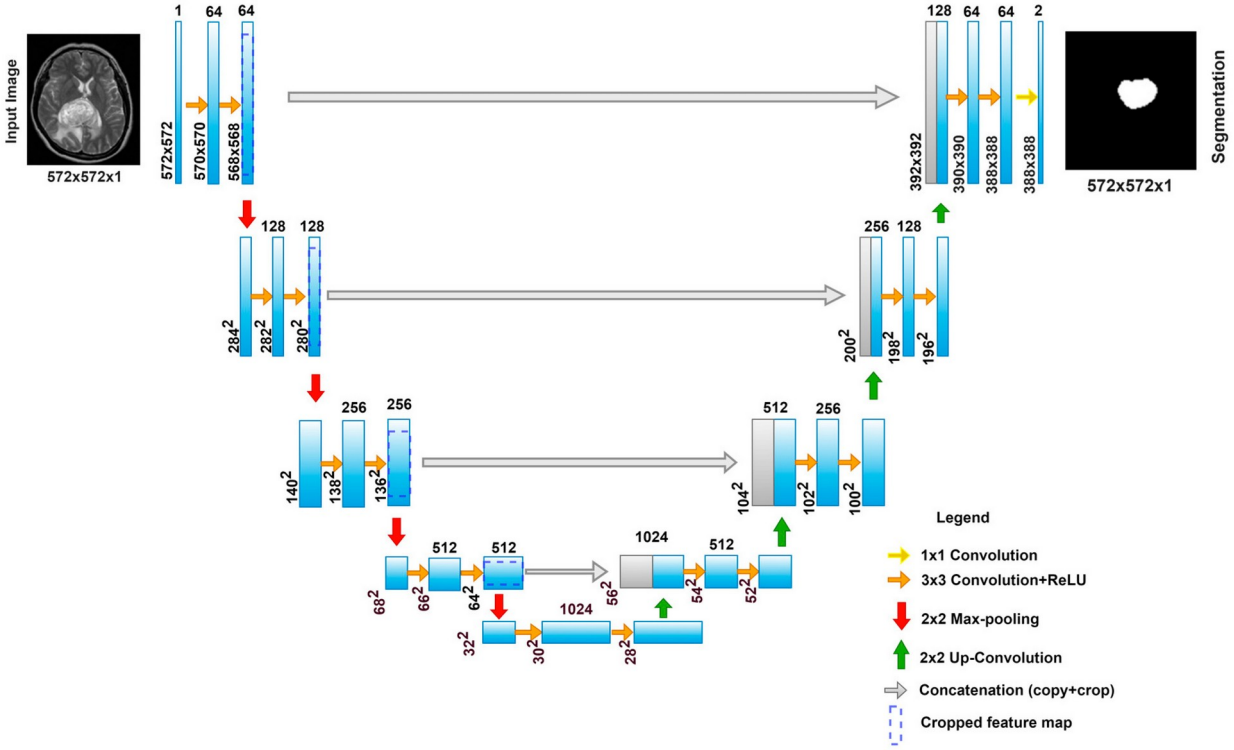
Brain tumor segmentation with UNET



Steps to perform brain tumor segmentation using the U-Net architecture:

1. **Data Collection and Preparation**:
   - Gather a dataset of brain MRI images along with corresponding ground truth masks indicating tumor regions.
   - Preprocess the images (e.g., resizing, normalization) to ensure uniformity and enhance model performance.
2. **Data Augmentation**:
   - Augment the dataset to increase its size and diversity, which helps in improving the robustness and generalization of the model.
   - Common augmentation techniques include rotation, flipping, scaling, and adding noise to the images.
3. **Splitting the Dataset**:
   - Divide the dataset into training, validation, and testing sets. The training set is used to train the model, the validation set is used to tune hyperparameters and monitor training progress, and the testing set is used to evaluate the model's performance.
4. **Building the U-Net Model**:
   - Implement the U-Net architecture, which consists of an encoder (downsampling path) and a decoder (upsampling path) connected by skip connections.

- Use convolutional layers with appropriate activation functions (e.g., ReLU), pooling layers (e.g., max pooling), and upsampling layers (e.g., transposed convolution) to build the network.

