PROJECT REPORT

Detecting Brain Tumor in MRI images using Deep Learning

By:- Mahin Anup

ABSRATCT

A brain tumour is an abnormal growth of brain cells in an uncontrollable way and is one of the most dangerous diseases according to a recent study. This disease requires early and accurate detection methods. It has been observed that manual identification of brain tumors and tracking their changes over time are tedious and error-prone activities. In this project, deep learning algorithms have been deployed to perform an automated brain tumour detection using brain MRI images and measure its performance. The proposed methodology aims to differentiate between a normal brain and a brain with some kind of tumor

Table Of Contents

Statement of Purpose

The main purpose of this project is to build a robust CNN model that can classify if the subject has a tumor or not based on brain MRI images with a good accuracy for medical grade application. Every year, a large number of deaths are caused due to brain tumour and as mentioned earlier, early detection is essential. With this project of mine, I hope to create a model that can detect brain tumour with the highest accuracy possible with the limited dataset and computational power.

PROCEDURE

The project was divided into Five stages namely: -

- Data acquisition
- Data pre-processing
- Data augmentation
- Model architecture
- Training and testing the model

DATA ACQUISITION

For this project I've used the following Kaggle dataset **https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection**

- This dataset contains a total of **253 Brain MRI images**
- **155** of them are images of brain with tumour and **98** of them are images of normal brain.

As the images are divided only into 2 folders i.e. Yes and No, I divided the dataset into Train, Test and Val folders for the ease of training the model.

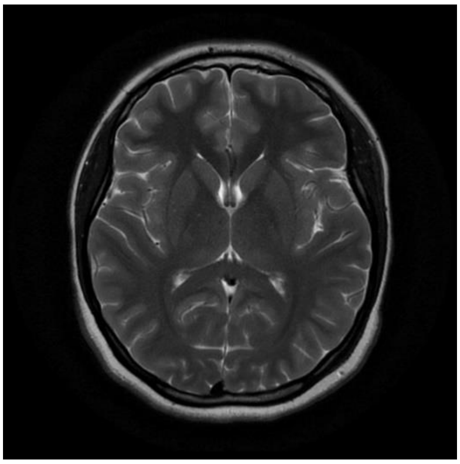
```
.
├── TEST
│   ├── NO
│   └── YES
├── TRAIN
│   ├── NO
│   └── YES
└── VAL
    ├── NO
    └── YES

9 directories
```

DATA PREPROCESSING

Image contouring and cropping - Crop the part of the image that contains only the brain

Before Cropping



After Cropping

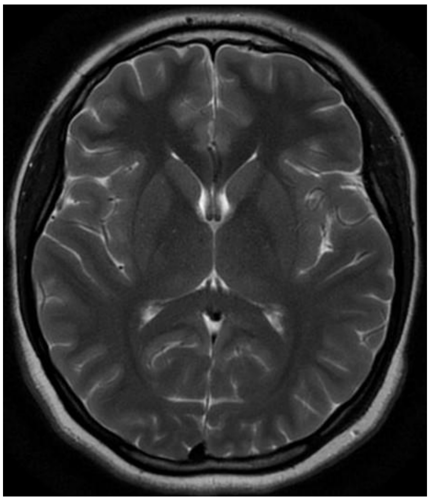


Image resizing - As images should have the same shape to feed it as an input to the neural network. Here for VGG16 it is (224,224)

