```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import zipfile
import cv2
from skimage import io
import tensorflow as tf
from tensorflow.python.keras import Sequential
from tensorflow.keras import layers, optimizers
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.initializers import glorot_uniform
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint,
from IPython.display import display
from tensorflow.keras import backend as K
from sklearn.preprocessing import StandardScaler, normalize
import os
import glob
import random
from google.colab import files #library to upload files to colab notebook
%matplotlib inline
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
%cd /content/drive/My Drive/ai/Healthcare AI Datasets/Brain_MRI
```

```
/content/drive/My Drive/ai/Healthcare AI Datasets/Brain_MRI
```

```
brain_df = pd.read_csv('data_mask.csv')
```

```
brain_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3929 entries, 0 to 3928
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   patient_id  3929 non-null   object
 1   image_path  3929 non-null   object
 2   mask_path   3929 non-null   object
```

```
 3   mask        3929 non-null   int64
dtypes: int64(1), object(3)
memory usage: 122.9+ KB
```

```
brain_df.head(50)
```

	patient_id	image_path
0	TCGA_CS_5395_19981004	TCGA_CS_5395_19981004/TCGA_CS_5395_19981004_1.tif
1	TCGA_CS_5395_19981004	TCGA_CS_4944_20010208/TCGA_CS_4944_20010208_1.tif
2	TCGA_CS_5395_19981004	TCGA_CS_4941_19960909/TCGA_CS_4941_19960909_1.tif
3	TCGA_CS_5395_19981004	TCGA_CS_4943_20000902/TCGA_CS_4943_20000902_1.tif
4	TCGA_CS_5395_19981004	TCGA_CS_5396_20010302/TCGA_CS_5396_20010302_1.tif
5	TCGA_CS_5395_19981004	TCGA_CS_5393_19990606/TCGA_CS_5393_19990606_1.tif
6	TCGA_CS_5395_19981004	TCGA_CS_4942_19970222/TCGA_CS_4942_19970222_1.tif
7	TCGA_CS_5395_19981004	TCGA_CS_5397_20010315/TCGA_CS_5397_20010315_1.tif
8	TCGA_CS_5395_19981004	TCGA_CS_6188_20010812/TCGA_CS_6188_20010812_1.tif
9	TCGA_CS_5395_19981004	TCGA_CS_6666_20011109/TCGA_CS_6666_20011109_1.tif
10	TCGA_CS_5395_19981004	TCGA_CS_6669_20020102/TCGA_CS_6669_20020102_1.tif
11	TCGA_CS_5395_19981004	TCGA_CS_6186_20000601/TCGA_CS_6186_20000601_1.tif
12	TCGA_CS_5395_19981004	TCGA_DU_5851_19950428/TCGA_DU_5851_19950428_1.tif
13	TCGA_CS_5395_19981004	TCGA_CS_6665_20010817/TCGA_CS_6665_20010817_1.tif
14	TCGA_CS_5395_19981004	TCGA_CS_6668_20011025/TCGA_CS_6668_20011025_1.tif
15	TCGA_CS_5395_19981004	TCGA_DU_5849_19950405/TCGA_DU_5849_19950405_1.tif
16	TCGA_CS_5395_19981004	TCGA_CS_6290_20000917/TCGA_CS_6290_20000917_1.tif
17	TCGA_CS_5395_19981004	TCGA_DU_5872_19950223/TCGA_DU_5872_19950223_1.tif
18	TCGA_CS_5395_19981004	TCGA_DU_5874_19950510/TCGA_DU_5874_19950510_1.tif
19	TCGA_CS_5395_19981004	TCGA_DU_5855_19951217/TCGA_DU_5855_19951217_1.tif
20	TCGA_CS_4944_20010208	TCGA_DU_5854_19951104/TCGA_DU_5854_19951104_1.tif

```
brain_df.mask_path[1]
```

```
'TCGA_CS_4944_20010208/TCGA_CS_4944_20010208_1_mask.tif'
```

```
brain_df.image_path[1]
```

```
'TCGA_CS_4944_20010208/TCGA_CS_4944_20010208_1.tif'
```

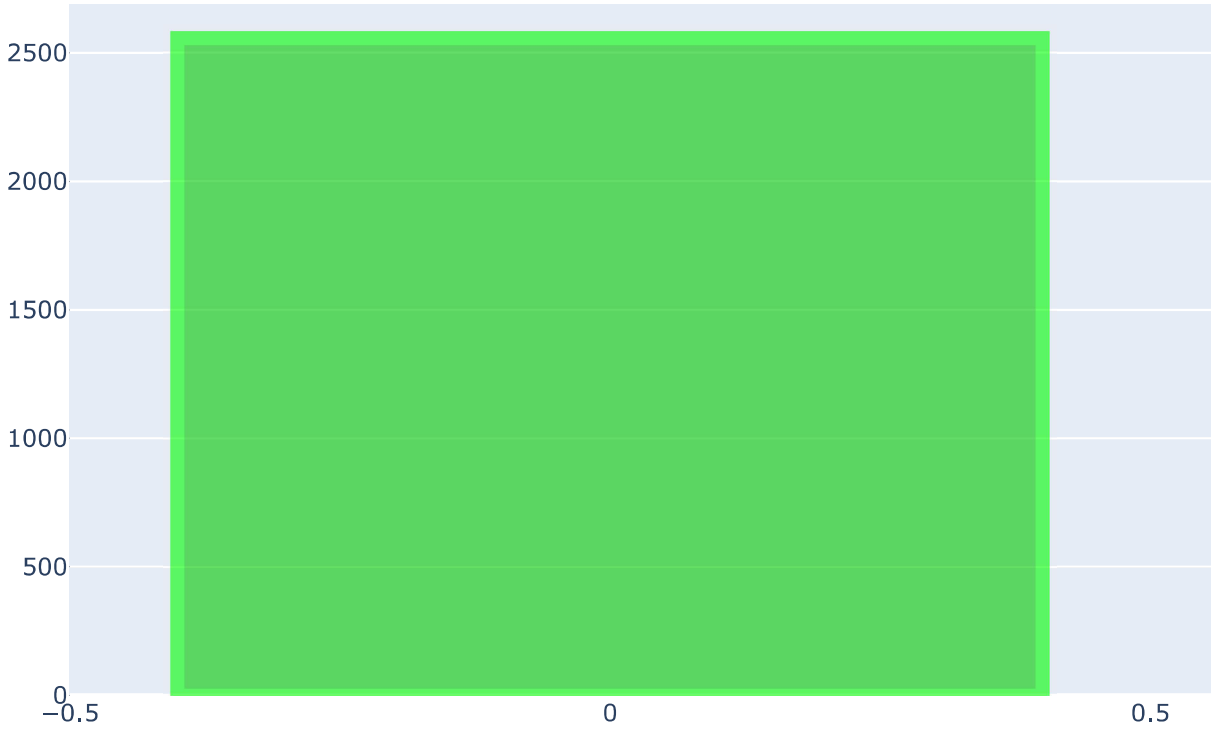
```
brain_df
```

	patient_id	image_path
0	TCGA_CS_5395_19981004	TCGA_CS_5395_19981004/TCGA_CS_5395_19981004_1.tif
1	TCGA_CS_5395_19981004	TCGA_CS_4944_20010208/TCGA_CS_4944_20010208_1.tif
2	TCGA_CS_5395_19981004	TCGA_CS_4941_19960909/TCGA_CS_4941_19960909_1.tif
3	TCGA_CS_5395_19981004	TCGA_CS_4943_20000902/TCGA_CS_4943_20000902_1.tif
4	TCGA_CS_5395_19981004	TCGA_CS_5396_20010302/TCGA_CS_5396_20010302_1.tif
...	...	...
3924	TCGA_DU_6401_19831001	TCGA_HT_A61B_19991127/TCGA_HT_A61B_19991127_86...
3925	TCGA_DU_6401_19831001	TCGA_HT_A61A_20000127/TCGA_HT_A61A_20000127_87...

```
brain_df['mask'].value_counts().index
```

```
Int64Index([0, 1], dtype='int64')
```

```
import plotly.graph_objects as go

fig = go.Figure([go.Bar(x = brain_df['mask'].value_counts().index, y = brain_df['mask'].va
fig.update_traces(marker_color = 'rgb(0,200,0)', marker_line_color = 'rgb(0,255,0)',
                  marker_line_width = 7, opacity = 0.6)
fig.show()
```



```
pip install
```

```
ERROR: You must give at least one requirement to install (see "pip help install")
```

```
brain_df.mask_path
```

```
0       TCGA_CS_5395_19981004/TCGA_CS_5395_19981004_1_...
1       TCGA_CS_4944_20010208/TCGA_CS_4944_20010208_1_...
2       TCGA_CS_4941_19960909/TCGA_CS_4941_19960909_1_...
3       TCGA_CS_4943_20000902/TCGA_CS_4943_20000902_1_...
4       TCGA_CS_5396_20010302/TCGA_CS_5396_20010302_1_...
                              ...
3924    TCGA_HT_A61B_19991127/TCGA_HT_A61B_19991127_86...
3925    TCGA_HT_A61A_20000127/TCGA_HT_A61A_20000127_87...
3926    TCGA_HT_A61B_19991127/TCGA_HT_A61B_19991127_87...
3927    TCGA_HT_A61A_20000127/TCGA_HT_A61A_20000127_88...
3928    TCGA_HT_A61B_19991127/TCGA_HT_A61B_19991127_88...
Name: mask_path, Length: 3929, dtype: object
```

```
brain_df.image_path
```

```
0       TCGA_CS_5395_19981004/TCGA_CS_5395_19981004_1.tif
1       TCGA_CS_4944_20010208/TCGA_CS_4944_20010208_1.tif
2       TCGA_CS_4941_19960909/TCGA_CS_4941_19960909_1.tif
3       TCGA_CS_4943_20000902/TCGA_CS_4943_20000902_1.tif
4       TCGA_CS_5396_20010302/TCGA_CS_5396_20010302_1.tif
                              ...
3924    TCGA_HT_A61B_19991127/TCGA_HT_A61B_19991127_86...
3925    TCGA_HT_A61A_20000127/TCGA_HT_A61A_20000127_87...
3926    TCGA_HT_A61B_19991127/TCGA_HT_A61B_19991127_87...
3927    TCGA_HT_A61A_20000127/TCGA_HT_A61A_20000127_88...
3928    TCGA_HT_A61B_19991127/TCGA_HT_A61B_19991127_88...
Name: image_path, Length: 3929, dtype: object
```

```
brain_df
```

