BRAIN TUMORS DETECTION

by

D.M.K.M. Dissanayake

dmkasun22@gmail.com

A project report submitted in partial fulfillment of the requirements for the

Deep Learning Reinforcement Learning program conducted by

IBM Skills Network

INTRODUCTION

An intracranial tumor also referred to as a brain tumor, is an abnormal mass of tissue where cells live and spread out of control, appearing to be unaffected by the systems that regulate normal cells. It is regarded as one of the more aggressive illnesses that can affect both children and adults. In 2022 alone an estimated 72,360 adults age 40+ will be diagnosed with a primary brain tumor in the U.S. alone. Magnetic resonance imaging (MRI) is typically the first step in the diagnosis of a brain tumor (MRI). The most typical method of identifying the type of brain tumor when an MRI reveals the presence of a tumor in the brain is to analyze the results from a sample of tissue after a biopsy or surgery. Because of the complexity of brain tumors and their characteristics, a manual examination can be prone to inaccuracy. Machine learning and artificial intelligence (ML/AI)-based automated classification systems have consistently outperformed manual categorization in terms of accuracy. Therefore, from this report, a solution will be presented to detect and classify brain tumors from a given image using Deep Learning Algorithms such as Convolutional Neural Networks and Transfer Learning.

DATASET USED

The dataset that was used in this project provides hundreds of images taken from MRI machines to diagnose brain tumors to build a robust and capable deep-learning solution to binary classify whether the observed image provides enough evidence to diagnose it as a possible brain tumor. Data augmentation was used to generate more data from the given dataset in order to train the model more accurately and produce more precise results. Using ImageDataGenerator library from Keras, it was possible to generate a total of 2668 images producing more ammunition to build a better deep-learning solution. Rather than just copying the same image over and over ImageDataGenerator produce multiple instances of an image altering its rotation, width shift, height shift, and other properties like brightness range making it a sufficient data augmentation module.

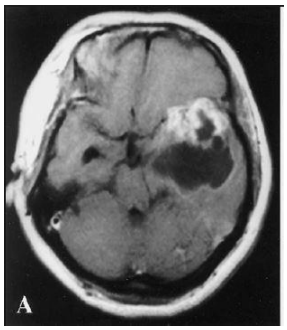

Figure 1. Tumor Present

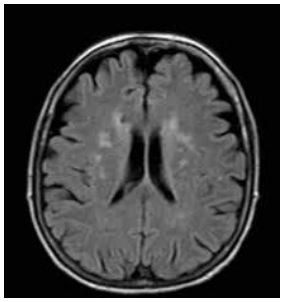

Figure 2. Tumor Absent

As you can see from Figure 1 and Figure 2, all of the image data used to train the models of this project are labeled whether a tumor is present or not based on the prior diagnosis. Because of this nature, the dataset presents only 2 classes, which are 1. Yes, a tumor is present, and 2. No, there is no tumor is present. After the data augmentation as you can see from Figure 3, there are a total of 1273 images exist in the pre-labeled class 'No' and a total of 1395 images exist in the class of 'yes'.

```
[7]
    num_of_images_in_a_class = ["{}: {}".format(key, aug_images_dic[key]) for key in aug_images_dic]
    print(num_of_images_in_a_class)

... ['no: 1273', 'yes: 1395']
```

Figure 3. Samples for each class

OBJECTIVE

As the main objective of this project, using a dataset of MRI brain images, a more robust and capable machine learning model will be selected from a set of deep learning approaches which use convolutional neural networks, evaluating and analyzing key parameters in order to provide a helping hand to brain surgeons around the world in diagnosing brain tumors.

More importantly, the project will analyze and demonstrate how different approaches and techniques can improve or decrease the capability of deep learning models trained on these kinds of datasets.

EXPLORATORY DATA ANALYSIS AND FEATURE ENGINEERING

As depicted in Figure 4 the original dataset only provides us with a total of 253 images of MRI scanned images, containing 155 images for class Yes and 98 images for class No. Therefore, in order to train our deep learning models better, the original images were subjected to a data augmentation operation which produced several variations of the existing dataset. Variations include a change in slight rotations, slight shifts in horizontal and vertical directions, and different brightness conditions.

```
original_datasize_yes = len(os.listdir(image_dir+'/yes'))
original_datasize_no = len(os.listdir(image_dir+'/no'))

print(f'Original number of images for yes class {original_datasize_yes}')
print(f'Original number of images for no class {original_datasize_no}')
```

✓ 0.7s Python

```
Original number of images for yes class 155
Original number of images for no class 98
```