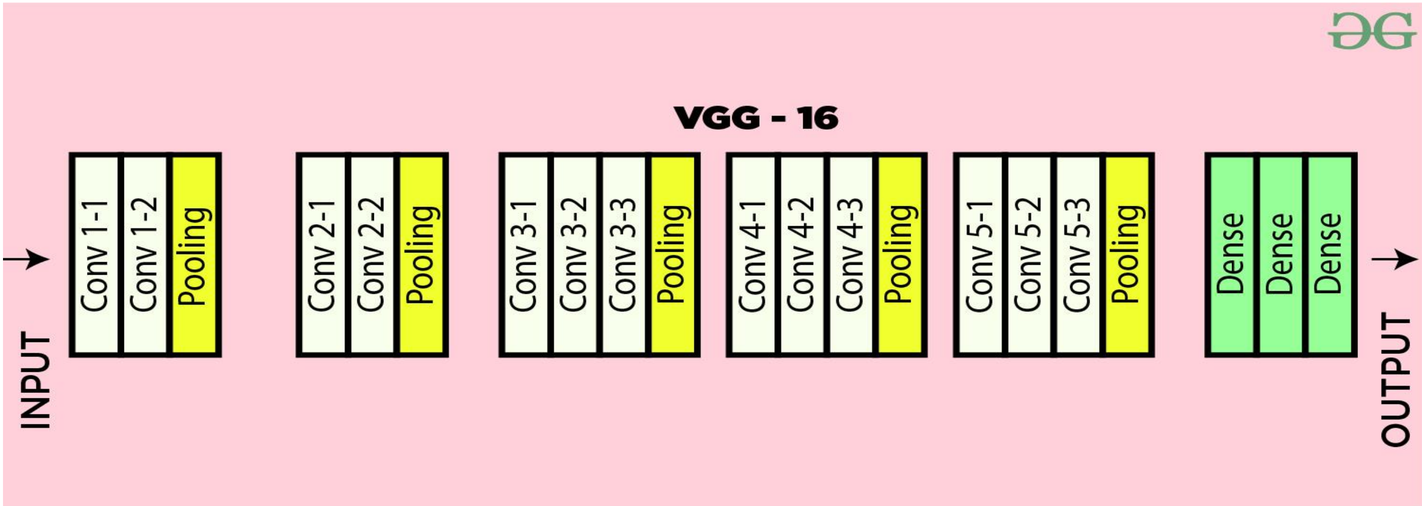
DATA AUGMENTATION

- As the dataset is very small with a total of only 253 images, we need to perform data augmentation.

- In Keras this can be done via the **keras.preprocessing.image.ImageDataGenerator** class.
- Data augmentation not only increases the size of dataset but also significantly increases the diversity of data available for training models, without actually collecting new data

MODEL ARCHITECTURE

I’ve used a predefined VGG16 network for the purpose of transfer learning.



