Brain Tumor Detection

Presenter: Bishrul Farha Imthiyaz

Road Map

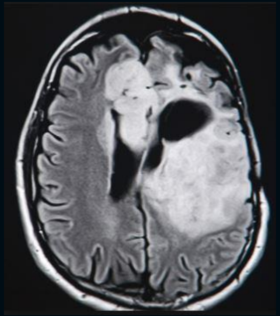
1. Data Discovery & Formulation
2. Data Preparation & Processing
3. Design a Model
4. Model Building
5. Results
6. Measuring of Effeteness

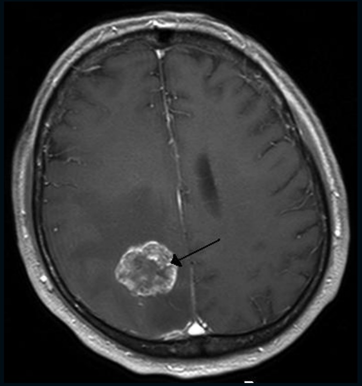
1. Data Discovery & Formulation

- **Business Problem-** Identifying the Brain Tumor by predict whether the patient has a brain tumor or not from automate the process of brain cropping from MRI scans without human intervention.
- **Solution (Critical Objective)-** This project is a combination of CNN model classification problem & Computer Vision problem that would classify and predict if subject has a tumor or not base on MRI scan’s existing and new brain images.

1. Data Discovery & Formulation- cont.

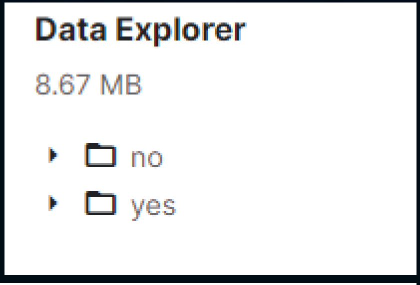
- **About Brain Tumor (Background)-**
  - A brain tumor occurs when abnormal cells form within the brain.
  - There are 2 main types of tumors:
    I. Cancerous (malignant) tumors
    II. Benign tumors.
  - Cancerous tumors can be divided into primary tumors, which start within the brain, and secondary tumors, which have spread from elsewhere, known as brain metastasis tumors.
  - All types of brain tumors may produce symptoms that vary depending on the part of the brain involved.
  - These symptoms may include headaches, seizures, problems with vision, vomiting and mental changes.

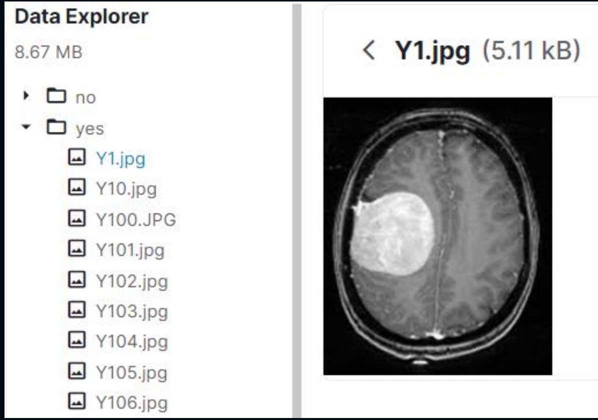


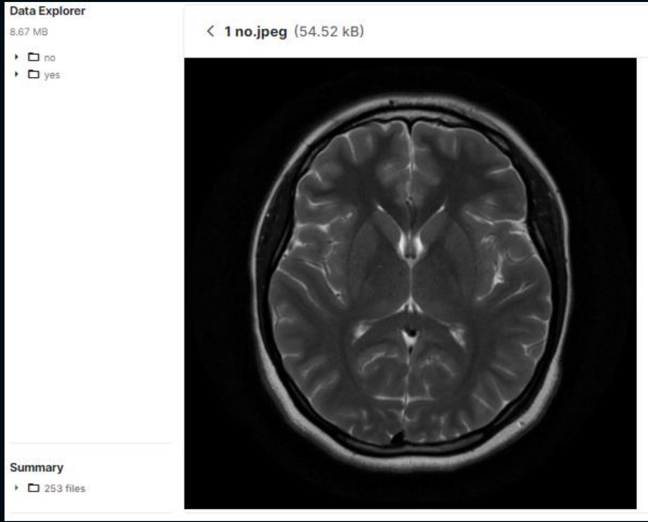


1. Data Discovery & Formulation- cont.

- **About Data -**
  - Data Source- Brain MRI Images for Brain Tumor Detection Dataset from Kaggle Open Source.
  - It consists of MRI scans of two classes:
    - NO - no tumor, encoded as 0 (98 samples)
    - YES - tumor, encoded as 1 (155 samples)







1. Data Discovery & Formulation- cont.

- **Tools & Technologies-**
  - Python Language
  - Python libraries- Numpy, Pandas, Matplotlib, OpenCV, scikit-learn, plotly & tqdm
  - Python Utility Modules- os, shutil, itertools & imutils
  - Python-based frameworks- keras & tensorflow
  - Python IDE- Jupyter Notebook with Anaconda
  - CNN- Neural Network Model for Images
  - VGG-16 – CNN Architecture
- **Target People-** Hospital Director Board, Clinicians and Physicians- Directly and Patients- indirectly

2. Data Preparation & Processing

- Already, our required Data is in digital form from MRI Scan (we have information in digital).
- We are using Kaggle data. So, Already collection, processing and cleaning parts are done. So, we are using Data Acquisition method for data collection.
- Finally, we have our data in 2 folders for this Project which are tumor and no tumor.