5. **Loss Function Selection**:
   - Choose an appropriate loss function for training the model. For binary segmentation tasks like tumor segmentation, commonly used loss functions include binary cross-entropy loss, Dice loss, and Jaccard loss.
6. **Training the Model**:
   - Train the U-Net model on the training dataset using the selected loss function and an optimization algorithm (e.g., Adam optimizer).
   - Monitor the model's performance on the validation set and adjust hyperparameters if necessary (e.g., learning rate, batch size) to improve performance and prevent overfitting.
   - Train the model until convergence or until the validation loss stops decreasing.
7. **Evaluation**:
   - Evaluate the trained model on the testing set to assess its performance in segmenting brain tumors.
   - Compute evaluation metrics such as Dice coefficient, Jaccard index, sensitivity, specificity, and accuracy to quantify the model's performance.
8. **Post-processing**:
   - Perform post-processing techniques (e.g., morphological operations like dilation and erosion) to refine the segmentation masks and improve the segmentation results.
9. **Visualization**:
   - Visualize the segmented tumor regions overlaid on the original MRI images to visually inspect the quality of the segmentation results.
10. **Deployment**:

- Deploy the trained model for practical use, such as automating tumor segmentation in clinical settings or integrating it into medical image analysis software.

```
import os
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from mpl_toolkits.axes_grid1 import ImageGrid

import cv2
from glob import glob

import tensorflow as tf
from tensorflow.keras import Input
from tensorflow.keras.models import Model, load_model, save_model
from tensorflow.keras.layers import Input, Activation,
BatchNormalization, Dropout, Lambda, Conv2D
from tensorflow.keras.layers import Conv2DTranspose, MaxPooling2D,
```

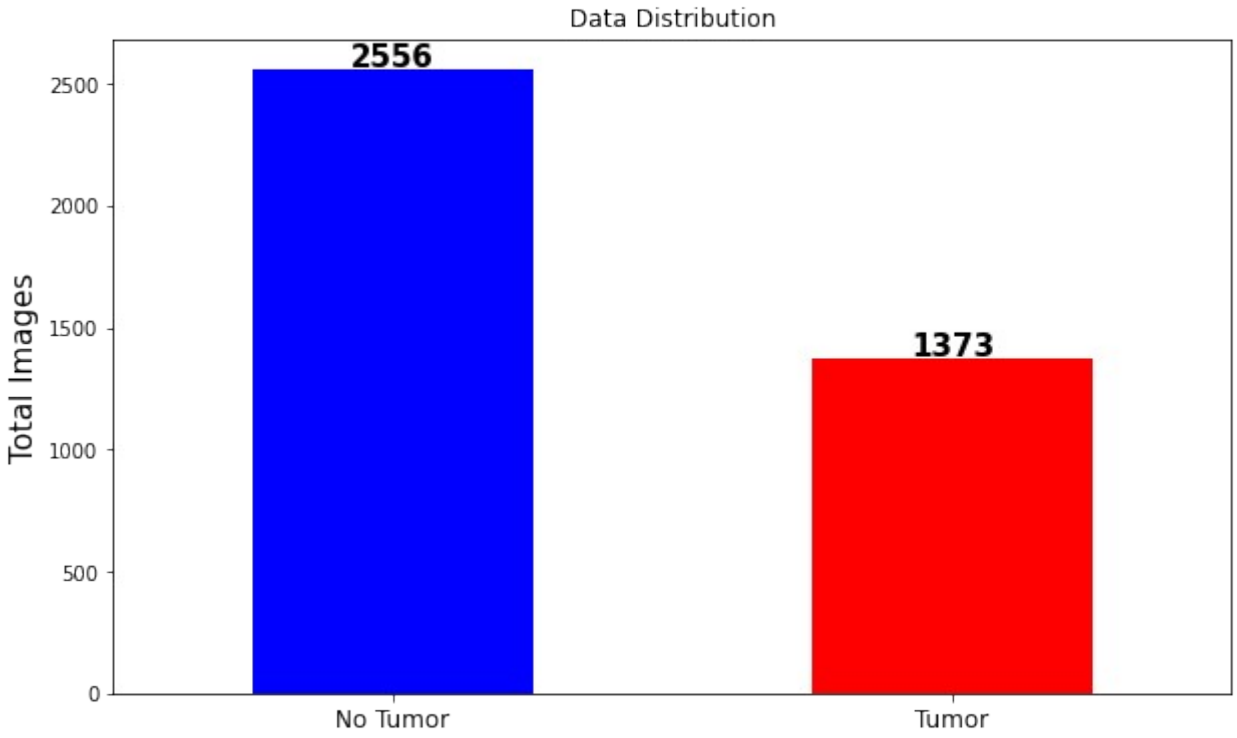
```
concatenate, AveragePooling2D, Dense, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,
ReduceLROnPlateau

from tensorflow.keras import backend as K
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Set parameters
IMAGE_SIZE = (256, 256)
```