Model Summary

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 25089
=================================================================
Total params: 14,739,777
Trainable params: 25,089
Non-trainable params: 14,714,688
_________________________________________________________________
```

TRAINING AND TESTING

The model was trained with epochs =100 and early stopping monitoring the “val_acc” with patience = 6.

The steps per epochs was 50 and the validation steps was 25.

Stepwise training data

Epoch 1/100

50/50 [==============================] - 26s 519ms/step - loss: 3.3611 - acc: 0.6443 - val_loss: 2.2074 - val_acc: 0.6709

Epoch 2/100

50/50 [==============================] - 22s 437ms/step - loss: 2.1932 - acc: 0.7265 - val_loss: 2.1229 - val_acc: 0.6487

Epoch 3/100

50/50 [==============================] - 24s 480ms/step - loss: 2.0471 - acc: 0.7222 - val_loss: 1.7276 - val_acc: 0.7405

Epoch 4/100

50/50 [==============================] - 22s 449ms/step - loss: 1.5012 - acc: 0.8157 - val_loss: 1.6466 - val_acc: 0.7583

Epoch 5/100

50/50 [==============================] - 23s 454ms/step - loss: 1.3201 - acc: 0.8395 - val_loss: 2.4264 - val_acc: 0.7120

Epoch 6/100

50/50 [==============================] - 23s 457ms/step - loss: 1.4367 - acc: 0.8456 - val_loss: 1.5555 - val_acc: 0.7816

Epoch 7/100

50/50 [==============================] - 22s 440ms/step - loss: 1.4631 - acc: 0.8479 - val_loss: 1.4692 - val_acc: 0.8386

Epoch 8/100

50/50 [==============================] - 23s 463ms/step - loss: 0.9829 - acc: 0.8625 - val_loss: 1.4379 - val_acc: 0.8013

Epoch 9/100

50/50 [==============================] - 22s 433ms/step - loss: 0.9181 - acc: 0.8903 - val_loss: 1.1030 - val_acc: 0.8671

Epoch 10/100

50/50 [==============================] - 22s 441ms/step - loss: 1.1738 - acc: 0.8404 - val_loss: 1.3315 - val_acc: 0.8418

Epoch 11/100

50/50 [==============================] - 23s 457ms/step - loss: 0.7890 - acc: 0.8997 - val_loss: 1.1203 - val_acc: 0.8544

Epoch 12/100

50/50 [==============================] - 21s 428ms/step - loss: 0.6999 - acc: 0.9122 - val_loss: 0.9990 - val_acc: 0.8411

Epoch 13/100

50/50 [==============================] - 23s 458ms/step - loss: 0.6692 - acc: 0.9110 - val_loss: 1.1640 - val_acc: 0.8608

Epoch 14/100

50/50 [==============================] - 22s 433ms/step - loss: 0.8404 - acc: 0.9089 - val_loss: 1.0977 - val_acc: 0.8449

Epoch 15/100

50/50 [==============================] - 22s 431ms/step - loss: 0.8449 - acc: 0.8899 - val_loss: 1.4844 - val_acc: 0.8734

Epoch 16/100

50/50 [==============================] - 23s 452ms/step - loss: 0.4923 - acc: 0.9317 - val_loss: 1.1117 - val_acc: 0.8609

Epoch 17/100

50/50 [==============================] - 22s 446ms/step - loss: 0.5004 - acc: 0.9292 - val_loss: 1.0082 - val_acc: 0.8734

Epoch 18/100

50/50 [==============================] - 22s 432ms/step - loss: 0.9575 - acc: 0.8874 - val_loss: 1.1232 - val_acc: 0.8829