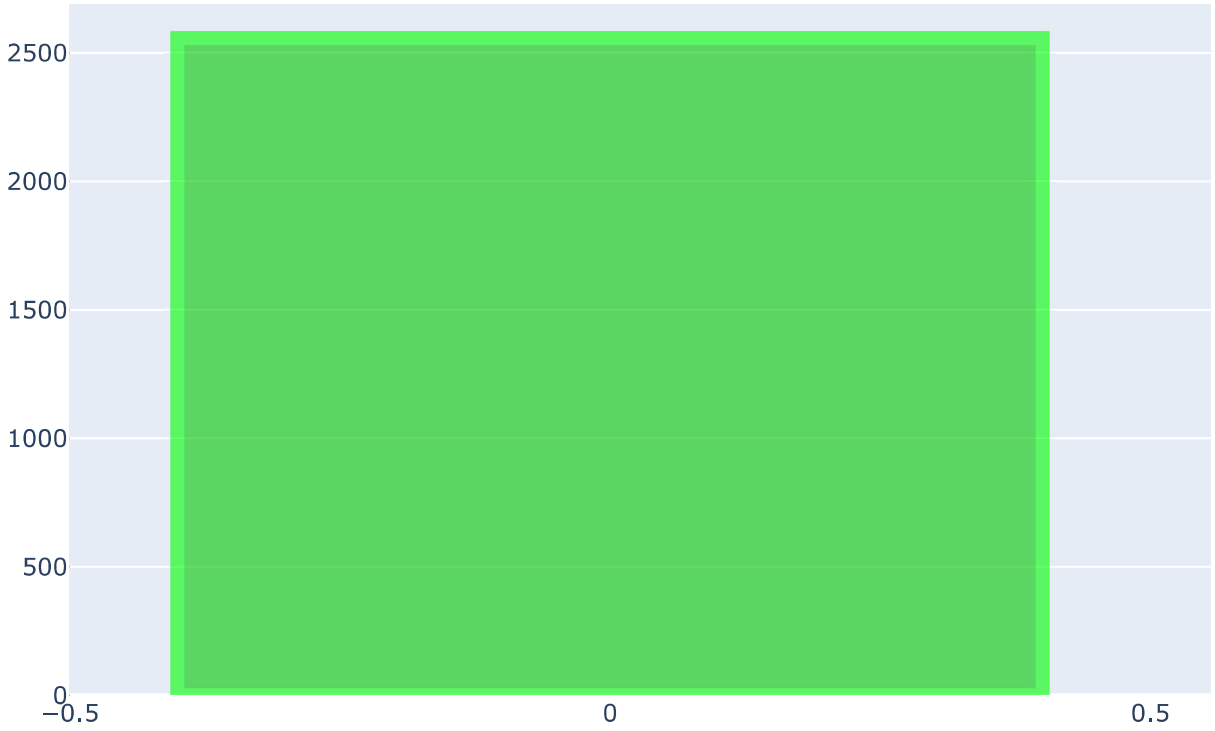
	patient_id	image_path
0	TCGA_CS_5395_19981004	TCGA_CS_5395_19981004/TCGA_CS_5395_19981004_1.tif

```
brain_df['mask'].value_counts().index
```

```
Int64Index([0, 1], dtype='int64')
```

3	TCGA_CS_5395_19981004	TCGA_CS_4943_20000902/TCGA_CS_4943_20000902_1.tif

```
import plotly.graph_objects as go

fig = go.Figure([go.Bar(x = brain_df['mask'].value_counts().index, y = brain_df['mask'].va
fig.update_traces(marker_color = 'rgb(0,200,0)', marker_line_color = 'rgb(0,255,0)',
                  marker_line_width = 7, opacity = 0.6)
fig.show()
```




```
brain_df.mask_path
```

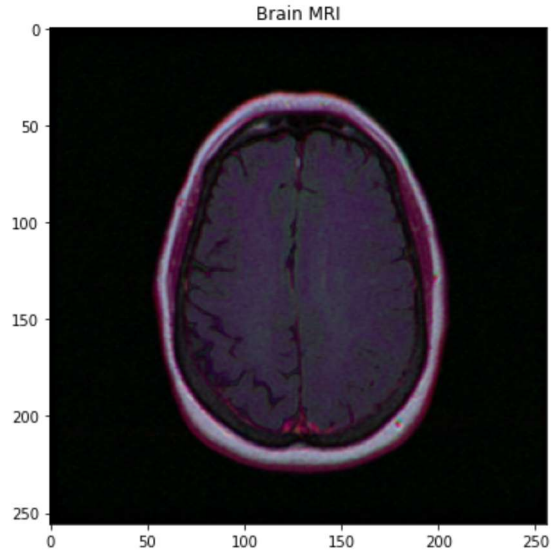
```
0       TCGA_CS_5395_19981004/TCGA_CS_5395_19981004_1_...
1       TCGA_CS_4944_20010208/TCGA_CS_4944_20010208_1_...
2       TCGA_CS_4941_19960909/TCGA_CS_4941_19960909_1_...
3       TCGA_CS_4943_20000902/TCGA_CS_4943_20000902_1_...
4       TCGA_CS_5396_20010302/TCGA_CS_5396_20010302_1_...
                              ...
3924    TCGA_HT_A61B_19991127/TCGA_HT_A61B_19991127_86...
3925    TCGA_HT_A61A_20000127/TCGA_HT_A61A_20000127_87...
3926    TCGA_HT_A61B_19991127/TCGA_HT_A61B_19991127_87...
```

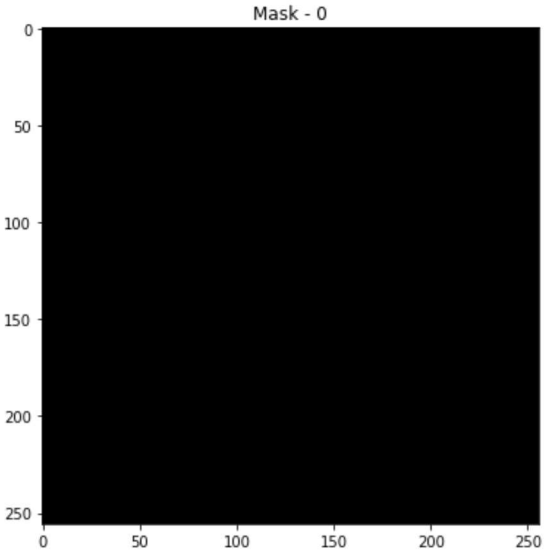
```
3927    TCGA_HT_A61A_20000127/TCGA_HT_A61A_20000127_88...
3928    TCGA_HT_A61B_19991127/TCGA_HT_A61B_19991127_88...
Name: mask_path, Length: 3929, dtype: object
```