3. Design a Model

- In this phase, we will discuss each step by step up to Model Building phase.
- In here, we will explore how, we prepare our data for building CNN model.

3. Design a Model- cont.

- **Creating Folder Directories**
  - TRAIN, TEST and VAL with YES and NO Folders (9 Directories).
  - TRAIN→ 193 Images, TEST→ 10 Images & VAL→ 50 Images

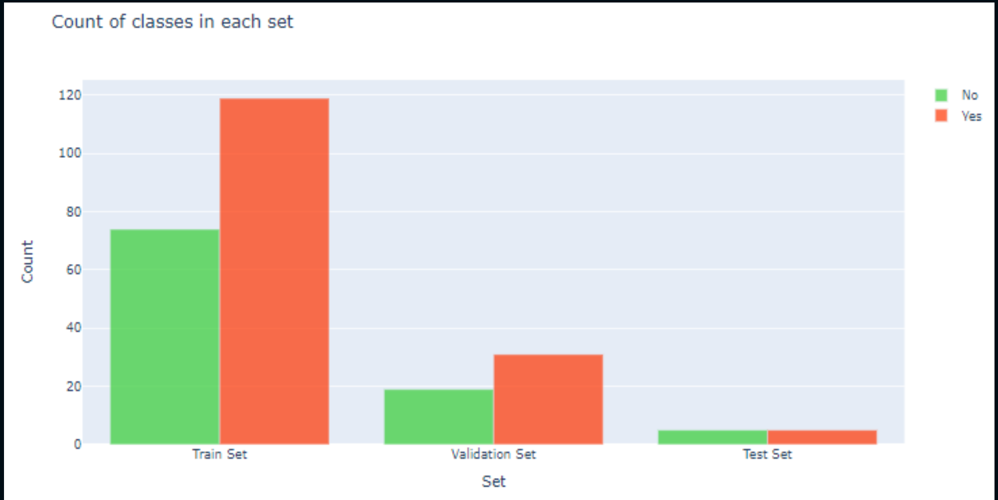
```
TRAIN_DIR = 'TRAIN/'
TEST_DIR = 'TEST/'
VAL_DIR = 'VAL/'
IMG_SIZE = (224,224)

# use predefined function to load the image data into workspace
X_train, y_train, labels = load_data(TRAIN_DIR, IMG_SIZE)
X_test, y_test, _ = load_data(TEST_DIR, IMG_SIZE)
X_val, y_val, _ = load_data(VAL_DIR, IMG_SIZE)
```

```
100%|██████████| 2/2 [00:02<00:00,  1.12s/it]
D:\Anaconda\lib\site-packages\ipykernel_launcher.py:18: VisibleDeprecationWarning:

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different length
s or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.

193 images loaded from TRAIN/ directory.
100%|██████████| 2/2 [00:00<00:00, 34.41it/s]
10 images loaded from TEST/ directory.
100%|██████████| 2/2 [00:00<00:00,  4.73it/s]
50 images loaded from VAL/ directory.
```