Create DataFrame

```
mask_files = glob('../input/lgg-mri-segmentation/kaggle_3m/*/*_mask*')
train_files = [file.replace('_mask', '') for file in mask_files]

def diagnosis(mask_path):
    value = np.max(cv2.imread(mask_path))
    return '1' if value > 0 else '0'
df = pd.DataFrame({"image_path": train_files,
                   "mask_path": mask_files,
                  "diagnosis":[diagnosis(x) for x in mask_files]})
df.head()
```

```
                                          image_path  \
0  ../input/lgg-mri-segmentation/kaggle_3m/TCGA_D...
1  ../input/lgg-mri-segmentation/kaggle_3m/TCGA_D...
2  ../input/lgg-mri-segmentation/kaggle_3m/TCGA_D...
3  ../input/lgg-mri-segmentation/kaggle_3m/TCGA_D...
4  ../input/lgg-mri-segmentation/kaggle_3m/TCGA_D...

                                           mask_path diagnosis
0  ../input/lgg-mri-segmentation/kaggle_3m/TCGA_D...         0
1  ../input/lgg-mri-segmentation/kaggle_3m/TCGA_D...         0
2  ../input/lgg-mri-segmentation/kaggle_3m/TCGA_D...         1
3  ../input/lgg-mri-segmentation/kaggle_3m/TCGA_D...         1
4  ../input/lgg-mri-segmentation/kaggle_3m/TCGA_D...         1
```

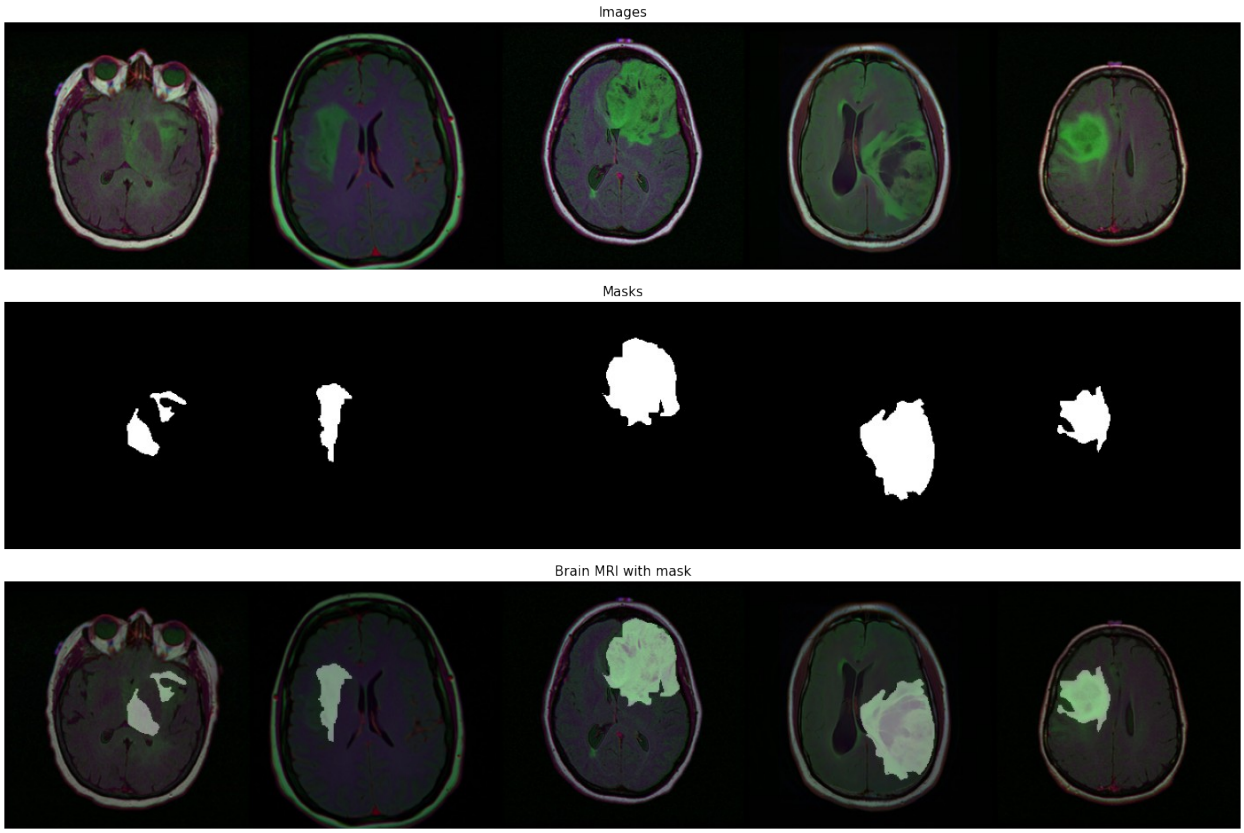
```
ax = df['diagnosis'].value_counts().plot(kind='bar', stacked=True,
figsize=(10,6), color=['blue', 'red'])
ax.set_title('Data Distribution')
ax.set_ylabel('Total Images', fontsize=15)
ax.set_xticklabels(['No Tumor', 'Tumor'], fontsize=12, rotation=0)
for i, rows in enumerate(df['diagnosis'].value_counts().values):
    ax.annotate(int(rows), xy=(i, rows+12), ha='center',
fontweight='bold', fontsize=15)
```