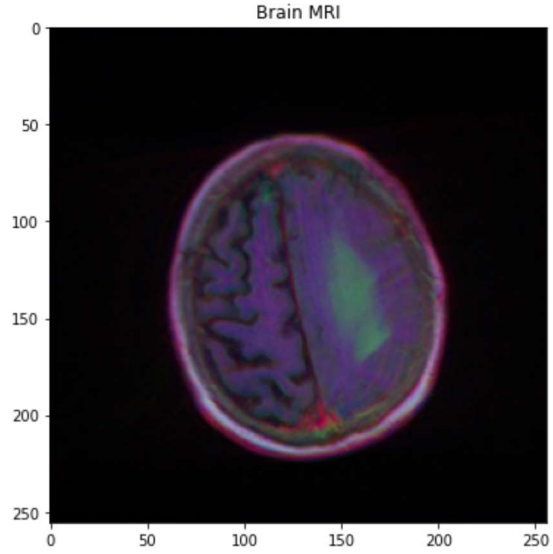
```
brain_df.image_path
```

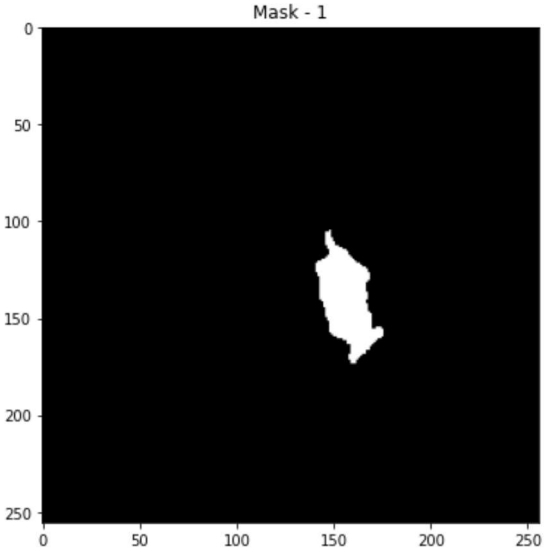
```
0       TCGA_CS_5395_19981004/TCGA_CS_5395_19981004_1.tif
1       TCGA_CS_4944_20010208/TCGA_CS_4944_20010208_1.tif
2       TCGA_CS_4941_19960909/TCGA_CS_4941_19960909_1.tif
3       TCGA_CS_4943_20000902/TCGA_CS_4943_20000902_1.tif
4       TCGA_CS_5396_20010302/TCGA_CS_5396_20010302_1.tif
                              ...
3924    TCGA_HT_A61B_19991127/TCGA_HT_A61B_19991127_86...
3925    TCGA_HT_A61A_20000127/TCGA_HT_A61A_20000127_87...
3926    TCGA_HT_A61B_19991127/TCGA_HT_A61B_19991127_87...
3927    TCGA_HT_A61A_20000127/TCGA_HT_A61A_20000127_88...
3928    TCGA_HT_A61B_19991127/TCGA_HT_A61B_19991127_88...
Name: image_path, Length: 3929, dtype: object
```

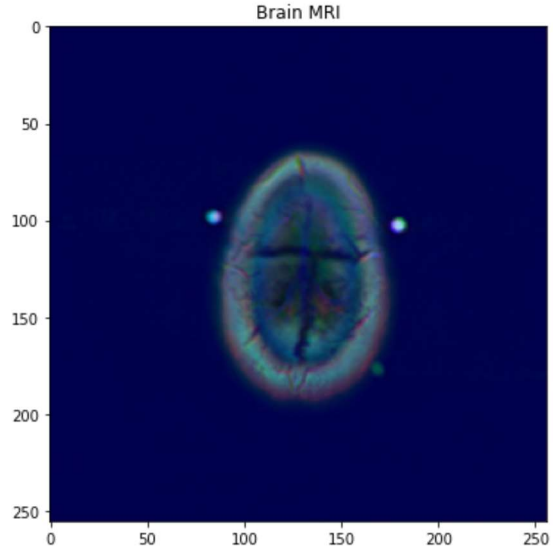
```
import random
fig, axs = plt.subplots(6,2, figsize=(16,32))
count = 0
for x in range(6):
  i = random.randint(0, len(brain_df)) # select a random index
  axs[count][0].title.set_text("Brain MRI") # set title
  axs[count][0].imshow(cv2.imread(brain_df.image_path[i])) # show MRI
  axs[count][1].title.set_text("Mask - " + str(brain_df['mask'][i])) # plot title on the m
  axs[count][1].imshow(cv2.imread(brain_df.mask_path[i])) # Show corresponding mask
  count += 1

fig.tight_layout()
```