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

9 directories
```

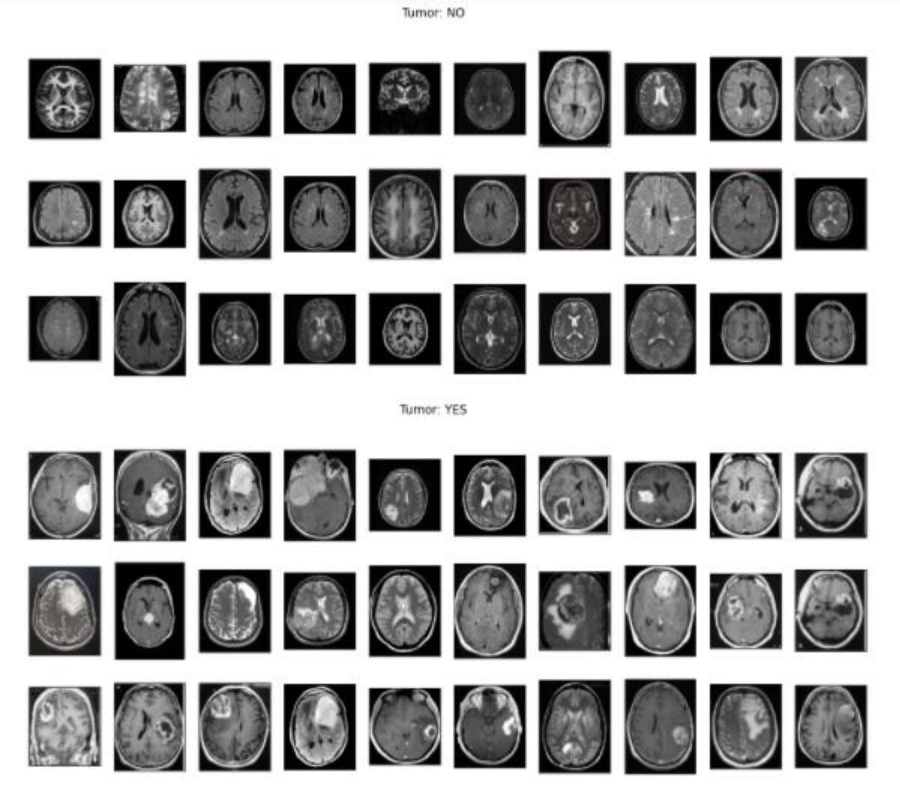
3. Design a Model- cont.

- **Plot the of Count Directories-**
  - TRAIN, TEST and VAL with YES and NO Folders (9 Directories).
  - TRAIN→ Yes-119 Images & NO- 74 Images
  - VAL→ Yes-31 Images & NO- 19 Images
  - TEST→ Yes-5 Images & NO- 5 Images



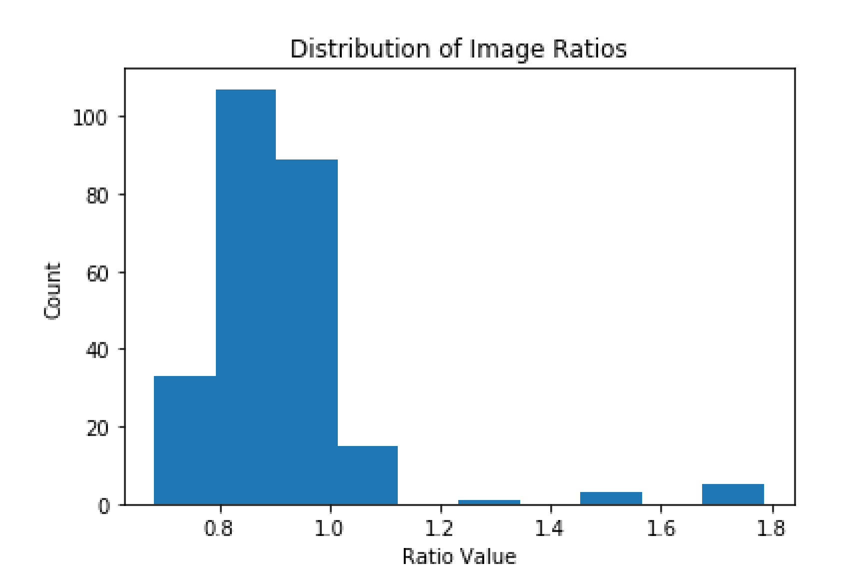
3. Design a Model- cont.

- **Plot for 30 images from the specified set (YES & NO)**
  - As we can see, images have different width and height and different size of "black corners".
  - Since the image size for VGG-16 input layer is (224,224)
  - some wide images may look weird after resizing.



3. Design a Model- cont.

- **Plot the Distribution of Image Ratios**
  - Histogram of ratio distributions (ratio = width/height)
  - So, to solve this problem as a first step of "normalization" would be to crop the brain out of the images.

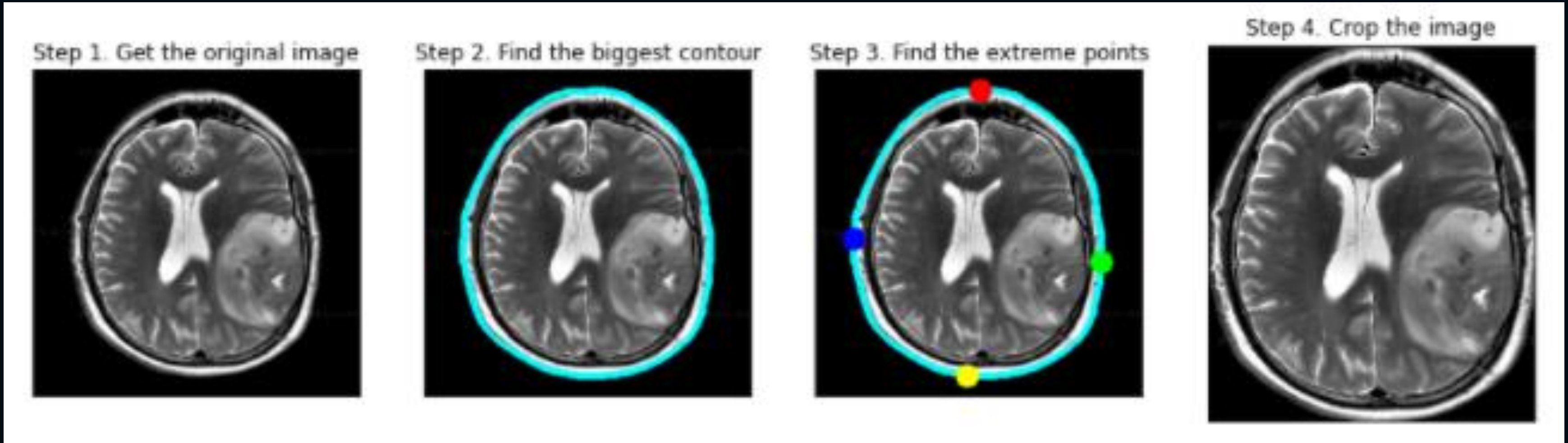


3. Design a Model- cont.

- **Crop the Images**
  - Finds the extreme points on the image and crops the rectangular out of them.
  - We do some steps to crop the images. They are;
    - Change RGB (Color) Image to Grey Image
    - Then, We Apply Gaussian Blur Filter (threshold the image)