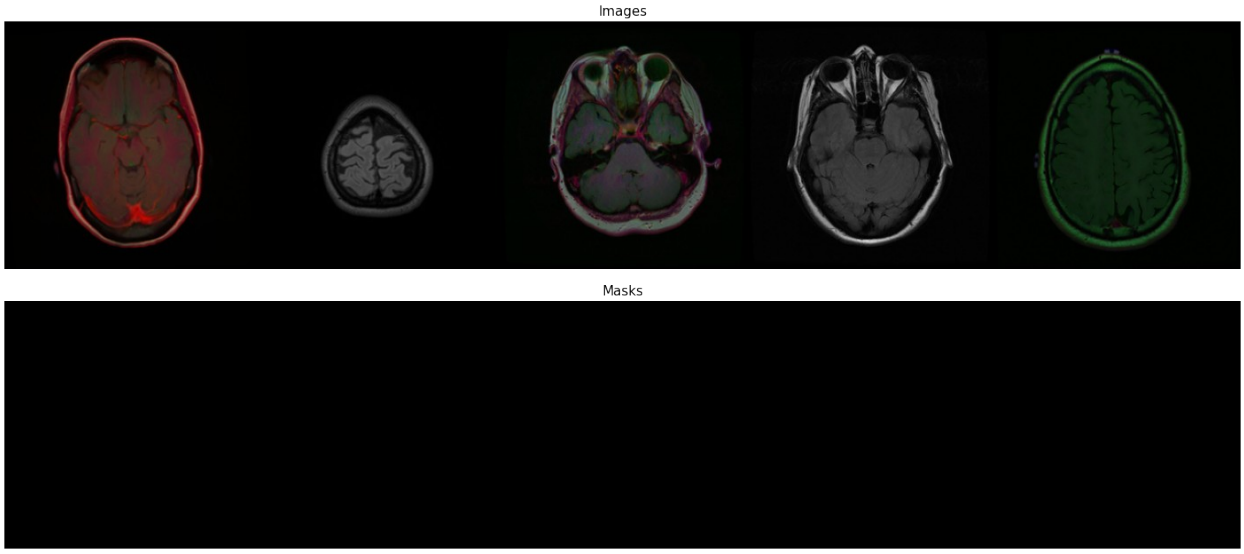


Visualize MRI with Mask

```
df_positive = df[df['diagnosis']=='1'].sample(5).values
df_negative = df[df['diagnosis']=='0'].sample(5).values

def show_data(df, positive=True):
    images = []
    masks = []
    for data in df:
        img = cv2.imread(data[0])
        mask = cv2.imread(data[1])
        images.append(img)
        masks.append(mask)
    images = np.hstack(np.array(images))
    masks = np.hstack(np.array(masks))

    fig = plt.figure(figsize=(25,25))
    if positive:
        grid = ImageGrid(fig, 111, nrows_ncols=(3,1), axes_pad=0.5)
    else:
        grid = ImageGrid(fig, 111, nrows_ncols=(2,1), axes_pad=0.5)
    grid[0].imshow(images)
    grid[0].set_title('Images', fontsize=15)
    grid[0].axis('off')
    grid[1].imshow(masks)
    grid[1].set_title('Masks', fontsize=15)
    grid[1].axis('off')
```

```
    if positive:
        grid[2].imshow(images)
        grid[2].imshow(masks, alpha=0.4)
        grid[2].set_title('Brain MRI with mask', fontsize=15)
        grid[2].axis('off')

show_data(df_positive)
show_data(df_negative, positive=False)
```