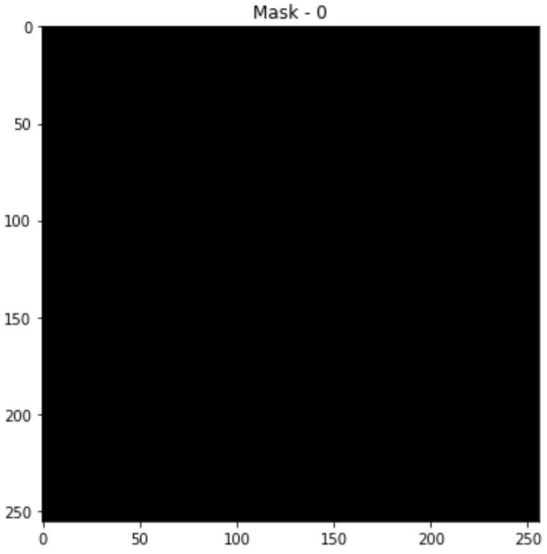

Brain MRI

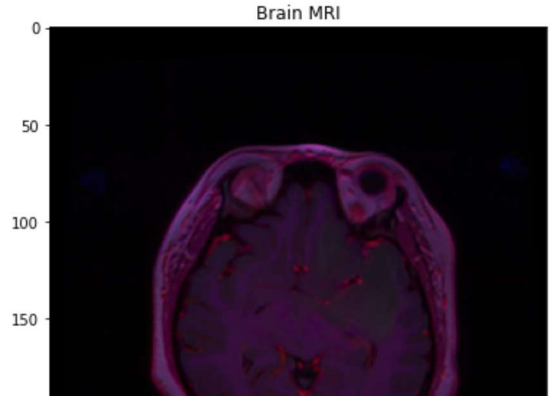

Mask - 0

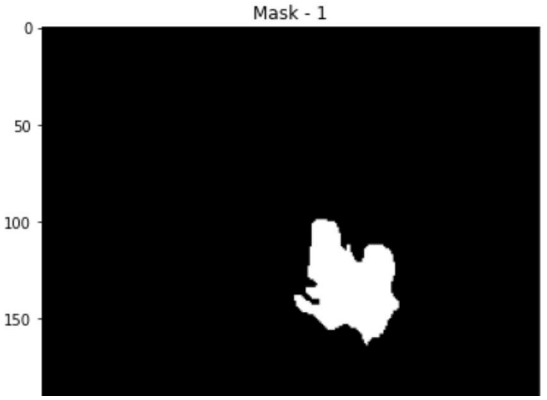

Brain MRI

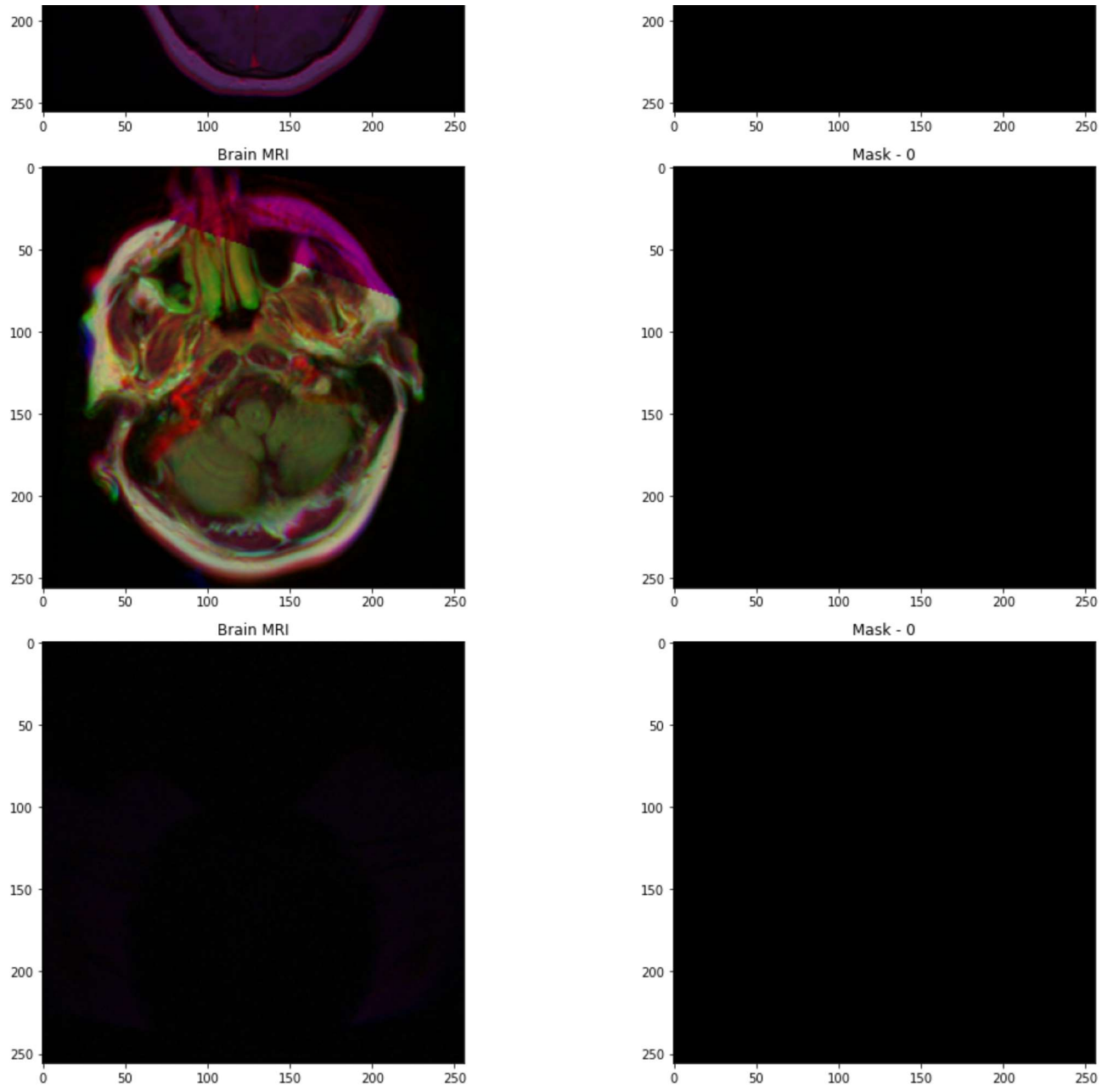

Mask - 1


Brain MRI


Mask - 0

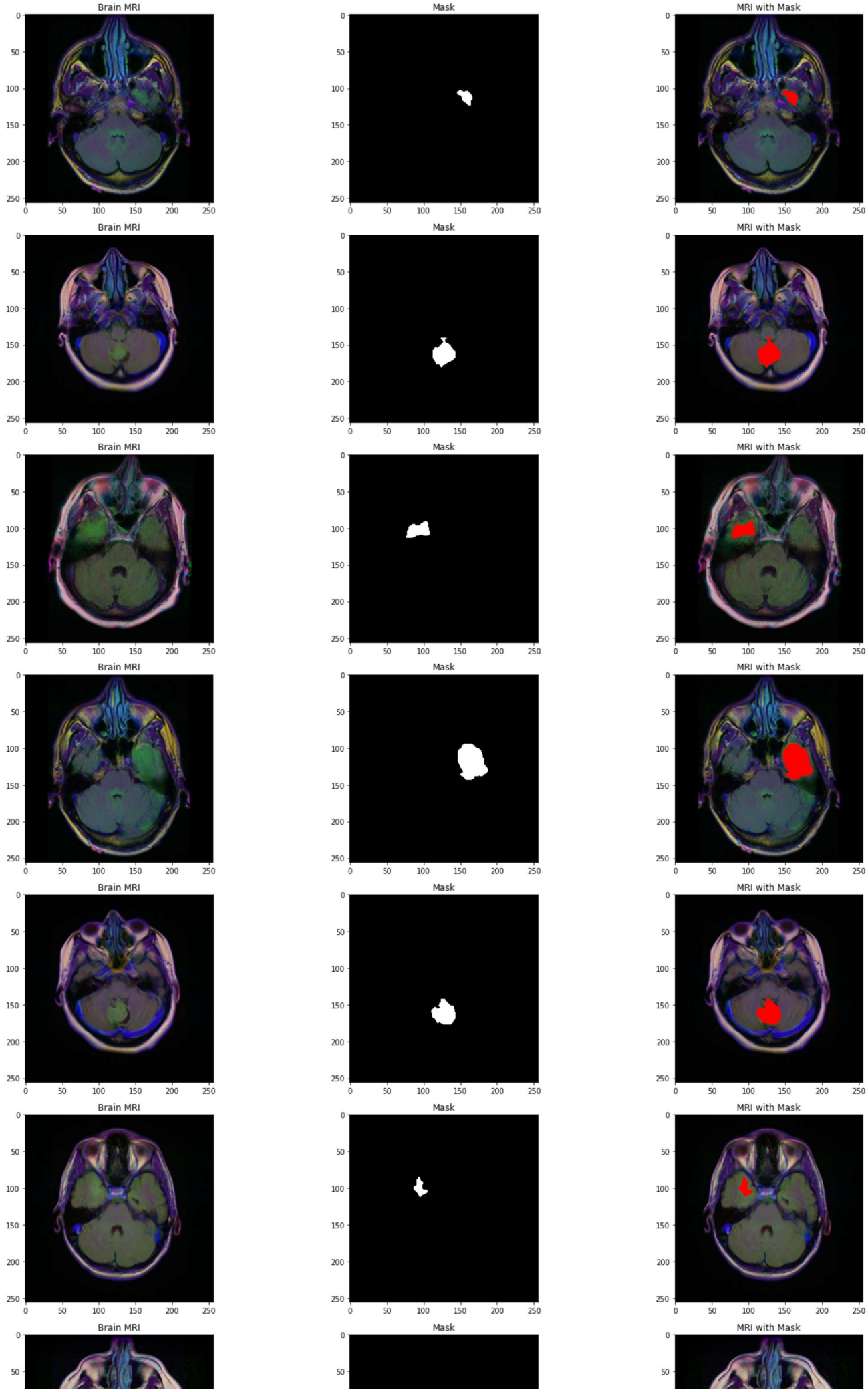

Brain MRI

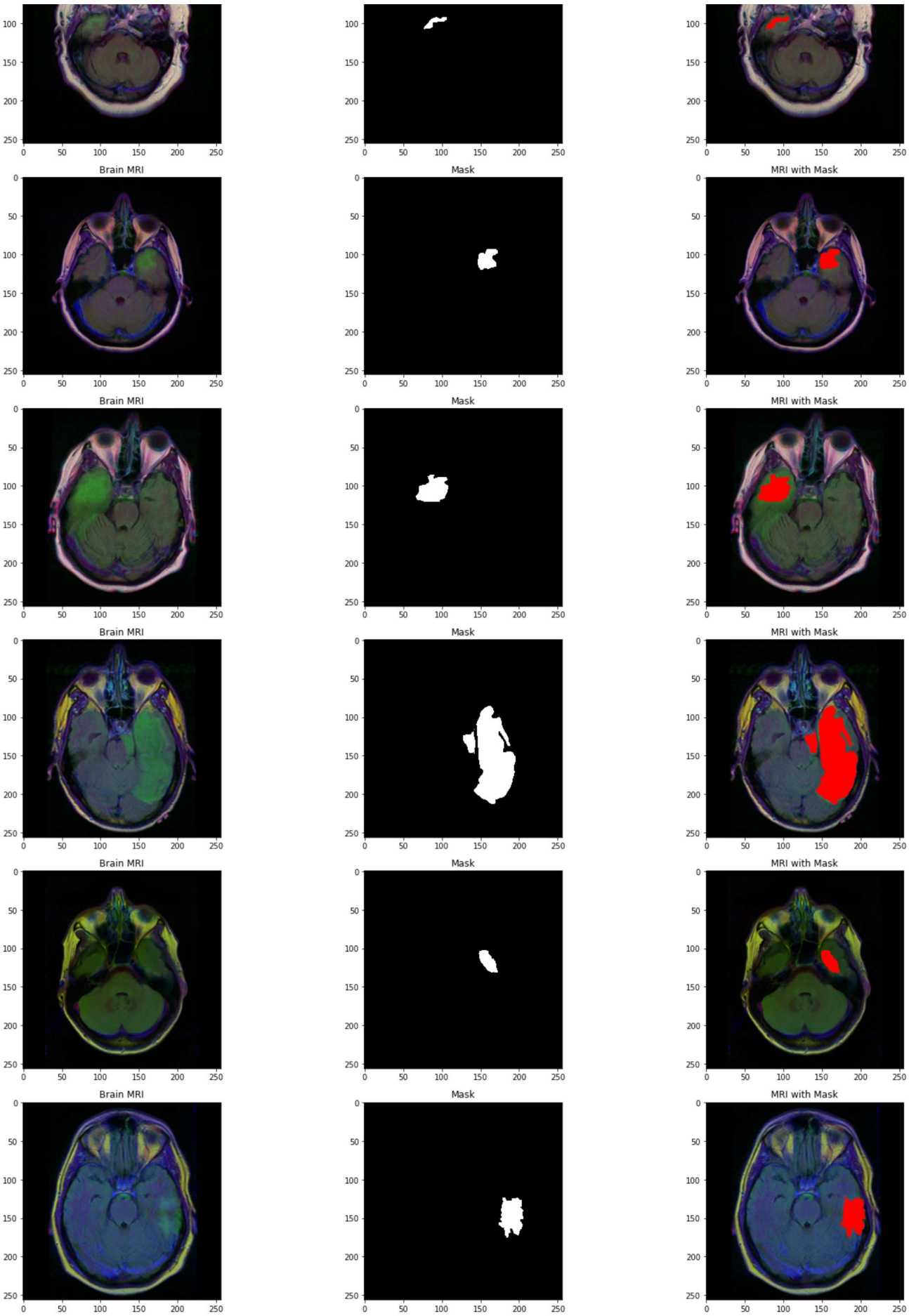

Mask - 1




```
count = 0
fig, axs = plt.subplots(12, 3, figsize = (20, 50))
for i in range(len(brain_df)):
  if brain_df['mask'][i] ==1 and count <12:
    img = io.imread(brain_df.image_path[i])
    axs[count][0].title.set_text('Brain MRI')
    axs[count][0].imshow(img)

    mask = io.imread(brain_df.mask_path[i])
    axs[count][1].title.set_text('Mask')
```

```
    axs[count][1].imshow(mask, cmap = 'gray')


    img[mask == 255] = (255, 0, 0)
    axs[count][2].title.set_text('MRI with Mask')
    axs[count][2].imshow(img)
    count+=1

fig.tight_layout()
```


Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask


Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask
Brain MRI
Mask
MRI with Mask

```
# Drop the patient id column
brain_df_train = brain_df.drop(columns = ['patient_id'])
brain_df_train.shape
```

```
(3929, 3)
```

```
# Convert the data in mask column to string format, to use categorical mode in flow_from_d
# You will get this error message if you comment out the following code line:
# TypeError: If class_mode="categorical", y_col="mask" column values must be type string,
brain_df_train['mask'] = brain_df_train['mask'].apply(lambda x: str(x))
```

```
brain_df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3929 entries, 0 to 3928
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
```

```
 0   image_path  3929 non-null   object
 1   mask_path   3929 non-null   object
 2   mask        3929 non-null   object
dtypes: object(3)
memory usage: 92.2+ KB
```

```
# split the data into train and test data

from sklearn.model_selection import train_test_split

train, test = train_test_split(brain_df_train, test_size = 0.15)
```

```
# create a image generator
from keras_preprocessing.image import ImageDataGenerator

# Create a data generator which scales the data from 0 to 1 and makes validation split of
datagen = ImageDataGenerator(rescale=1./255., validation_split = 0.15)
```

```
train_generator=datagen.flow_from_dataframe(
dataframe=train,
directory= './',
x_col='image_path',
y_col='mask',
subset="training",
batch_size=16,
shuffle=True,
class_mode="categorical",
target_size=(256,256))


valid_generator=datagen.flow_from_dataframe(
dataframe=train,
directory= './',
x_col='image_path',
y_col='mask',
subset="validation",
batch_size=16,
shuffle=True,
class_mode="categorical",
target_size=(256,256))

# Create a data generator for test images
test_datagen=ImageDataGenerator(rescale=1./255.)

test_generator=test_datagen.flow_from_dataframe(
dataframe=test,
directory= './',
x_col='image_path',
y_col='mask',
batch_size=16,
shuffle=False,
class_mode='categorical',
target_size=(256,256))
```

```
Found 2839 validated image filenames belonging to 2 classes.
Found 500 validated image filenames belonging to 2 classes.
Found 590 validated image filenames belonging to 2 classes.
```

```
# Get the ResNet50 base model
basemodel = ResNet50(weights = 'imagenet', include_top = False, input_tensor = Input(shape
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/re
94773248/94765736 [==============================] - 2s 0us/step
```

```
basemodel.summary()
```

```
Model: "resnet50"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 3)	0	
conv1_pad (ZeroPadding2D)	(None, 262, 262, 3)	0	input_1[0][0]
conv1_conv (Conv2D)	(None, 128, 128, 64)	9472	conv1_pad[0][0]
conv1_bn (BatchNormalization)	(None, 128, 128, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 128, 128, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 130, 130, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 64, 64, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 64, 64, 64)	4160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormali	(None, 64, 64, 64)	256	conv2_block1_1_co
conv2_block1_1_relu (Activation	(None, 64, 64, 64)	0	conv2_block1_1_bn
conv2_block1_2_conv (Conv2D)	(None, 64, 64, 64)	36928	conv2_block1_1_re
conv2_block1_2_bn (BatchNormali	(None, 64, 64, 64)	256	conv2_block1_2_co
conv2_block1_2_relu (Activation	(None, 64, 64, 64)	0	conv2_block1_2_bn
conv2_block1_0_conv (Conv2D)	(None, 64, 64, 256)	16640	pool1_pool[0][0]
conv2_block1_3_conv (Conv2D)	(None, 64, 64, 256)	16640	conv2_block1_2_re
conv2_block1_0_bn (BatchNormali	(None, 64, 64, 256)	1024	conv2_block1_0_co
conv2_block1_3_bn (BatchNormali	(None, 64, 64, 256)	1024	conv2_block1_3_co
conv2_block1_add (Add)	(None, 64, 64, 256)	0	conv2_block1_0_bn conv2_block1_3_bn
conv2_block1_out (Activation)	(None, 64, 64, 256)	0	conv2_block1_add[(
conv2_block2_1_conv (Conv2D)	(None, 64, 64, 64)	16448	conv2_block1_out[(

```
conv2_block2_1_bn (BatchNormali (None, 64, 64, 64)   256         conv2_block2_1_co
conv2_block2_1_relu (Activation (None, 64, 64, 64)   0           conv2_block2_1_bn
conv2_block2_2_conv (Conv2D)    (None, 64, 64, 64)   36928       conv2_block2_1_re
conv2_block2_2_bn (BatchNormali (None, 64, 64, 64)   256         conv2_block2_2_co
conv2_block2_2_relu (Activation (None, 64, 64, 64)   0           conv2_block2_2_bn
conv2_block2_3_conv (Conv2D)    (None, 64, 64, 256)  16640       conv2_block2_2_re
conv2_block2_3_bn (BatchNormali (None, 64, 64, 256)  1024        conv2_block2_3_co
```

```
# freeze the model weights

for layer in basemodel.layers:
  layers.trainable = False
```

```
# Add classification head to the base model

headmodel = basemodel.output
headmodel = AveragePooling2D(pool_size = (4,4))(headmodel)
headmodel = Flatten(name= 'flatten')(headmodel)
headmodel = Dense(256, activation = "relu")(headmodel)
headmodel = Dropout(0.3)(headmodel)#
headmodel = Dense(256, activation = "relu")(headmodel)
headmodel = Dropout(0.3)(headmodel)
#headmodel = Dense(256, activation = "relu")(headmodel)
#headmodel = Dropout(0.3)(headmodel)
headmodel = Dense(2, activation = 'softmax')(headmodel)

model = Model(inputs = basemodel.input, outputs = headmodel)
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 256, 256, 3)      0         
_________________________________________________________________
conv1_pad (ZeroPadding2D)    (None, 262, 262, 3)       0         
_________________________________________________________________
conv1_conv (Conv2D)          (None, 128, 128, 64)      9472      
_________________________________________________________________
conv1_bn (BatchNormalization (None, 128, 128, 64)      256       
_________________________________________________________________
conv1_relu (Activation)       (None, 128, 128, 64)      0         
_________________________________________________________________
pool1_pad (ZeroPadding2D)    (None, 130, 130, 64)      0         
_________________________________________________________________
pool1_pool (MaxPooling2D)    (None, 64, 64, 64)        0         
_________________________________________________________________
conv2_block1_1_conv (Conv2D) (None, 64, 64, 64)        4160      
```

conv2_block1_1_bn (BatchNormali	(None, 64, 64, 64)	256	conv2_block1_1_co
conv2_block1_1_relu (Activation	(None, 64, 64, 64)	0	conv2_block1_1_bn
conv2_block1_2_conv (Conv2D)	(None, 64, 64, 64)	36928	conv2_block1_1_re
conv2_block1_2_bn (BatchNormali	(None, 64, 64, 64)	256	conv2_block1_2_co
conv2_block1_2_relu (Activation	(None, 64, 64, 64)	0	conv2_block1_2_bn
conv2_block1_0_conv (Conv2D)	(None, 64, 64, 256)	16640	pool1_pool[0][0]
conv2_block1_3_conv (Conv2D)	(None, 64, 64, 256)	16640	conv2_block1_2_re
conv2_block1_0_bn (BatchNormali	(None, 64, 64, 256)	1024	conv2_block1_0_co
conv2_block1_3_bn (BatchNormali	(None, 64, 64, 256)	1024	conv2_block1_3_co
conv2_block1_add (Add)	(None, 64, 64, 256)	0	conv2_block1_0_bn conv2_block1_3_bn
conv2_block1_out (Activation)	(None, 64, 64, 256)	0	conv2_block1_add[(
conv2_block2_1_conv (Conv2D)	(None, 64, 64, 64)	16448	conv2_block1_out[(
conv2_block2_1_bn (BatchNormali	(None, 64, 64, 64)	256	conv2_block2_1_co
conv2_block2_1_relu (Activation	(None, 64, 64, 64)	0	conv2_block2_1_bn
conv2_block2_2_conv (Conv2D)	(None, 64, 64, 64)	36928	conv2_block2_1_re
conv2_block2_2_bn (BatchNormali	(None, 64, 64, 64)	256	conv2_block2_2_co
conv2_block2_2_relu (Activation	(None, 64, 64, 64)	0	conv2_block2_2_bn
conv2_block2_3_conv (Conv2D)	(None, 64, 64, 256)	16640	conv2_block2_2_re
conv2_block2_3_bn (BatchNormali	(None, 64, 64, 256)	1024	conv2_block2_3_co