    - Then, we Apply Erosion and Dilation filters (to remove any small regions of noise)
    - Next, find contours in threshold image, then grab the largest one
    - Finally, find the extreme points.
  - Apply this crop function with one specific image

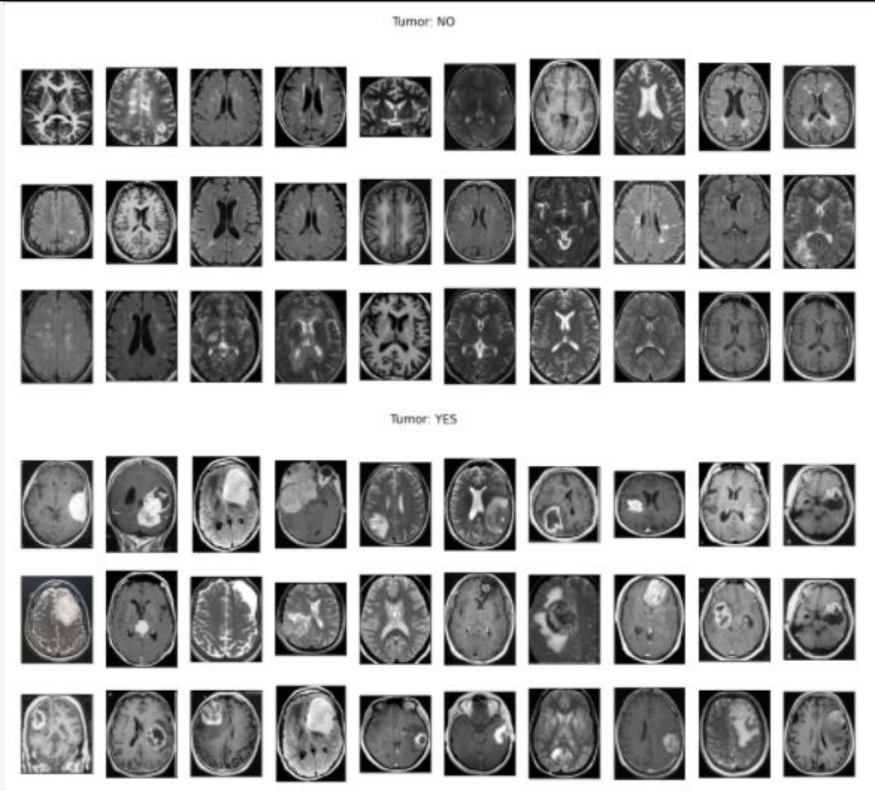
3. Design a Model- cont.

- **Crop the Images- cont.**
  - Plot that image with 4 stages. They are the original image, Find the biggest contour, Find the extreme points and Crop the image.



3. Design a Model- cont.

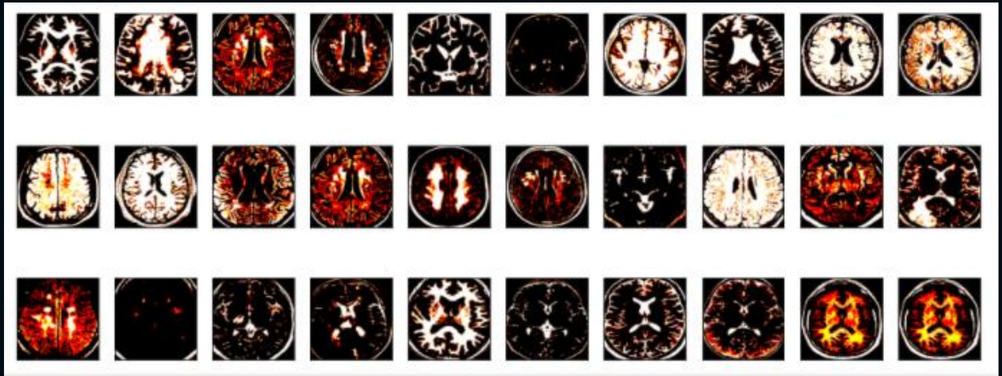
- **Crop the Images- cont.**
  - Apply this crop function to all the images in TRAIN, TEST and VAL directories.
  - Creating new directories called TRAIN_CROP, TEST_CROP, VAL_CROP with YES and NO directories and saving the new cropped images in them.



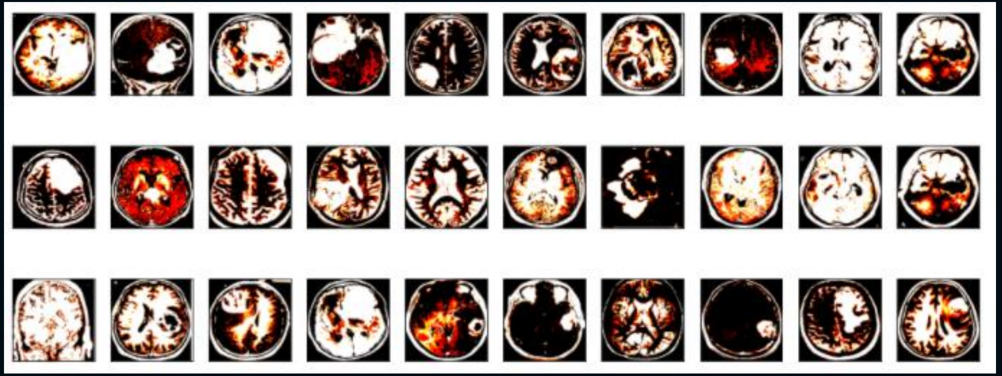
3. Design a Model- cont.

- **Resize the images for VGG-16**
  - We have to resize the images for 224 * 224 size.
  - Then, Plot the images.

NO



YES



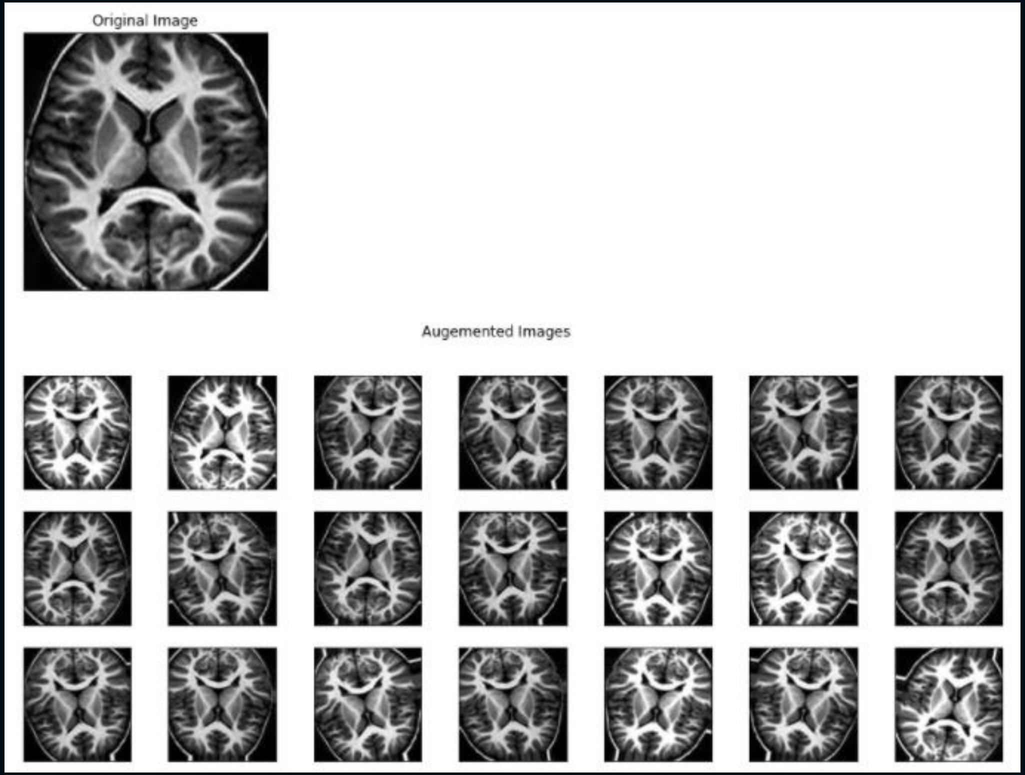
3. Design a Model- cont.

- **Data Augmentation**
  - we used many parameter for this. They are:
    - Rotation range
    - Width shift range
    - Height shift range
    - Rescale
    - Shear range
    - Brightness range
    - Horizontal flip
    - Vertical flip