Figure 4. Samples for each class

As you can see from Figure 5, numerous properties have been defined in the operation of augmenting the data in order to generate the new batch of the data set. From the augmentation process, a total of 2668 images are now included in the dataset as depicted in Figure 3.

```
def augment_data(file_dir, no_generated_samples, save_to_dir):
    data_gen = ImageDataGenerator(rotation_range=8,
                                  width_shift_range=0.1,
                                  height_shift_range=0.1,
                                  shear_range=0.1,
                                  brightness_range=(0.3, 1.0),
                                  horizontal_flip=True,
                                  vertical_flip=True,
                                  fill_mode='nearest'
                                 )
```

Figure 5. Data Augmentation Properties

Even though the MRI images in the dataset are most likely captured from similar environments, the dimensionality of each image differs vastly. In order to effectively feed those images in the dataset into corresponding models, the operation of resizing images needed to be done. Specifically, a single height value and a single width value needed to be defined for each and every image. Furthermore, in order to obtain better and more accurate results, most of the machine learning models demand normalization of the data. As you can see from Figure 6, the pixel values of images were subjected to this normalization process as required.

```
image = cv2.imread(directory+'/'+filename)
image = cv2.resize(image, dsize=(image_width, image_height), interpolation=cv2.INTER_CUBIC)
# normalize values
image = image / 255.
print(image)
```

Figure 6. Resizing and Normalizing Images

Before feeding into any deep learning models, the entire augmented data set need to be split into three main parts. They are called training dataset, validation dataset, and test dataset. Here only the training dataset is used to train models and the validation dataset is used to check the quality metrics in every epoch and the test data set is used to check the quality metrics of predictions after the models are fully trained. Following Figure 7 shows the figures for each slice in terms of images. The total image dataset is first split into two slices, training and validation as 7 to 3 and then the validation dataset has again split into validation and tests evenly.

```
number of training examples = 1867
number of validation examples = 401
number of test examples = 400
```

Figure 7. Number of Train, Validation, and Test Data

Following Figure 8 shows the final input shape of the data that is going to feed into the neural network. It has 240 pixels high, 240 pixels wide 1867 RGB images as the input.

```
    print(X_train.shape)
[15] ✓ 0.4s                                                    Python
...  (1867, 240, 240, 3)
```

Figure 8. Input Shape

MACHINE LEARNING ANALYSIS

In this analysis report, three deep learning models are discussed and analyzed. The details about those deep learning models and results are presented analytically.

1. *First Sequential Model*

The first deep learning model that was tested is a neural network consisting of 2 convolutional layers followed by 2 max-pooling layers and 1 fully connected dense layer with a Sigmoid activation function. The sigmoid activation function is used due to the fact that this is a binary classification problem. The model was built using the Keras Sequential library and more details about this first model can be found on the following Figure 9.

Building the Model (using keras sequential)

```
model_1 = Sequential()

model_1.add(Conv2D(32, (5,5), strides = (1,1), padding='valid', input_shape=X_train.shape[1:], activation='relu'))
model_1.add(MaxPooling2D(pool_size=(2, 2)))
model_1.add(Dropout(0.2))

model_1.add(Conv2D(16, (3,3), strides = (2,2), activation='relu'))

model_1.add(MaxPooling2D(pool_size=(2, 2)))
model_1.add(Dropout(0.2))

model_1.add(Flatten())
model_1.add(Dense(1, activation='sigmoid'))

model_1.summary()
```

Figure 9. 1st Sequential Model

It can be further observed that through the following Figure 10, there was a total of 20,513 parameters within the neural network and all of them were trainable parameters.

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 236, 236, 32)      2432

 max_pooling2d_8 (MaxPooling  (None, 118, 118, 32)     0
 2D)

 dropout_8 (Dropout)         (None, 118, 118, 32)      0

 conv2d_9 (Conv2D)           (None, 58, 58, 16)        4624

 max_pooling2d_9 (MaxPooling  (None, 29, 29, 16)       0
 2D)

 dropout_9 (Dropout)         (None, 29, 29, 16)        0

 flatten_4 (Flatten)         (None, 13456)             0

 dense_4 (Dense)             (None, 1)                 13457

=================================================================
Total params: 20,513
Trainable params: 20,513
Non-trainable params: 0
_________________________________________________________________
```

Figure 11. Model Summary

Then the model was trained using the ‘Adam’ optimizer and ‘Binary Cross Entropy’ as the loss function in order to measure the accuracy. Mini batch gradient descent was used in back-propagation feeding 32 sample batches to the model each time. The whole data set was fed 16 times (16 epochs) to the model in order to train it.

As shown in Figure 12, the model resulted in a final training accuracy of 0.97 and a final training loss of 0.079, which is an extremely good performance on the training dataset. Although, the final accuracy on the validation dataset was 0.779 and the final validation loss was 0.913, which was good but not that great against the validation dataset.