```
# compile the model

model.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics= ["accuracy"])
```

```
# use early stopping to exit training if validation loss is not decreasing even after cert
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=20)

# save the best model with least validation loss
checkpointer = ModelCheckpoint(filepath="classifier-resnet-weights.hdf5", verbose=1, save_
```

```
history = model.fit(train_generator, steps_per_epoch= train_generator.n // 16, epochs = 1,
```

```
177/177 [==============================] - 2711s 15s/step - loss: 0.9275 - accuracy:

Epoch 00001: val_loss improved from inf to 2.61816, saving model to classifier-resnet
```

```
# save the model architecture to json file for future use

model_json = model.to_json()
with open("classifier-resnet-model.json","w") as json_file:
  json_file.write(model_json)
```

```
# Load pretrained model (instead of training the model for 1+ hours)
with open('resnet-50-MRI.json', 'r') as json_file:
    json_savedModel= json_file.read()
# load the model
model = tf.keras.models.model_from_json(json_savedModel)
model.load_weights('weights.hdf5')
model.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics= ["accuracy"])
```

```
test_predict = model.predict(test_generator, steps = test_generator.n // 16, verbose =1)
```

```
36/36 [==============================] - 349s 10s/step
```

```
test_predict.shape
```

```
(576, 2)
```

```
test_predict
```

```
array([[1.0000000e+00, 3.2683627e-13],
       [1.0000000e+00, 5.1155008e-10],
       [1.0000000e+00, 3.0363190e-14],
       ...,
       [1.0000000e+00, 9.6832897e-10],
       [9.9999690e-01, 3.0787119e-06],
       [9.9400061e-01, 5.9993262e-03]], dtype=float32)
```

```
# Obtain the predicted class from the model prediction
predict = []

for i in test_predict:
  predict.append(str(np.argmax(i)))

predict = np.asarray(predict)
```

```
predict
```

```
array(['0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '0', '1',
       '0', '0', '1', '1', '1', '0', '1', '1', '0', '1', '1', '0', '0',
       '0', '1', '0', '1', '0', '0', '1', '0', '0', '0', '0', '1', '0',
       '1', '0', '0', '1', '1', '0', '0', '0', '0', '0', '1', '1', '0',
       '1', '0', '1', '0', '1', '0', '1', '1', '0', '1', '0', '1', '0',
       '0', '0', '0', '1', '1', '1', '0', '0', '0', '1', '0', '0', '0',
       '0', '1', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0',
       '0', '0', '0', '1', '1', '1', '0', '1', '1', '1', '0', '0', '1',
       '0', '0', '1', '0', '1', '0', '0', '1', '0', '0', '1', '1', '1',
       '1', '0', '1', '0', '0', '1', '0', '0', '1', '0', '0', '0', '0',
       '0', '0', '1', '0', '1', '0', '0', '0', '0', '1', '0', '1', '0',
```

```
'0', '1', '1', '0', '0', '0', '1', '0', '0', '1', '0', '0', '0',
'0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0',
'1', '1', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'0', '0', '1', '1', '0', '0', '0', '1', '0', '1', '0', '1', '1',
'0', '0', '0', '1', '0', '1', '0', '1', '0', '0', '1', '1', '0',
'0', '0', '0', '0', '1', '0', '0', '1', '0', '0', '0', '0', '0',
'1', '0', '0', '1', '0', '0', '0', '1', '0', '0', '0', '0', '1',
'0', '0', '1', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0',
'1', '0', '1', '0', '1', '0', '1', '0', '1', '1', '0', '0', '0',
'0', '0', '0', '0', '1', '0', '0', '1', '1', '0', '1', '0', '0',
'0', '1', '1', '1', '0', '0', '1', '0', '0', '0', '0', '1', '0',
'1', '0', '0', '0', '1', '1', '1', '0', '0', '0', '1', '1', '0',
'0', '1', '0', '0', '0', '1', '0', '1', '1', '1', '1', '0', '0',
'0', '0', '1', '1', '0', '0', '1', '1', '0', '0', '0', '0', '0',
'0', '1', '1', '0', '0', '0', '1', '0', '1', '0', '0', '1', '0',
'0', '0', '0', '0', '0', '0', '0', '1', '0', '1', '0', '0', '0',
'0', '1', '1', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0',
'0', '0', '1', '1', '0', '0', '0', '0', '1', '0', '1', '1', '0',
'0', '0', '1', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0',
'0', '1', '1', '0', '0', '0', '1', '0', '0', '0', '1', '0', '0',
'1', '0', '0', '0', '0', '1', '0', '0', '1', '1', '1', '0', '0',
'0', '0', '0', '1', '0', '1', '0', '1', '0', '0', '0', '0', '0',
'0', '0', '1', '0', '0', '1', '1', '0', '1', '0', '0', '1', '1',
'0', '0', '1', '0', '0', '0', '0', '0', '0', '1', '0', '0', '1',
'0', '1', '0', '1', '1', '0', '1', '1', '0', '0', '0', '0', '1',
'0', '1', '0', '1', '1', '0', '0', '0', '0', '0', '0', '0', '1',
'0', '0', '0', '0', '1', '0', '0', '1', '1', '1', '0', '0', '1',
'1', '0', '1', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0',
'0', '0', '1', '0', '0', '0', '0', '1', '0', '1', '0', '0', '1',
'0', '1', '1', '1', '0', '1', '1', '1', '0', '1', '0', '0', '0',
'1', '1', '0', '0', '1', '1', '1', '0', '0', '1', '0', '0', '0',
'0', '0', '0', '1', '0', '0', '0', '0', '1', '0', '1', '0', '1',
'0', '0', '1', '1', '0', '0', '0', '1', '0', '0', '1', '0', '0',
'0', '0', '0', '0'], dtype='<U1')
```

```
# since we have used test generator, it limited the images to len(predict), due to batch s
original = np.asarray(test['mask'])[:len(predict)]
len(original)
```

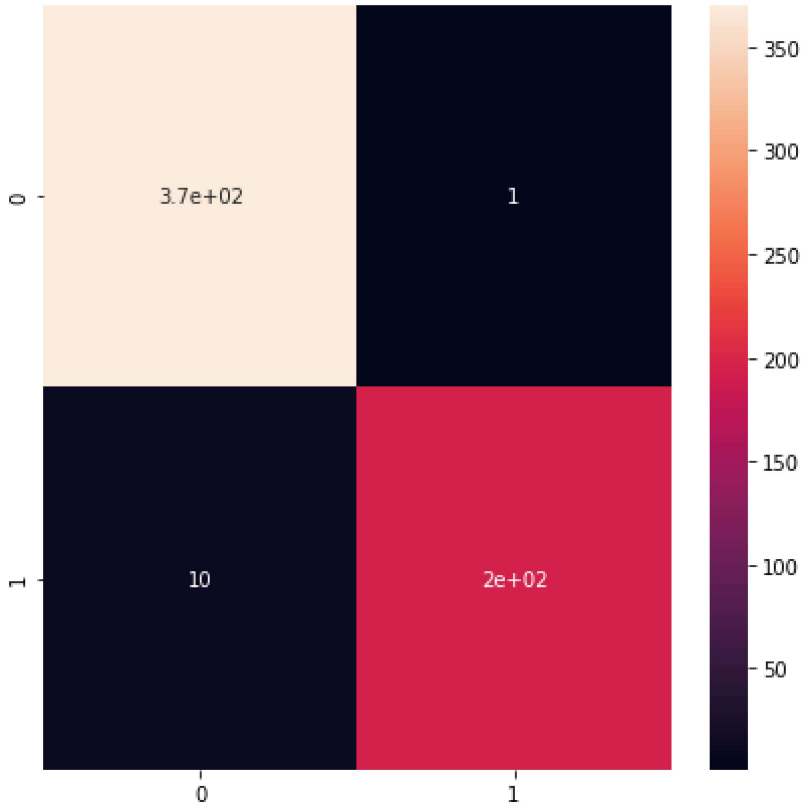
```
576
```

```
# Obtain the accuracy of the model
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(original, predict)
accuracy
```