```
In [32]: # set the paramters we want to change randomly
         demo_datagen = ImageDataGenerator(
             rotation_range=15,
             width_shift_range=0.05,
             height_shift_range=0.05,
             rescale=1./255,
             shear_range=0.05,
             brightness_range=[0.1, 1.5],
             horizontal_flip=True,
             vertical_flip=True
         )
```

3. Design a Model- cont.

- **Data Augmentation- cont.**
  - Applying this function to one image and plotting after transforming images.
  - Then, saving them in ‘preview’ directory (creating).
  - Then, applying same function for whole images in TRAIN_CROP & VAL_CROP.



3. Design a Model- cont.

- **Data Augmentation- cont.**
  - Finally, original images were transformed in TRAIN_CROP & VAL_CROP.
  - So, TRAIN_CROP has 193 images with 2 classes and VAL_CROP has 50 images with 2 classes.

```
In [35]: TRAIN_DIR = 'TRAIN_CROP/'
         VAL_DIR = 'VAL_CROP/'

         train_datagen = ImageDataGenerator(
             rotation_range=15,
             width_shift_range=0.1,
             height_shift_range=0.1,
             shear_range=0.1,
             brightness_range=[0.5, 1.5],
             horizontal_flip=True,
             vertical_flip=True,
             preprocessing_function=preprocess_input
         )

         test_datagen = ImageDataGenerator(
             preprocessing_function=preprocess_input
         )


         train_generator = train_datagen.flow_from_directory(
             TRAIN_DIR,
             color_mode='rgb',
             target_size=IMG_SIZE,
             batch_size=32,
             class_mode='binary',
             seed=RANDOM_SEED
         )


         validation_generator = test_datagen.flow_from_directory(
             VAL_DIR,
             color_mode='rgb',
             target_size=IMG_SIZE,
             batch_size=16,
             class_mode='binary',
             seed=RANDOM_SEED
         )

         Found 193 images belonging to 2 classes.
         Found 50 images belonging to 2 classes.
```

3. Design a Model- cont.

- **Model Selection-**
  - We can use many transfer learning techniques and CNN from the scratch.
  - But, we used VGG16 because, VGG16 is used in many deep learning image classification problems and it is a great building block for learning purpose as it is easy to implement.
  - So, In our Model we have used VGG-16 and CNN to solve classification problem.

4. Model Building

- In this phase,
  - First, we load VGG-16 weights and build this architecture
  - Then, we have created Feed Forward Network (Final layer) of CNN

4. Model Building- cont.

- **First Step (VGG-16)-**

```
In [36]: # Load base model
         vgg16_weight_path = 'C:/Users/Dell/Desktop/Brain_Tumor_Detection/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5'
         base_model = VGG16(
             weights=vgg16_weight_path,
             include_top=False,
             input_shape=IMG_SIZE + (3,)
         )
```

4. Model Building- cont.

- **Second Step (CNN- FFN)-**
  - VGG-16 Base Model
  - Flatten layer
  - Dropout layer
  - Dense layer
  - Model Compiling
  - Model Summary

```
In [38]: NUM_NURON = 1

model = Sequential()
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(NUM_NURON, activation='sigmoid'))

model.layers[0].trainable = False

model.compile(
    loss='binary_crossentropy',
    optimizer=RMSprop(lr=1e-4),
    metrics=['accuracy']
)

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (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
_________________________________________________________________
```

4. Model Building- cont.

- **Second Step (CNN- FFN)- cont.-**
  - Model Fit
    - Epochs- 30
    - Steps per epochs- 6
    - Validation steps- 3,
    - Early Stopping

```
In [47]: EPOCHS = 30
         es = EarlyStopping(
             monitor='val_accuracy',
             mode='max',
             patience=6
         )