Split data into Train, Validation and Test Set

```
from sklearn.model_selection import train_test_split
df_train, df_test = train_test_split(df, test_size=0.15)
df_train, df_val = train_test_split(df_train, test_size=0.15)
print(df_train.values.shape)
print(df_val.values.shape)
print(df_test.values.shape)
```

```
(2838, 3)
(501, 3)
(590, 3)
```

U-Net Model

Data Generator, Data Augmentation and Adjust Data

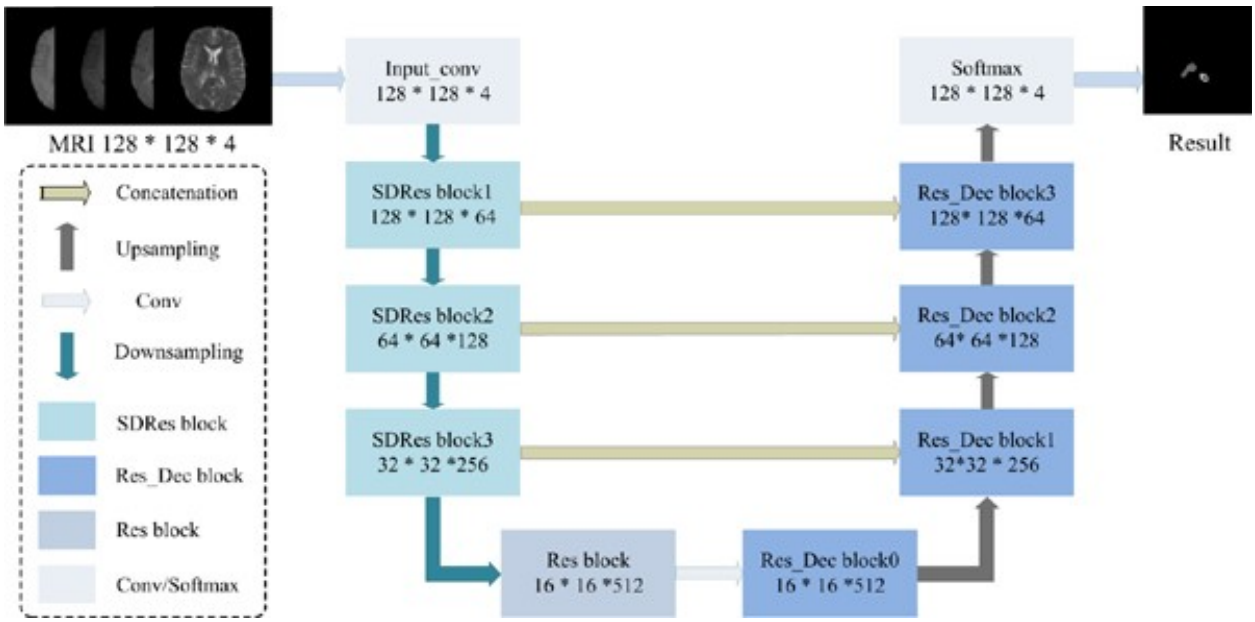
```
def train_generator(data_frame, batch_size, aug_dict,
        image_color_mode="rgb",
        mask_color_mode="grayscale",
        image_save_prefix="image",
        mask_save_prefix="mask",
        save_to_dir=None,
        target_size=(256,256),
        seed=1):

    image_datagen = ImageDataGenerator(**aug_dict)
    mask_datagen = ImageDataGenerator(**aug_dict)

    image_generator = image_datagen.flow_from_dataframe(
        data_frame,
        x_col = "image_path",
        class_mode = None,
```

```
        color_mode = image_color_mode,
        target_size = target_size,
        batch_size = batch_size,
        save_to_dir = save_to_dir,
        save_prefix  = image_save_prefix,
        seed = seed)

    mask_generator = mask_datagen.flow_from_dataframe(
        data_frame,
        x_col = "mask_path",
        class_mode = None,
        color_mode = mask_color_mode,
        target_size = target_size,
        batch_size = batch_size,
        save_to_dir = save_to_dir,
        save_prefix  = mask_save_prefix,
        seed = seed)

    train_gen = zip(image_generator, mask_generator)

    for (img, mask) in train_gen:
        img, mask = adjust_data(img, mask)
        yield (img,mask)

def adjust_data(img,mask):
    img = img / 255.
    mask = mask / 255.
    mask[mask > 0.5] = 1
    mask[mask <= 0.5] = 0

    return (img, mask)

smooth=1.

def dice_coef(y_true, y_pred):
    y_true = K.flatten(y_true)


    y_pred = K.flatten(y_pred)
    intersection = K.sum(y_true * y_pred)
    union = K.sum(y_true) + K.sum(y_pred)
    return (2.0 * intersection + smooth) / (union + smooth)

def dice_coef_loss(y_true, y_pred):
    return 1 - dice_coef(y_true, y_pred)

def bce_dice_loss(y_true, y_pred):
    bce = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    return dice_coef_loss(y_true, y_pred) + bce(y_true, y_pred)
```

```
def iou(y_true, y_pred):
    intersection = K.sum(y_true * y_pred)
    sum_ = K.sum(y_true + y_pred)
    jac = (intersection + smooth) / (sum_ - intersection + smooth)
    return jac
```