```
Epoch 1/16
59/59 [==============================] - 25s 405ms/step - loss: 0.6136 - accuracy: 0.6845 - val_loss: 0.5501 - val_accuracy: 0.7307
Epoch 2/16
59/59 [==============================] - 24s 402ms/step - loss: 0.5219 - accuracy: 0.7424 - val_loss: 0.4930 - val_accuracy: 0.7781
Epoch 3/16
59/59 [==============================] - 23s 394ms/step - loss: 0.4630 - accuracy: 0.7777 - val_loss: 0.4719 - val_accuracy: 0.7955
Epoch 4/16
59/59 [==============================] - 23s 396ms/step - loss: 0.4353 - accuracy: 0.7890 - val_loss: 0.4515 - val_accuracy: 0.7731
Epoch 5/16
59/59 [==============================] - 23s 396ms/step - loss: 0.3782 - accuracy: 0.8307 - val_loss: 0.5309 - val_accuracy: 0.7357
Epoch 6/16
59/59 [==============================] - 23s 396ms/step - loss: 0.3420 - accuracy: 0.8490 - val_loss: 0.5386 - val_accuracy: 0.7506
Epoch 7/16
59/59 [==============================] - 24s 408ms/step - loss: 0.3063 - accuracy: 0.8645 - val_loss: 0.5496 - val_accuracy: 0.7681
Epoch 8/16
59/59 [==============================] - 24s 413ms/step - loss: 0.2531 - accuracy: 0.8988 - val_loss: 0.6103 - val_accuracy: 0.7257
Epoch 9/16
59/59 [==============================] - 24s 404ms/step - loss: 0.2168 - accuracy: 0.9127 - val_loss: 0.6068 - val_accuracy: 0.7656
Epoch 10/16
59/59 [==============================] - 24s 401ms/step - loss: 0.2080 - accuracy: 0.9164 - val_loss: 0.5791 - val_accuracy: 0.7606
Epoch 11/16
59/59 [==============================] - 24s 403ms/step - loss: 0.1532 - accuracy: 0.9448 - val_loss: 0.6759 - val_accuracy: 0.7781
Epoch 12/16
59/59 [==============================] - 24s 406ms/step - loss: 0.1287 - accuracy: 0.9486 - val_loss: 0.7032 - val_accuracy: 0.7706
Epoch 13/16
...
Epoch 15/16
59/59 [==============================] - 24s 406ms/step - loss: 0.0726 - accuracy: 0.9791 - val_loss: 0.8888 - val_accuracy: 0.7456
Epoch 16/16
59/59 [==============================] - 24s 405ms/step - loss: 0.0787 - accuracy: 0.9700 - val_loss: 0.9132 - val_accuracy: 0.7781
```

Figure 12. 1st Model Training Result

If you further observe the training results, it is identifiable that training loss continued to drop down while the validation loss decreased until a certain point and then began to gradually increase. Further, it can be observed that when the training accuracy became increased gradually until the very last epoch and resulted in a high value (0.97), the validation accuracy did not increase all the time. The validation accuracy increased rapidly and then kind of saturated in a certain range of values. This is further demonstrated by Figure 13 and Figure 14. All of this information nudges us to the conclusion that this model has overfitted the training dataset. Therefore, in the second model, one of the primary goals was to add regularization to treat this high variance problem.

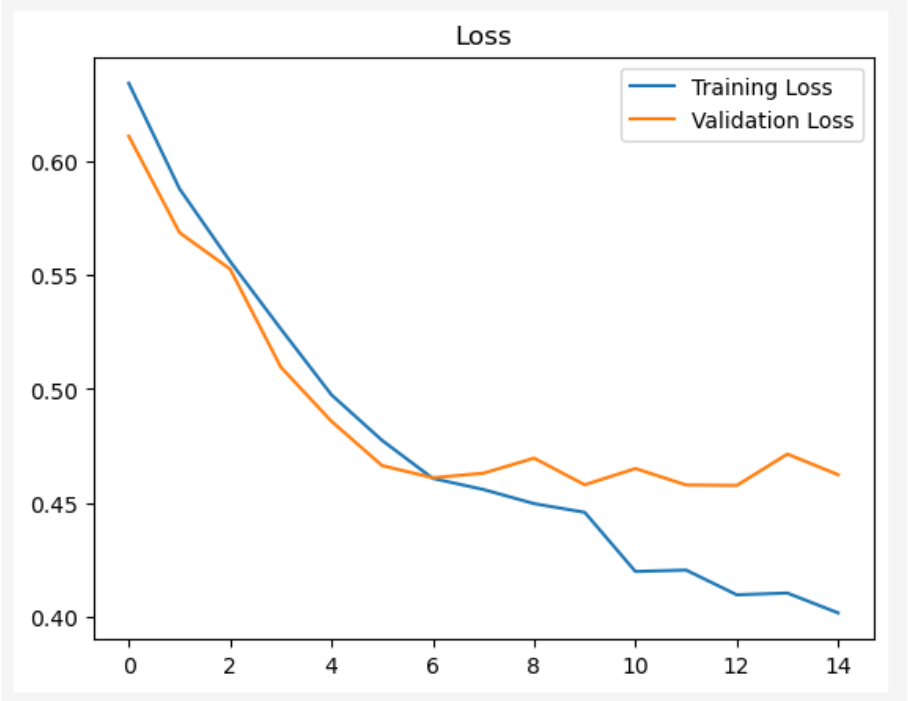

Figure 13. Training Loss and Validation Loss

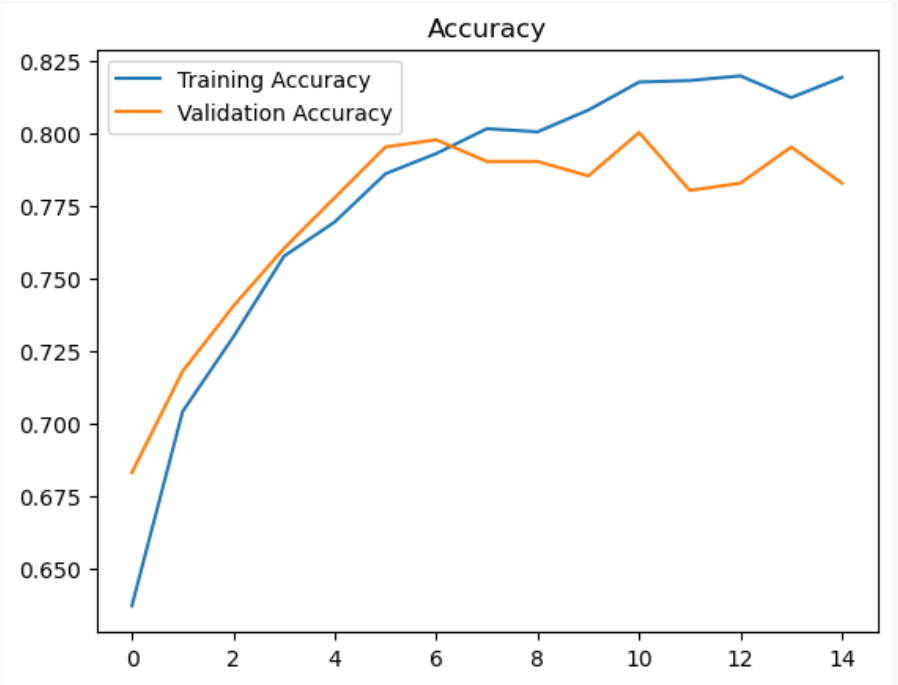

Figure 14. Training Accuracy and Validation Accuracy

2. *Second Sequential Model with L2 Regularization*

The neural network architecture was kept unchanged from the first sequential model and the only change added was the L2 regularization. It is considered a good counter solution to over-fitting. L2 regularization basically punishes high values in training parameters to hold the effect of them training the model so influenced by the training dataset. Following Figure 15 shows the neural network architecture used in the Second Sequential Model with L2 Regularization.