Epoch 19/100

50/50 [==============================] - 23s 463ms/step - loss: 0.5028 - acc: 0.9329 - val_loss: 0.9911 - val_acc: 0.8797

Epoch 20/100

50/50 [==============================] - 22s 443ms/step - loss: 0.3827 - acc: 0.9511 - val_loss: 1.3120 - val_acc: 0.9007

Epoch 21/100

50/50 [==============================] - 23s 451ms/step - loss: 0.7517 - acc: 0.9208 - val_loss: 1.1348 - val_acc: 0.8829

Epoch 22/100

50/50 [==============================] - 24s 485ms/step - loss: 0.4329 - acc: 0.9411 - val_loss: 1.1416 - val_acc: 0.9019

Epoch 23/100

50/50 [==============================] - 24s 476ms/step - loss: 0.4901 - acc: 0.9342 - val_loss: 1.0675 - val_acc: 0.8608

Epoch 24/100

50/50 [==============================] - 25s 494ms/step - loss: 0.3641 - acc: 0.9498 - val_loss: 1.2673 - val_acc: 0.8808

Epoch 25/100

50/50 [==============================] - 22s 445ms/step - loss: 0.5092 - acc: 0.9396 - val_loss: 1.1518 - val_acc: 0.8608

Epoch 26/100

50/50 [==============================] - 23s 464ms/step - loss: 0.4388 - acc: 0.9309 - val_loss: 1.1409 - val_acc: 0.8797

Epoch 27/100

50/50 [==============================] - 23s 455ms/step - loss: 0.3242 - acc: 0.9524 - val_loss: 1.3740 - val_acc: 0.8987

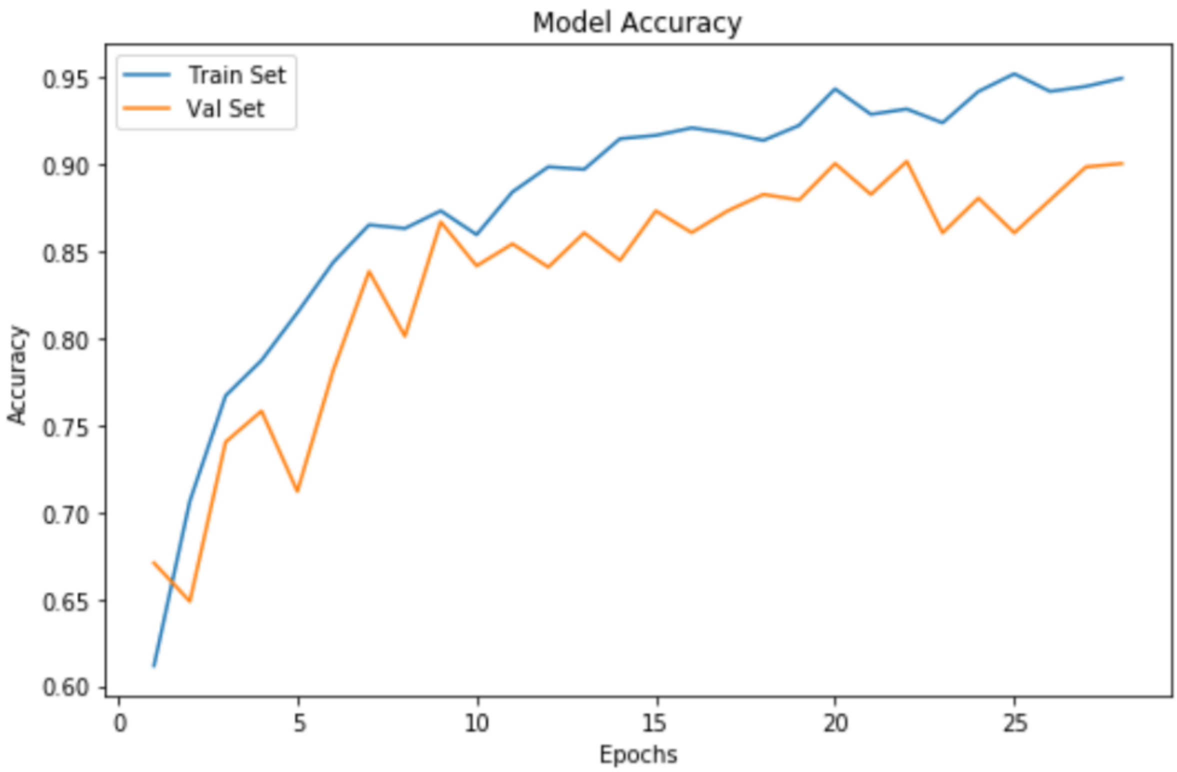
Epoch 28/100

50/50 [==============================] - 22s 435ms/step - loss: 0.3314 - acc: 0.9573 - val_loss: 1.2144 - val_acc: 0.9007

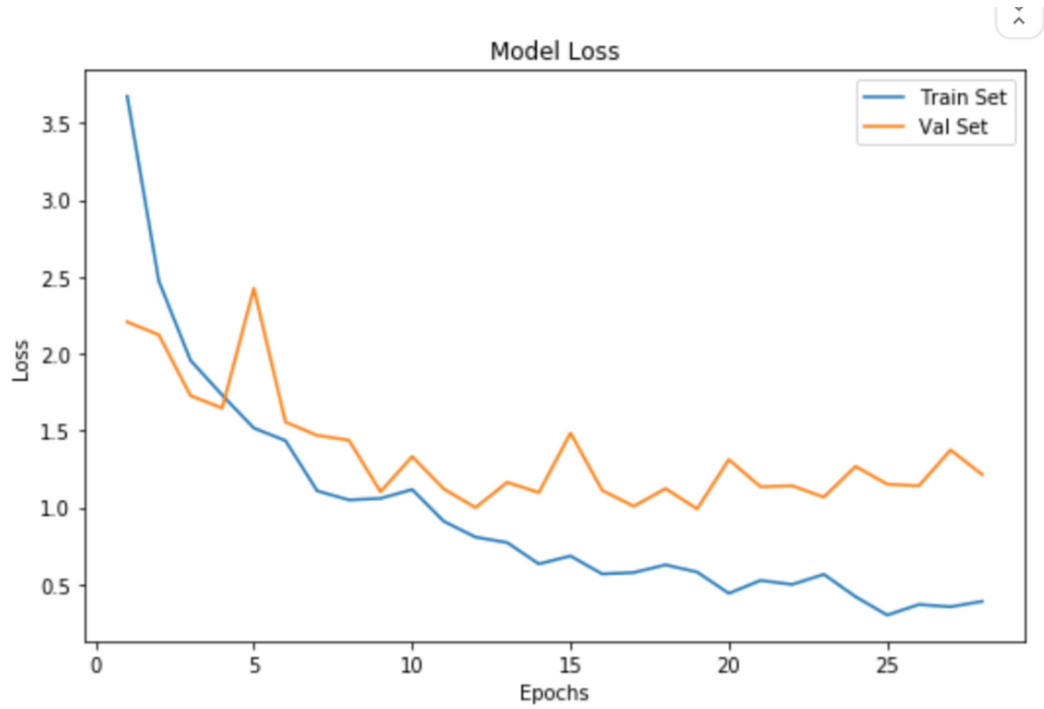
OBSERVATIONS

1. Val Accuracy = 0.90
2. Test Accuracy = 0.70

Model Accuracy



Model Loss



CONCLUSION

This project was a combination of CNN and computer vison problem. At the end I'm able to get good accuracy but it can be improved by increasing dataset and through model hyperparameter tuning. The accuracy that the model achieved is much greater than the baseline percentage. There is also further scope of research in this field as a model should not only detect the tumor but also identify it's location and mark it for the doctors.

REFERENCES

1. Deep learning based brain tumor classification and detection system Authors: ALİ ARI, DAVUT HANBAY
2. S. K. Shil, F. P. Polly, M. A. Hossain, M. S. Ifthekhar, M. N. Uddin and Y. M. Jang, "An improved brain tumor detection and classification mechanism," 2017 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, 2017, pp. 54-57, doi: 10.1109/ICTC.2017.8190941.
3. Brain Tumor Segmentation Using Deep Learning by Type Specific Sorting of Images Zahra Sobhaninia, Safiyeh Rezaei, Alireza Noroozi, Mehdi Ahmadi, Hamidreza Zarrabi, Nader Karimi, Ali Emami, Shadrokh Samavi Department of Electrical and Computer Engineering, Isfahan University of Technology, Isfaha
4. https://www.pyimagesearch.com/2016/04/11/finding-extreme-points-in-contours-with-opencv/
5. https://github.com/MohamedAliHabib/Brain-Tumor-Detection
6. https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/