UNet Model Architecture

Working and Flow Diagram of UNet



```
def unet(input_size=(256,256,3)):
    inputs = Input(input_size)

    conv1 = Conv2D(64, (3, 3), padding='same')(inputs)
    bn1 = Activation('relu')(conv1)
    conv1 = Conv2D(64, (3, 3), padding='same')(bn1)
    bn1 = BatchNormalization(axis=3)(conv1)
    bn1 = Activation('relu')(bn1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(bn1)

    conv2 = Conv2D(128, (3, 3), padding='same')(pool1)
    bn2 = Activation('relu')(conv2)
    conv2 = Conv2D(128, (3, 3), padding='same')(bn2)
    bn2 = BatchNormalization(axis=3)(conv2)
    bn2 = Activation('relu')(bn2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(bn2)

    conv3 = Conv2D(256, (3, 3), padding='same')(pool2)
    bn3 = Activation('relu')(conv3)
    conv3 = Conv2D(256, (3, 3), padding='same')(bn3)
    bn3 = BatchNormalization(axis=3)(conv3)
    bn3 = Activation('relu')(bn3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(bn3)

    conv4 = Conv2D(512, (3, 3), padding='same')(pool3)
    bn4 = Activation('relu')(conv4)
    conv4 = Conv2D(512, (3, 3), padding='same')(bn4)
    bn4 = BatchNormalization(axis=3)(conv4)
    bn4 = Activation('relu')(bn4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(bn4)

    conv5 = Conv2D(1024, (3, 3), padding='same')(pool4)
    bn5 = Activation('relu')(conv5)
    conv5 = Conv2D(1024, (3, 3), padding='same')(bn5)
    bn5 = BatchNormalization(axis=3)(conv5)
    bn5 = Activation('relu')(bn5)

    up6 = concatenate([Conv2DTranspose(512, (2, 2), strides=(2, 2),
padding='same')(bn5), conv4], axis=3)
    conv6 = Conv2D(512, (3, 3), padding='same')(up6)
    bn6 = Activation('relu')(conv6)
    conv6 = Conv2D(512, (3, 3), padding='same')(bn6)
    bn6 = BatchNormalization(axis=3)(conv6)
    bn6 = Activation('relu')(bn6)

    up7 = concatenate([Conv2DTranspose(256, (2, 2), strides=(2, 2),
padding='same')(bn6), conv3], axis=3)
    conv7 = Conv2D(256, (3, 3), padding='same')(up7)
    bn7 = Activation('relu')(conv7)
    conv7 = Conv2D(256, (3, 3), padding='same')(bn7)
```

```
    bn7 = BatchNormalization(axis=3)(conv7)
    bn7 = Activation('relu')(bn7)

    up8 = concatenate([Conv2DTranspose(128, (2, 2), strides=(2, 2),
padding='same')(bn7), conv2], axis=3)
    conv8 = Conv2D(128, (3, 3), padding='same')(up8)
    bn8 = Activation('relu')(conv8)
    conv8 = Conv2D(128, (3, 3), padding='same')(bn8)
    bn8 = BatchNormalization(axis=3)(conv8)
    bn8 = Activation('relu')(bn8)

    up9 = concatenate([Conv2DTranspose(64, (2, 2), strides=(2, 2),
padding='same')(bn8), conv1], axis=3)
    conv9 = Conv2D(64, (3, 3), padding='same')(up9)
    bn9 = Activation('relu')(conv9)
    conv9 = Conv2D(64, (3, 3), padding='same')(bn9)
    bn9 = BatchNormalization(axis=3)(conv9)
    bn9 = Activation('relu')(bn9)

    conv10 = Conv2D(1, (1, 1), activation='sigmoid')(bn9)

    return Model(inputs=[inputs], outputs=[conv10])
```