```
model_2 = Sequential()

model_2.add(Conv2D(32, (5,5), strides = (1,1), padding='valid', input_shape=X_train.shape[1:], activation='relu', kernel_regularizer =tf.keras.regularizers.l2( l=0.01)))
model_2.add(MaxPooling2D(pool_size=(2, 2)))
model_2.add(Dropout(0.2))

model_2.add(Conv2D(16, (3,3), strides = (2,2), activation='relu', kernel_regularizer =tf.keras.regularizers.l2( l=0.015)))
model_2.add(MaxPooling2D(pool_size=(2, 2)))
model_2.add(Dropout(0.2))

model_2.add(Flatten())
model_2.add(Dense(1, activation='sigmoid'))

model_2.summary()
```

Figure 15. 2nd Sequential Model Architecture

In terms of trainable parameters for each layer, the Second Sequential Model with L2 Regularization matched with the First Sequential Model as observed in Figure 16. The only difference between the two models is the L2 regularization.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_40 (Conv2D)          (None, 236, 236, 32)      2432

 max_pooling2d_40 (MaxPoolin  (None, 118, 118, 32)     0
 g2D)

 dropout_40 (Dropout)        (None, 118, 118, 32)      0

 conv2d_41 (Conv2D)          (None, 58, 58, 16)        4624

 max_pooling2d_41 (MaxPoolin  (None, 29, 29, 16)       0
 g2D)

 dropout_41 (Dropout)        (None, 29, 29, 16)        0

 flatten_20 (Flatten)        (None, 13456)             0

 dense_18 (Dense)            (None, 1)                 13457

=================================================================
Total params: 20,513
Trainable params: 20,513
Non-trainable params: 0
```