         history = model.fit_generator(
             train_generator,
             steps_per_epoch=6,
             epochs=EPOCHS,
             validation_data=validation_generator,
             validation_steps=3,
             callbacks=[es]
         )
```

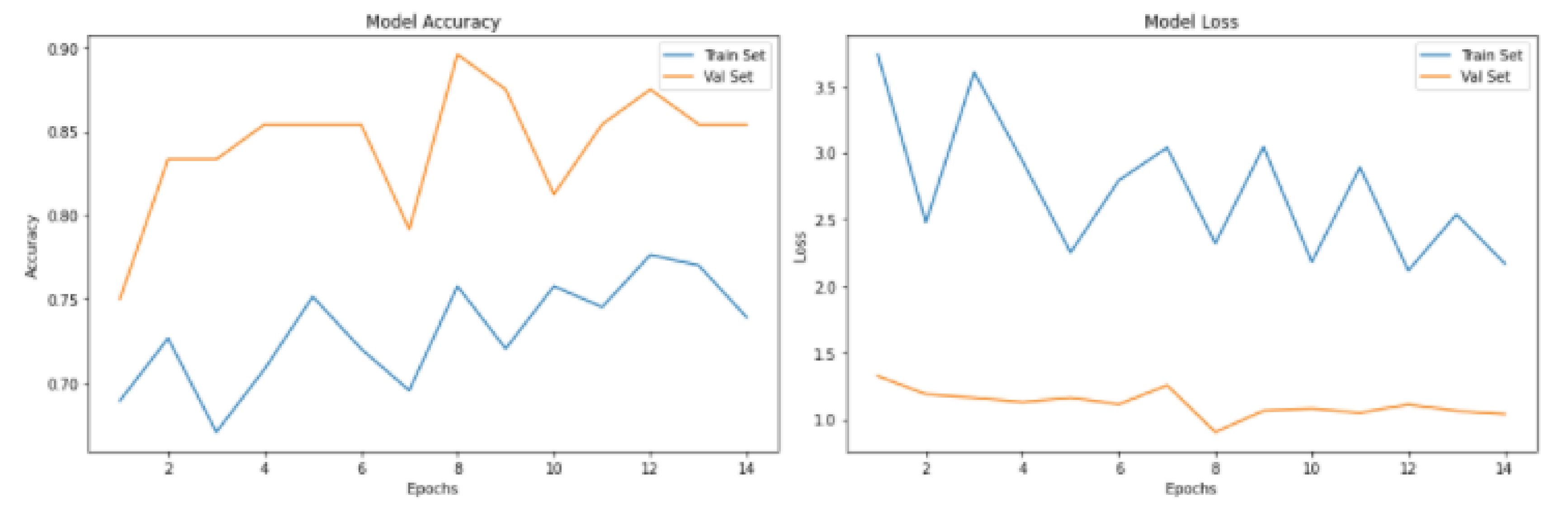
```
Epoch 1/30
6/6 [==============================] - 62s 10s/step - loss: 3.7390 - accuracy: 0.6894 - val_loss: 1.3249 - val_accuracy: 0.7500
Epoch 2/30
6/6 [==============================] - 69s 12s/step - loss: 2.4777 - accuracy: 0.7267 - val_loss: 1.1888 - val_accuracy: 0.8333
Epoch 3/30
6/6 [==============================] - 72s 12s/step - loss: 3.6052 - accuracy: 0.6708 - val_loss: 1.1622 - val_accuracy: 0.8333
Epoch 4/30
6/6 [==============================] - 82s 14s/step - loss: 2.9390 - accuracy: 0.7083 - val_loss: 1.1284 - val_accuracy: 0.8542
Epoch 5/30
6/6 [==============================] - 67s 11s/step - loss: 2.2526 - accuracy: 0.7516 - val_loss: 1.1618 - val_accuracy: 0.8542
Epoch 6/30
6/6 [==============================] - 71s 12s/step - loss: 2.7945 - accuracy: 0.7205 - val_loss: 1.1129 - val_accuracy: 0.8542
Epoch 7/30
6/6 [==============================] - 69s 12s/step - loss: 3.0394 - accuracy: 0.6957 - val_loss: 1.2556 - val_accuracy: 0.7917
Epoch 8/30
6/6 [==============================] - 61s 10s/step - loss: 2.3205 - accuracy: 0.7578 - val_loss: 0.9040 - val_accuracy: 0.8958
Epoch 9/30
6/6 [==============================] - 61s 10s/step - loss: 3.0458 - accuracy: 0.7205 - val_loss: 1.0647 - val_accuracy: 0.8750