Training

```
# Set parameters
EPOCHS = 10
BATCH_SIZE = 32
learning_rate = 1e-4

train_generator_args = dict(rotation_range=0.1,
                            width_shift_range=0.05,
                            height_shift_range=0.05,
                            shear_range=0.05,
                            zoom_range=0.05,
                            horizontal_flip=True,
                            vertical_flip=True,
                            fill_mode='nearest')
train_gen = train_generator(df_train, BATCH_SIZE,
                                train_generator_args,
                                target_size=IMAGE_SIZE)

val_gen = train_generator(df_val, BATCH_SIZE,
                                dict(),
                                target_size=IMAGE_SIZE)

model = unet(input_size=(IMAGE_SIZE[0], IMAGE_SIZE[1], 3))
```

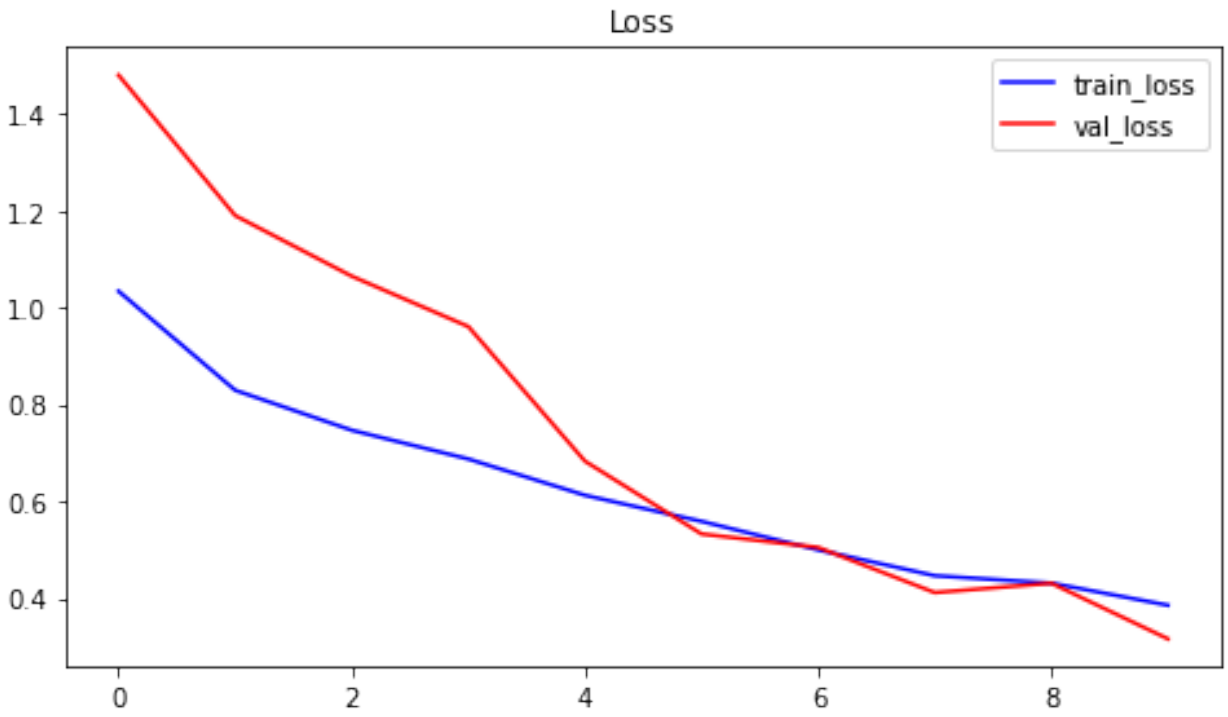
```
opt = Adam(lr=learning_rate, beta_1=0.9, beta_2=0.999, epsilon=None,
amsgrad=False)
model.compile(optimizer=opt, loss=bce_dice_loss, metrics=[iou,
dice_coef])

callbacks = [ModelCheckpoint('unet_brainMRI_seg.hdf5', verbose=0,
save_best_only=True),
             ReduceLROnPlateau(monitor='val_loss', factor=0.1,
patience=5, verbose=1, min_lr=1e-11),
             EarlyStopping(monitor='val_loss',
restore_best_weights=True, patience=15)]

history = model.fit(train_gen,
                    steps_per_epoch=len(df_train) / BATCH_SIZE,
                    epochs=EPOCHS,
                    callbacks=callbacks,
                    validation_data = val_gen,
                    validation_steps=len(df_val) / BATCH_SIZE)
```