Figure 16. 2nd Sequential Model Summary

The Second Sequential Model was also trained using the same loss function and the optimizer used for the First Sequential Model which is 'Binary Cross Entropy' and 'Adam'. The quality metrics observed are the accuracy of the model as demonstrated in the following Figure 17. The model was trained only for 8 epochs because after 8 epochs it was observed that validation loss getting increased.

```
model_2.compile(loss='binary_crossentropy',
                optimizer='adam',
                metrics=['accuracy'])

model_2.fit(X_train, y_train,
                batch_size=32,
                epochs=8,
                validation_data=(X_val, y_val))
```

Figure 17. Model Training Hyperparameters

As shown in Figure 18, the model resulted in a final training accuracy of 0.83 and a final training loss of 0.41, which is a slightly worse performance on the training dataset compared to the first model. Although, the final accuracy on the validation dataset is 0.79 and the final validation loss was 0.53, which is not extremely good but a better performance compared to the first model.

```
Epoch 1/8
59/59 [==============================] - 25s 409ms/step - loss: 0.8555 - accuracy: 0.6931 - val_loss: 0.7798 - val_accuracy: 0.6808
Epoch 2/8
59/59 [==============================] - 25s 416ms/step - loss: 0.6967 - accuracy: 0.7193 - val_loss: 0.6858 - val_accuracy: 0.7057
Epoch 3/8
59/59 [==============================] - 24s 401ms/step - loss: 0.5957 - accuracy: 0.7493 - val_loss: 0.5734 - val_accuracy: 0.7756
Epoch 4/8
59/59 [==============================] - 24s 401ms/step - loss: 0.5441 - accuracy: 0.7799 - val_loss: 0.5586 - val_accuracy: 0.7681
Epoch 5/8
59/59 [==============================] - 24s 399ms/step - loss: 0.4759 - accuracy: 0.8179 - val_loss: 0.5182 - val_accuracy: 0.7656
Epoch 6/8
59/59 [==============================] - 24s 403ms/step - loss: 0.4553 - accuracy: 0.8141 - val_loss: 0.5426 - val_accuracy: 0.7556
Epoch 7/8
59/59 [==============================] - 24s 409ms/step - loss: 0.4161 - accuracy: 0.8441 - val_loss: 0.5039 - val_accuracy: 0.7905
Epoch 8/8
59/59 [==============================] - 24s 410ms/step - loss: 0.4051 - accuracy: 0.8377 - val_loss: 0.5319 - val_accuracy: 0.7930
```

Figure 18. 2nd Sequential Model with L2 Regularization Result

When comparing to the results of the first model it is observable that L2 regularization treated the over-fitting, bringing down the training accuracy to 0.83 but it has only increased the validation accuracy by 0.02. Therefore, even though the 2nd Sequential model is better than the

first model in that aspect, it is quite evident that things can be improved further. Figure 19 and Figure 20 depict the above-made point.

