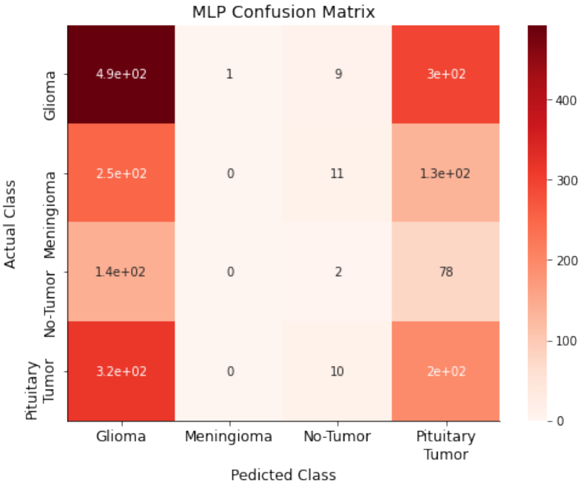

Figure: AlexNet Confusion Matrix

Results: InceptionV3

Figure: Inception V3 Training Statistics

Results: InceptionV3

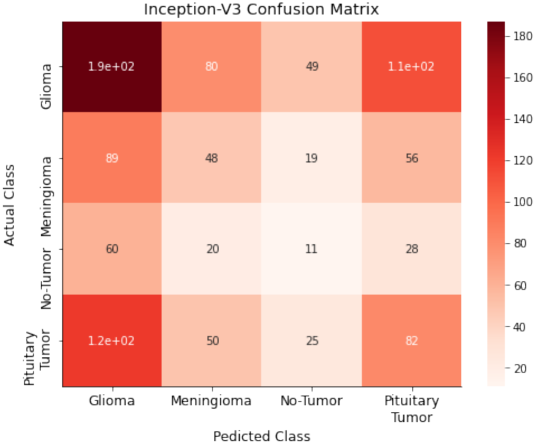

Figure: Inception V3 Confusion Matrix

Conclusions

- The pre-trained model **Inception V3** has performed very well as compared to AlexNet and Multi-Layer Perceptron.

Future Work

- To incorporate a Data Augmentation pipeline to efficiently generate various different variants of the iamges to make the model more roboust.

Future Work

- To incorporate a Data Augmentation pipeline to efficiently generate various different variants of the iamges to make the model more roboust.
- Training process will be migrated to TPUs (Tensor Processing Units) by representing the data in TFRecord format for significant reduction in training time.

Future Work

- To incorporate a Data Augmentation pipeline to efficiently generate various different variants of the iamges to make the model more roboust.
- Training process will be migrated to TPUs (Tensor Processing Units) by representing the data in TFRecord format for significant reduction in training time.
- Implementation of R-CNN to not only detect a image which has a tumor in it but to also label the location of the tumor in the image.

References

Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: (2015). arXiv: 1512.00567 [cs.CV].

Thank You

(The sildes were created using LaTeX.)

For Source Code: Refer Google Colab Notebook