Epoch 10/30
6/6 [==============================] - 61s 10s/step - loss: 2.1817 - accuracy: 0.7578 - val_loss: 1.0796 - val_accuracy: 0.8125
Epoch 11/30
6/6 [==============================] - 63s 12s/step - loss: 2.8925 - accuracy: 0.7453 - val_loss: 1.0494 - val_accuracy: 0.8542
Epoch 12/30
6/6 [==============================] - 63s 11s/step - loss: 2.1157 - accuracy: 0.7764 - val_loss: 1.1112 - val_accuracy: 0.8750
Epoch 13/30
6/6 [==============================] - 66s 11s/step - loss: 2.5382 - accuracy: 0.7702 - val_loss: 1.0636 - val_accuracy: 0.8542
Epoch 14/30
6/6 [==============================] - 63s 11s/step - loss: 2.1697 - accuracy: 0.7391 - val_loss: 1.0387 - val_accuracy: 0.8542
```

5. Results

- In this phase,
  - We have checked the performance of the model based on Model Accuracy and Loss.
  - Then, we checked from confusion matrix on validation and test set.

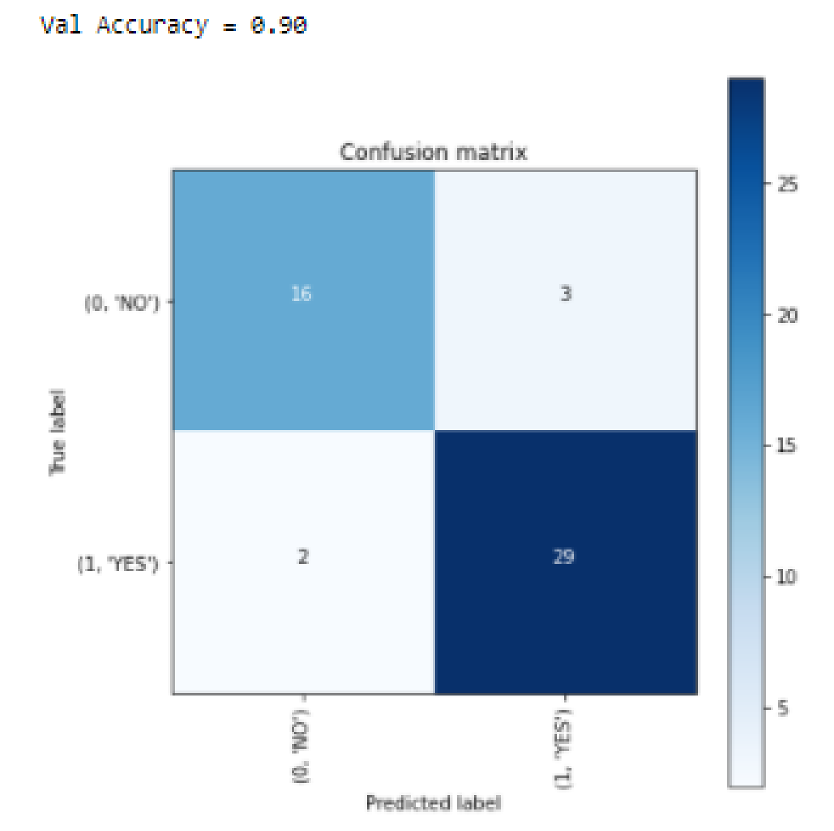
5. Results- cont.

- **Accuracy & Loss Graphs-**



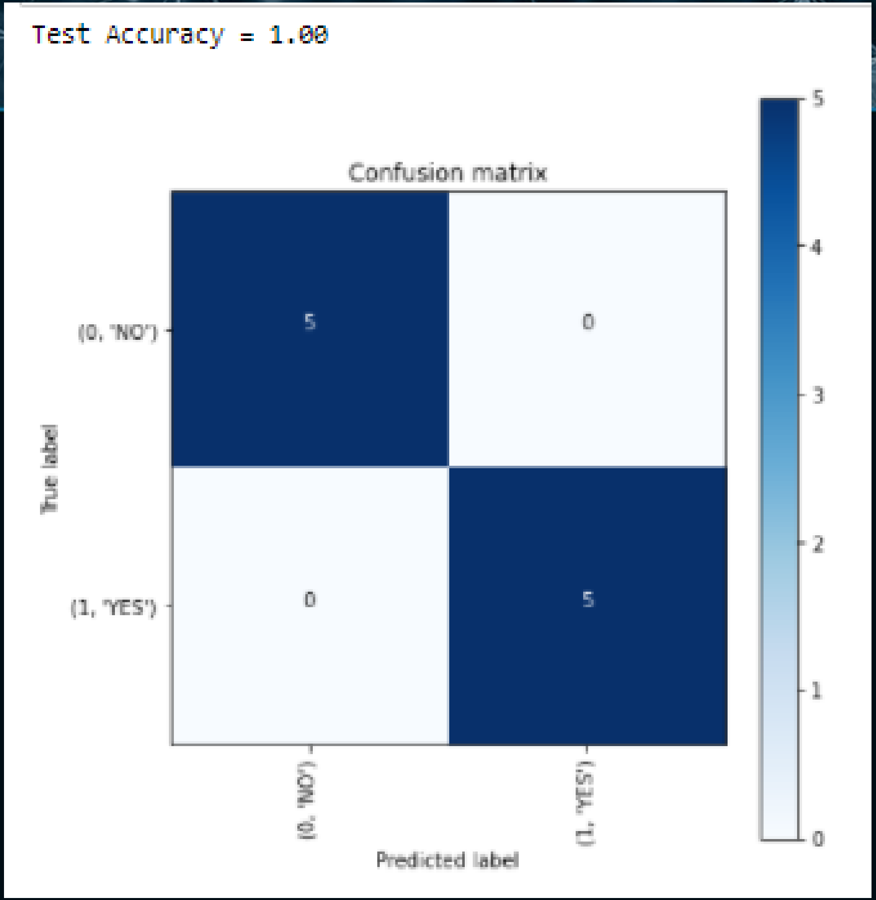
5. Results- cont.

- **Confusion Matrixes-**
  - Validate on validation set- Validation Accuracy = 90%
  - 50 Samples
  - FP=3, FN=2, TN=16 and TP=29.



5. Results- cont.

- **Confusion Matrixes- cont.**
  - Validate on test set- Test Accuracy = 100%
  - 10 Samples
  - FP and FN are 0. TN and TP are classified as 5.



5. Results- cont.

- **Checking misclassification in test set**
  - Test Accuracy is 100%. So, there is no misclassification.