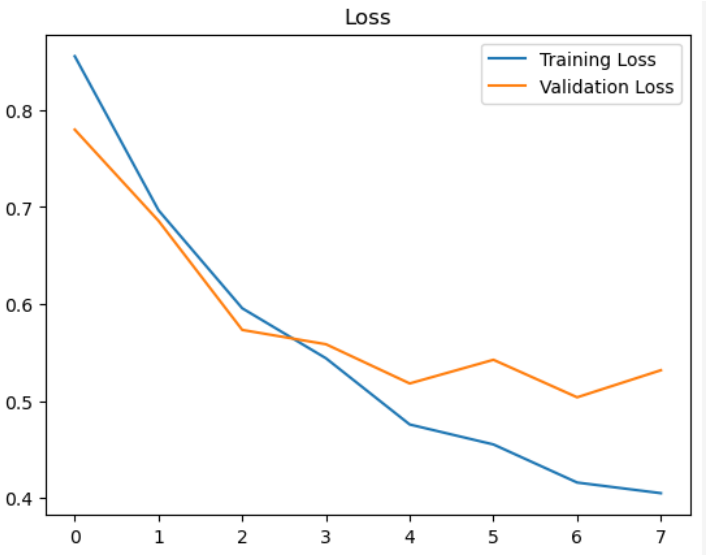

Figure 19. Training and Validation Losses of 2nd Sequential Model

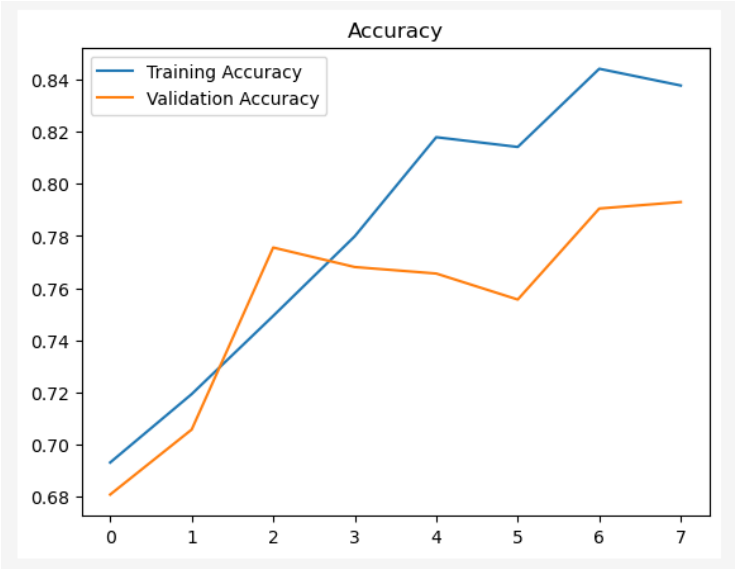

Figure 20. Training and Validation Accuracies of 2nd Sequential Model

3. *Third Sequential Model with Transfer Learning Plus Alterations*

To further improve the results, in this 3rd Sequential model, transfer learning was exercised. Two convolution layers followed by two max pooling layers were identified as the feature layers and a single dense layer is identified as the only classification layer. First, the model was trained as usual for 10 epochs and then those feature layers were frozen. Then the model was trained only on that classification layer, hence the transfer learning. Further alterations were used in order to better the performance of the model and these include increasing the number of channels for the first convolution layer from 32 to 64, increasing the number of channels of the second convolution layer from 16 to 32, and increasing the dropout hyperparameter to 0.2 to 0.5.

In the first run, out of all the parameters that can be trained every one of them are trainable. Although, in the second run, we intentionally freeze all the feature layers, and out of all the parameters, only classification layers are left alone to train. The reason being, since in the first run we have already trained the model with all the parameters and in the second run, we are only going to train classification layers because they are the ones that contribute to deciding the corresponding class of an image as opposed to feature extraction done by feature layers. This method actually aids to remove the vanishing gradient problem that occurs when training deep learning models for a longer number of epochs. Figure 21 presents the feature layers and classification layers.

```
feature_layers = [
    Conv2D(64, (5,5), strides = (2,2), padding='same', input_shape=X_train.shape[1:], activation='relu', kernel_regularizer =tf.keras.regularizers.l2( l=0.01)),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.5),
    Conv2D(32, (3,3), strides = (2,2), activation='relu', kernel_regularizer =tf.keras.regularizers.l2( l=0.01)),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.5),
    Flatten()
]
```

```
classification_layers = [
    Dense(1, activation='sigmoid')
]
```

Figure 21. Feature Layers and Classification Layers

In the results of the first run, we can observe that the model produces a validation accuracy of 0.79 and a training accuracy of 0.82. It further produces a validation loss of 0.49 and a training loss of 0.41. It is again a slight improvement compared to the previous models as we can see from Table 1. Although, the real improvement may come on the second run because that is what transfer learning is applied to. The results of the first run are depicted in the following Figure 23.

```
Epoch 1/10
59/59 [==============================] - 7s 108ms/step - loss: 0.4279 - accuracy: 0.8249 - val_loss: 0.4860
- val_accuracy: 0.8055
Epoch 2/10
59/59 [==============================] - 6s 99ms/step - loss: 0.4343 - accuracy: 0.8222 - val_loss: 0.4756 -
val_accuracy: 0.8005
Epoch 3/10
59/59 [==============================] - 6s 97ms/step - loss: 0.4213 - accuracy: 0.8345 - val_loss: 0.4785 -
val_accuracy: 0.8030
Epoch 4/10
59/59 [==============================] - 6s 97ms/step - loss: 0.4352 - accuracy: 0.8163 - val_loss: 0.4698 -
val_accuracy: 0.8030
Epoch 5/10
59/59 [==============================] - 6s 97ms/step - loss: 0.4205 - accuracy: 0.8291 - val_loss: 0.4707 -
val_accuracy: 0.8130
Epoch 6/10
59/59 [==============================] - 6s 96ms/step - loss: 0.4239 - accuracy: 0.8275 - val_loss: 0.4817 -
val_accuracy: 0.7955
Epoch 7/10
59/59 [==============================] - 6s 97ms/step - loss: 0.4259 - accuracy: 0.8179 - val_loss: 0.4869 -
val_accuracy: 0.7955
Epoch 8/10
59/59 [==============================] - 6s 97ms/step - loss: 0.4259 - accuracy: 0.8249 - val_loss: 0.4727 -
val_accuracy: 0.8130
Epoch 9/10
59/59 [==============================] - 6s 96ms/step - loss: 0.4105 - accuracy: 0.8340 - val_loss: 0.4767 -
val_accuracy: 0.7955
Epoch 10/10
59/59 [==============================] - 6s 99ms/step - loss: 0.4125 - accuracy: 0.8222 - val_loss: 0.4985 -
val_accuracy: 0.7930
```