```
0.9809027777777778
```

```
# plot the confusion matrix
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(original, predict)
plt.figure(figsize = (7,7))
sns.heatmap(cm, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa412fd6fd0>
```




```
from sklearn.metrics import classification_report

report = classification_report(original, predict, labels = [0,1])
print(report)
```

```
                precision    recall  f1-score   support

             0       0.97      1.00      0.99       371
             1       0.99      0.95      0.97       205

   micro avg       0.98      0.98      0.98       576
   macro avg       0.98      0.97      0.98       576
weighted avg       0.98      0.98      0.98       576

/usr/local/lib/python3.7/dist-packages/numpy/lib/arraysetops.py:576: FutureWarning:

elementwise comparison failed; returning scalar instead, but in the future will perfo
```

```
# Get the dataframe containing MRIs which have masks associated with them.
brain_df_mask = brain_df[brain_df['mask'] == 1]
brain_df_mask.shape
```

```
(1373, 4)
```

```
# split the data into train and test data

from sklearn.model_selection import train_test_split

X_train, X_val = train_test_split(brain_df_mask, test_size=0.15)
X_test, X_val = train_test_split(X_val, test_size=0.5)
```

```
# create separate list for imageId, classId to pass into the generator

train_ids = list(X_train.image_path)
train_mask = list(X_train.mask_path)

val_ids = list(X_val.image_path)
val_mask= list(X_val.mask_path)
```

```
# Utilities file contains the code for custom loss function and custom data generator
from utilities import DataGenerator

# create image generators

training_generator = DataGenerator(train_ids,train_mask)
validation_generator = DataGenerator(val_ids,val_mask)
```

```
def resblock(X, f):


  # make a copy of input
  X_copy = X

  # main path
  # Read more about he_normal: https://medium.com/@prateekvishnu/xavier-and-he-normal-he-e

  X = Conv2D(f, kernel_size = (1,1) ,strides = (1,1),kernel_initializer ='he_normal')(X)
  X = BatchNormalization()(X)
  X = Activation('relu')(X)

  X = Conv2D(f, kernel_size = (3,3), strides =(1,1), padding = 'same', kernel_initializer
  X = BatchNormalization()(X)

  # Short path
  # Read more here: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-ke

  X_copy = Conv2D(f, kernel_size = (1,1), strides =(1,1), kernel_initializer ='he_normal')
  X_copy = BatchNormalization()(X_copy)

  # Adding the output from main path and short path together

  X = Add()([X,X_copy])
  X = Activation('relu')(X)

  return X
```

```
# function to upscale and concatenate the values passsed
def upsample_concat(x, skip):
  x = UpSampling2D((2,2))(x)
  merge = Concatenate()([x, skip])

  return merge
```

```
input_shape = (256,256,3)

# Input tensor shape
X_input = Input(input_shape)

# Stage 1
conv1_in = Conv2D(16,3,activation= 'relu', padding = 'same', kernel_initializer ='he_norma
conv1_in = BatchNormalization()(conv1_in)
conv1_in = Conv2D(16,3,activation= 'relu', padding = 'same', kernel_initializer ='he_norma
conv1_in = BatchNormalization()(conv1_in)
pool_1 = MaxPool2D(pool_size = (2,2))(conv1_in)

# Stage 2
conv2_in = resblock(pool_1, 32)
pool_2 = MaxPool2D(pool_size = (2,2))(conv2_in)

# Stage 3
conv3_in = resblock(pool_2, 64)
pool_3 = MaxPool2D(pool_size = (2,2))(conv3_in)

# Stage 4
conv4_in = resblock(pool_3, 128)
pool_4 = MaxPool2D(pool_size = (2,2))(conv4_in)

# Stage 5 (Bottle Neck)
conv5_in = resblock(pool_4, 256)

# Upscale stage 1
up_1 = upsample_concat(conv5_in, conv4_in)
up_1 = resblock(up_1, 128)

# Upscale stage 2
up_2 = upsample_concat(up_1, conv3_in)
up_2 = resblock(up_2, 64)

# Upscale stage 3
up_3 = upsample_concat(up_2, conv2_in)
up_3 = resblock(up_3, 32)

# Upscale stage 4
up_4 = upsample_concat(up_3, conv1_in)
up_4 = resblock(up_4, 16)

# Final Output
output = Conv2D(1, (1,1), padding = "same", activation = "sigmoid")(up_4)

model_seg = Model(inputs = X_input, outputs = output )
```

```
model_seg.summary()
```

```
batch_normalization_19 (BatchNo (None, 64, 64, 64)   256         conv2d_19[0][0]
__________________________________________________________________________________
add_5 (Add)                     (None, 64, 64, 64)   0           batch_normalizati
                                                                 batch_normalizati
```

```
activation_11 (Activation)      (None, 64, 64, 64)   0           add_5[0][0]

up_sampling2d_2 (UpSampling2D)  (None, 128, 128, 64) 0           activation_11[0][0

concatenate_2 (Concatenate)     (None, 128, 128, 96) 0           up_sampling2d_2[0
                                                                 activation_1[0][0

conv2d_20 (Conv2D)              (None, 128, 128, 32) 3104        concatenate_2[0][0

batch_normalization_20 (BatchNo (None, 128, 128, 32) 128         conv2d_20[0][0]

activation_12 (Activation)      (None, 128, 128, 32) 0           batch_normalizati

conv2d_21 (Conv2D)              (None, 128, 128, 32) 9248        activation_12[0][0

conv2d_22 (Conv2D)              (None, 128, 128, 32) 3104        concatenate_2[0][0

batch_normalization_21 (BatchNo (None, 128, 128, 32) 128         conv2d_21[0][0]

batch_normalization_22 (BatchNo (None, 128, 128, 32) 128         conv2d_22[0][0]

add_6 (Add)                     (None, 128, 128, 32) 0           batch_normalizati
                                                                 batch_normalizati

activation_13 (Activation)      (None, 128, 128, 32) 0           add_6[0][0]

up_sampling2d_3 (UpSampling2D)  (None, 256, 256, 32) 0           activation_13[0][0

concatenate_3 (Concatenate)     (None, 256, 256, 48) 0           up_sampling2d_3[0
                                                                 batch_normalizati

conv2d_23 (Conv2D)              (None, 256, 256, 16) 784         concatenate_3[0][0

batch_normalization_23 (BatchNo (None, 256, 256, 16) 64          conv2d_23[0][0]

activation_14 (Activation)      (None, 256, 256, 16) 0           batch_normalizati

conv2d_24 (Conv2D)              (None, 256, 256, 16) 2320        activation_14[0][0

conv2d_25 (Conv2D)              (None, 256, 256, 16) 784         concatenate_3[0][0

batch_normalization_24 (BatchNo (None, 256, 256, 16) 64          conv2d_24[0][0]

batch_normalization_25 (BatchNo (None, 256, 256, 16) 64          conv2d_25[0][0]

add_7 (Add)                     (None, 256, 256, 16) 0           batch_normalizati
                                                                 batch_normalizati

activation_15 (Activation)      (None, 256, 256, 16) 0           add_7[0][0]

conv2d_26 (Conv2D)              (None, 256, 256, 1)  17          activation_15[0][0
==================================================================================
Total params: 1,210,513
Trainable params: 1,206,129
```

```
# Utilities file contains the code for custom loss function and custom data generator

from utilities import focal_tversky, tversky_loss, tversky
```

```
# Compile the model
adam = tf.keras.optimizers.Adam(lr = 0.05, epsilon = 0.1)
model_seg.compile(optimizer = adam, loss = focal_tversky, metrics = [tversky])
```

```
# use early stopping to exit training if validation loss is not decreasing even after cert
earlystopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=20)

# save the best model with lower validation loss
checkpointer = ModelCheckpoint(filepath="ResUNet-weights.hdf5", verbose=1, save_best_only=
```

```
history = model_seg.fit(training_generator, epochs = 1, validation_data = validation_gener
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-64-5b9c020f8c29> in <module>()
----> 1 history = model_seg.fit(training_generator, epochs = 1, validation_data =
validation_generator, callbacks = [checkpointer, earlystopping])

                                 9 frames
/usr/local/lib/python3.7/dist-packages/tensorflow/python/framework/func_graph.py in
wrapper(*args, **kwargs)
    975           except Exception as e:  # pylint:disable=broad-except
    976             if hasattr(e, "ag_error_metadata"):
--> 977               raise e.ag_error_metadata.to_exception(e)
    978             else:
    979               raise

TypeError: in user code:

    /usr/local/lib/python3.7/dist-
packages/tensorflow/python/keras/engine/training.py:805 train_function  *
        return step_function(self, iterator)
    /content/drive/My Drive/ai/Healthcare AI Datasets/Brain_MRI/utilities.py:218
focal_tversky  *
        pt_1 = tversky(y_true, y_pred)
    /content/drive/My Drive/ai/Healthcare AI Datasets/Brain_MRI/utilities.py:208
tversky  *
        true_pos = K.sum(y_true_pos * y_pred_pos)
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/ops/math_ops.py:1180
binary_op_wrapper
        raise e
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/ops/math_ops.py:1164
binary_op_wrapper
        return func(x, y, name=name)
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/ops/math_ops.py:1496
_mul_dispatch
        return multiply(x, y, name=name)
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:201
wrapper
        return target(*args, **kwargs)
    /usr/local/lib/python3.7/dist-packages/tensorflow/python/ops/math_ops.py:518
multiply
```

```
from utilities import focal_tversky, tversky_loss, tversky

with open('ResUNet-MRI.json', 'r') as json_file:
    json_savedModel= json_file.read()
```

```
    json_savedModel= json_file.read()

# load the model architecture
model_seg = tf.keras.models.model_from_json(json_savedModel)
model_seg.load_weights('weights_seg.hdf5')
adam = tf.keras.optimizers.Adam(lr = 0.05, epsilon = 0.1)
model_seg.compile(optimizer = adam, loss = focal_tversky, metrics = [tversky])
```

```
# Utilities file contains the code for custom loss function and custom data generator
from utilities import prediction

# making prediction
image_id, mask, has_mask = prediction(test, model, model_seg)
```

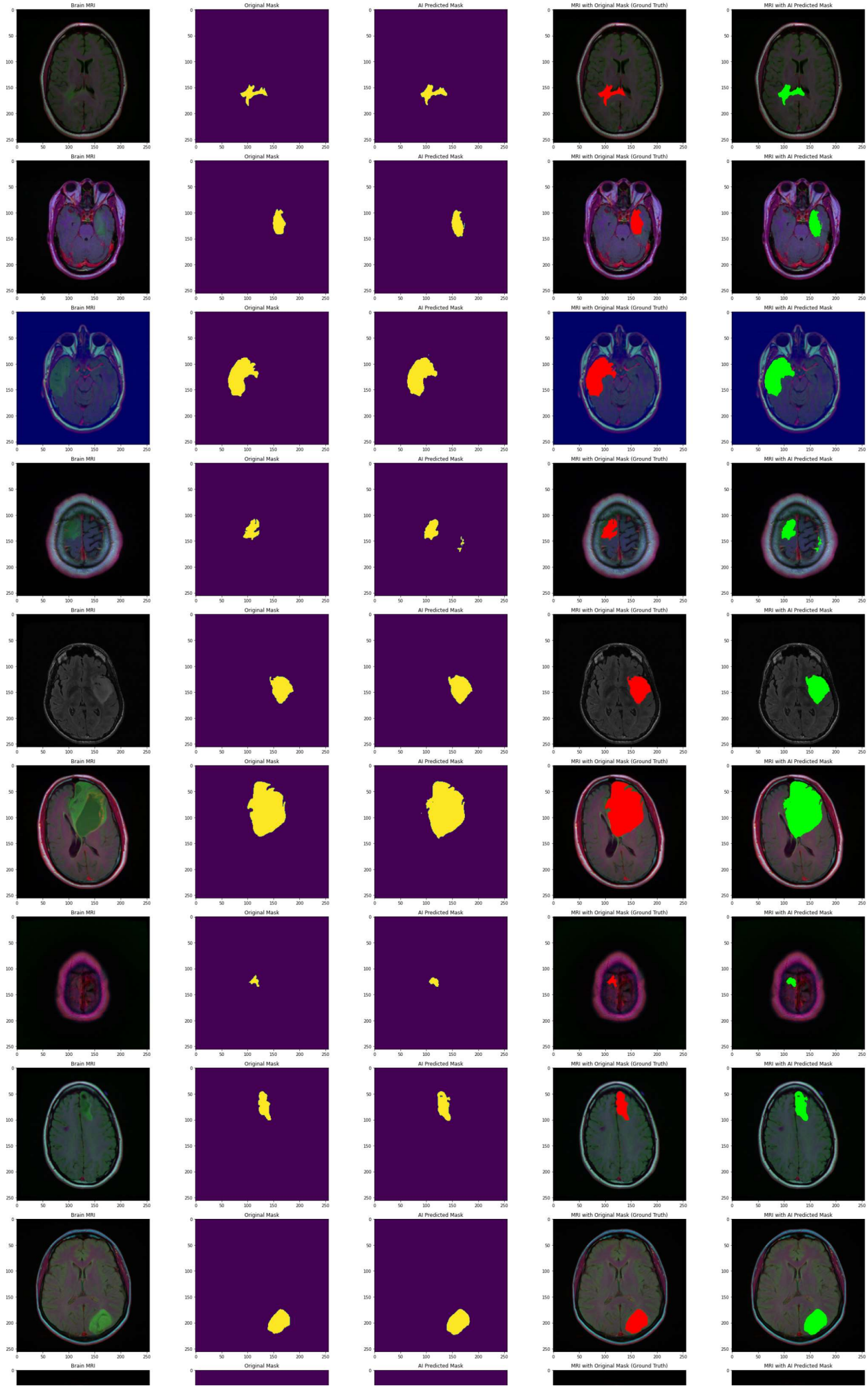
```
# creating a dataframe for the result
df_pred = pd.DataFrame({'image_path': image_id,'predicted_mask': mask,'has_mask': has_mask
df_pred
```

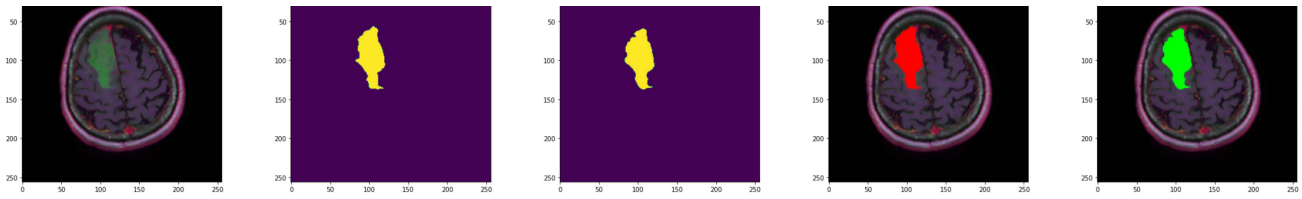
	image_path	predicted_mask	has_mask
0	TCGA_FG_6689_20020326/TCGA_FG_6689_20020326_15...	No mask	0
1	TCGA_CS_6665_20010817/TCGA_CS_6665_20010817_2.tif	No mask	0
2	TCGA_HT_7692_19960724/TCGA_HT_7692_19960724_3.tif	No mask	0
3	TCGA_DU_A5TP_19970614/TCGA_DU_A5TP_19970614_22...	[[[[8.0426537e-07], [3.3971035e-06], [8.833198...	1
4	TCGA_HT_7881_19981015/TCGA_HT_7881_19981015_27...	[[[[7.072562e-07], [2.4564724e-06], [4.5194197...	1
...	...	...	...
585	TCGA_DU_8163_19961119/TCGA_DU_8163_19961119_18...	[[[[8.269657e-07], [3.8015035e-06], [1.1302963...	1

```
# Merge the dataframe containing predicted results with the original test data.
df_pred = test.merge(df_pred, on = 'image_path')
df_pred.head()
```

		image_path
0	TCGA_FG_6689_20020326/TCGA_FG_6689_20020326_15...	TCGA_FG_6689_20020326/TC

```
count = 0
fig, axs = plt.subplots(10, 5, figsize=(30, 50))
for i in range(len(df_pred)):
  if df_pred['has_mask'][i] == 1 and count < 10:
    # read the images and convert them to RGB format
    img = io.imread(df_pred.image_path[i])
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    axs[count][0].title.set_text("Brain MRI")
    axs[count][0].imshow(img)

    # Obtain the mask for the image
    mask = io.imread(df_pred.mask_path[i])
    axs[count][1].title.set_text("Original Mask")
    axs[count][1].imshow(mask)

    # Obtain the predicted mask for the image
    predicted_mask = np.asarray(df_pred.predicted_mask[i])[0].squeeze().round()
    axs[count][2].title.set_text("AI Predicted Mask")
    axs[count][2].imshow(predicted_mask)

    # Apply the mask to the image 'mask==255'
    img[mask == 255] = (255, 0, 0)
    axs[count][3].title.set_text("MRI with Original Mask (Ground Truth)")
    axs[count][3].imshow(img)

    img_ = io.imread(df_pred.image_path[i])
    img_ = cv2.cvtColor(img_, cv2.COLOR_BGR2RGB)
    img_[predicted_mask == 1] = (0, 255, 0)
    axs[count][4].title.set_text("MRI with AI Predicted Mask")
    axs[count][4].imshow(img_)
    count += 1

fig.tight_layout()
```


Brain MRI
Original Mask
AI Predicted Mask
MRI with Original Mask (Ground Truth)
MRI with AI Predicted Mask



✓ 23s completed at 9:30 PM