```
In [51]: ind_list = np.argwhere((y_test == predictions) == False)[:, -1]
         if ind_list.size == 0:
             print('There are no missclassified images.')
         else:
             for i in ind_list:
                 plt.figure()
                 plt.imshow(X_test_crop[i])
                 plt.xticks([])
                 plt.yticks([])
                 plt.title(f'Actual class: {y_val[i]}\nPredicted class: {predictions[i]}')
                 plt.show()

         There are no missclassified images.
```

5. Results- cont.

- **Saving the Model**
  - We can reuse this model from these weights.

```
In [52]: # save the model
         model.save('2022-03-07-1pm_VGG_model.h5')
```

5. Results- cont.

- **Vital Findings from these Results**
  - Accuracy of Validation higher than training data.
  - Loss of Validation lesser than training data.
  - Validation Accuracy is 90%. But, test accuracy is 100%. So, Unseen data can predict more accurate.
  - When, we automate this model with MRI Scan images it will predict most probably right decision which classify correctly the patient is cancer or not.
  - This model will improve business (Hospital) value by reducing the wasting time for manual process and don’t need human intervention.

6. Measuring of effeteness

- This Model is more generalized to unseen data than seen data.
- We can integrate this model with MRI scan and when the patient taking the Scan itself it will predict the automatically, which the patient has cancer or not.
- Then, Patients(Customers) satisfaction will increase towards the hospital(Business). They don’t need to waste another time for get report.

6. Measuring of effeteness

- Patients can save their time and cost.
- If it is urgent(cancer), They can immediately take any actions rather than waiting many days for reports. Then, we can save many lives.
- Instead of Doctors are wasting their precious time on viewing Scans as a manual process; they can spend their time for other human involving treatments. This model can automate this task.
- This model will improve business (Hospital) value by reducing the time and cost of physicians and patients. So, Hospitals will attract more patients.
- Code GitHub Repository- https://github.com/Farha-Imthiyaz/Tumor_Classification


THANK
YOU