Figure 23. Results of the First Run

Then in order to do transfer learning, feature layers were frozen as depicted in Figure 24. It can be observed from Figure 25 that not all the parameters are eligible for training for the second run.

```
for l in feature_layers:
    l.trainable = False
```

Figure 25. Frozen Feature Layers

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 120, 120, 64)      4864

 max_pooling2d (MaxPooling2D  (None, 60, 60, 64)       0
 )

 dropout (Dropout)           (None, 60, 60, 64)        0

 conv2d_1 (Conv2D)           (None, 29, 29, 32)        18464

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 14, 14, 32)        0

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 1)                 6273

=================================================================
Total params: 29,601
Trainable params: 6,273
Non-trainable params: 23,328
```

Figure 26. Model Summary

As you can see from the model summary, out of 29,601 total parameters, only 6,273 parameters are trainable. It can be also seen that those trainable parameters belong to the only classification dense layer.

For the second run, the same optimizers and loss functions were used in building the model yet only the parameters corresponding to the classification dense layer were trained. The model was trained for 16 epochs. The results showed an increase in validation accuracy to 0.8 and the recorded training accuracy was 0.83. The final validation loss was recorded as 0.47 and the training loss was 0.4. The results are shown in the following Figure 27. Even though this is not a huge improvement compared to other models, the most important quality metric according to my understanding which is validation accuracy, could reach 0.8. It is the highest of all the 3 models. The validation loss also reached 0.47, which was the lowest of all 3 models.

```
Epoch 1/16
59/59 [==============================] - 7s 103ms/step - loss: 0.4159 - accuracy: 0.8297 - val_loss: 0.4818 - val_accuracy: 0.8005
Epoch 2/16
59/59 [==============================] - 6s 97ms/step - loss: 0.4168 - accuracy: 0.8200 - val_loss: 0.4739 - val_accuracy: 0.8005
Epoch 3/16
59/59 [==============================] - 6s 97ms/step - loss: 0.4076 - accuracy: 0.8238 - val_loss: 0.4726 - val_accuracy: 0.8005
Epoch 4/16
59/59 [==============================] - 6s 98ms/step - loss: 0.4137 - accuracy: 0.8377 - val_loss: 0.4716 - val_accuracy: 0.7980
Epoch 5/16
59/59 [==============================] - 6s 96ms/step - loss: 0.4109 - accuracy: 0.8302 - val_loss: 0.4730 - val_accuracy: 0.8005
Epoch 6/16
59/59 [==============================] - 6s 96ms/step - loss: 0.4068 - accuracy: 0.8372 - val_loss: 0.4878 - val_accuracy: 0.7905
Epoch 7/16
59/59 [==============================] - 6s 97ms/step - loss: 0.4162 - accuracy: 0.8302 - val_loss: 0.4982 - val_accuracy: 0.8055
Epoch 8/16
59/59 [==============================] - 6s 98ms/step - loss: 0.4177 - accuracy: 0.8318 - val_loss: 0.4793 - val_accuracy: 0.7955
Epoch 9/16
59/59 [==============================] - 6s 96ms/step - loss: 0.4116 - accuracy: 0.8232 - val_loss: 0.4729 - val_accuracy: 0.7980
Epoch 10/16
59/59 [==============================] - 6s 96ms/step - loss: 0.4176 - accuracy: 0.8265 - val_loss: 0.4791 - val_accuracy: 0.7980
Epoch 11/16
59/59 [==============================] - 6s 94ms/step - loss: 0.4078 - accuracy: 0.8377 - val_loss: 0.4823 - val_accuracy: 0.8005
Epoch 12/16
59/59 [==============================] - 6s 100ms/step - loss: 0.4052 - accuracy: 0.8366 - val_loss: 0.4967 - val_accuracy: 0.8005
Epoch 13/16
59/59 [==============================] - 6s 99ms/step - loss: 0.4088 - accuracy: 0.8291 - val_loss: 0.4751 - val_accuracy: 0.8130
Epoch 14/16
59/59 [==============================] - 6s 95ms/step - loss: 0.4145 - accuracy: 0.8366 - val_loss: 0.4889 - val_accuracy: 0.8005
Epoch 15/16
59/59 [==============================] - 6s 96ms/step - loss: 0.4037 - accuracy: 0.8307 - val_loss: 0.4789 - val_accuracy: 0.8030
Epoch 16/16
59/59 [==============================] - 6s 95ms/step - loss: 0.4000 - accuracy: 0.8334 - val_loss: 0.4773 - val_accuracy: 0.8005
```

Figure 27. Results After Transfer Learning

Figure 28. Loss After Transfer Learning

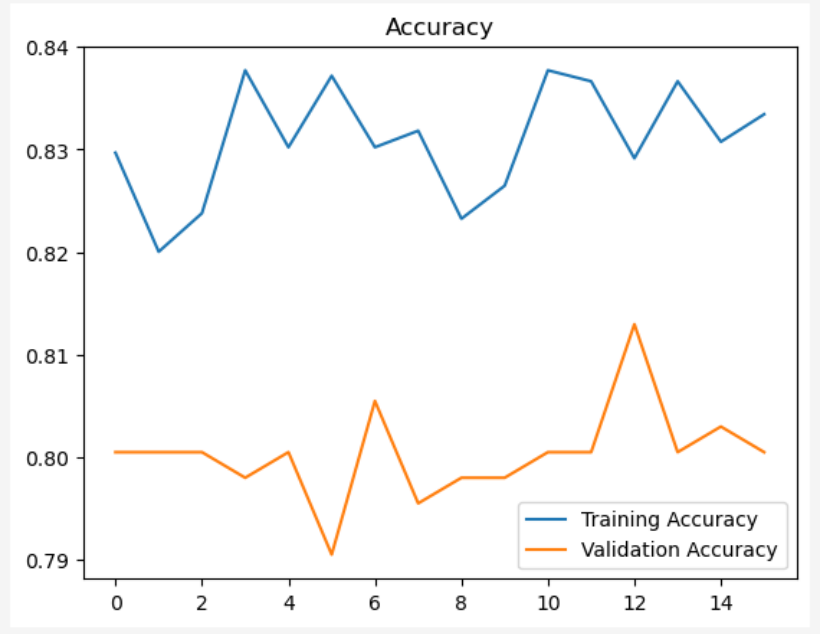

Figure 29. Accuracy After Transfer Learning

As you can observe from Figure 28 and Figure 29, even though it is not a huge improvement from the earlier models, this model after transfer learning resulted in the highest validation accuracy and the lowest validation loss out of all three models. Thus, from all the three trained deep learning models, I recommend this 3rd Sequential Model after Transfer Learning as the best fit for the objective, which is I believe to improve the validation accuracy as much as possible in detecting brain tumors. Following table 1 summarizes all the quality matrices evaluated after training all 3 deep learning models.

Model	Training Loss	Validation Loss	Training Accuracy	Validation Accuracy
1st Seq. Model	0.08	0.91	0.97	0.77
2nd Seq. Model with L2 Reg	0.4	0.53	0.84	0.79
3rd Seq. Model with Transfer Learning	0.4	0.47	0.83	0.8

Table 1. Summary of Model Results

FUTURE IMPROVEMENTS

As presented in this report, even though the model provides higher enough results in quality matrices, there is still room for improvement. It is quite evident that after the first sequential model, there was a major flaw in over-fitting. The training accuracy and the loss seemed to be doing just fine but the validation accuracy and the loss did not. This was a clear indication of the model over-fitting to the dataset. Even though we tried to solve this problem by adding L2 regularization in the second sequential model, the results did not improve significantly. As a future improvement, adding more data to the dataset could help to solve this problem.

Moreover, adding a learning rate reduction can be another improvement as we experienced an increase in our loss after it dropped to a minimum when training the models.

Furthermore, the dataset can further be subjected to more image processing operations in order to maximize the output. For example, these MRI-scanned images contain additional background pixels which are not part of the brain and not related to detecting tumors. A cropping algorithm might be of help to just get the required information only and feed those into the machine learning model.