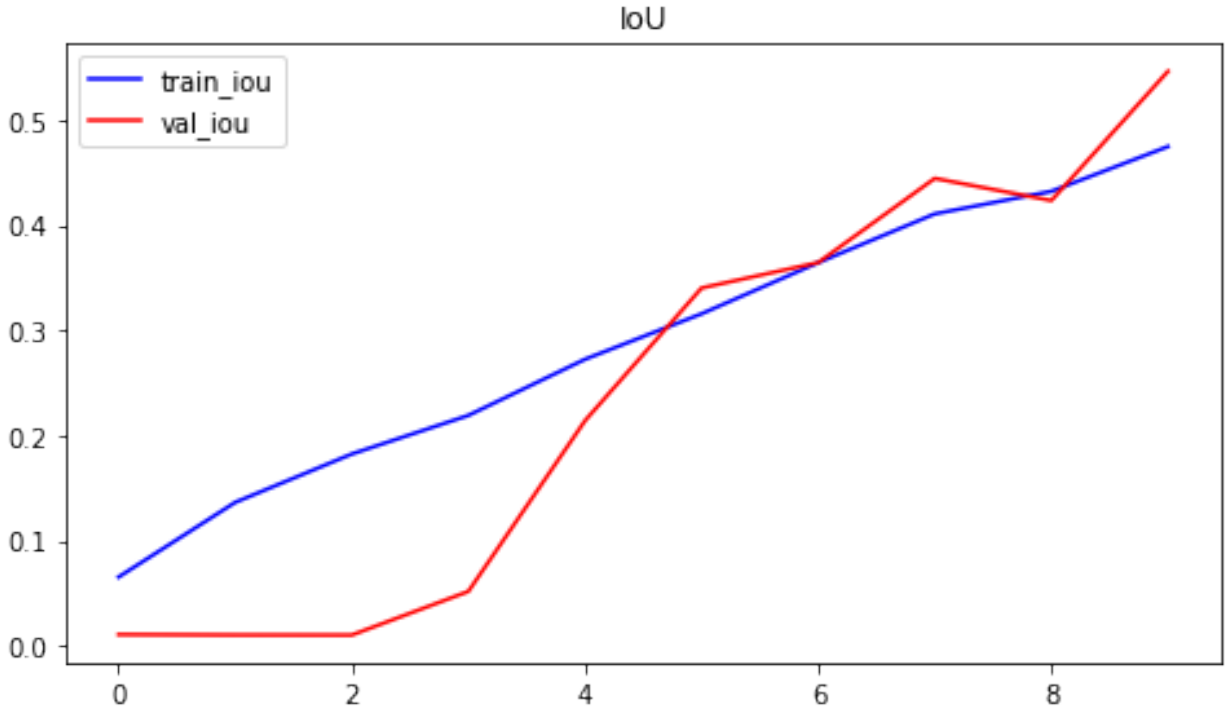
```
Found 2838 validated image filenames.
Found 2838 validated image filenames.
Epoch 1/10
89/88 [==============================] - ETA: 0s - loss: 1.1530 - iou:
0.0374 - dice_coef: 0.0708Found 501 validated image filenames.
Found 501 validated image filenames.
88/88 [==============================] - 109s 1s/step - loss: 1.1517 -
iou: 0.0377 - dice_coef: 0.0713 - val_loss: 1.4798 - val_iou: 0.0104 -
val_dice_coef: 0.0206
Epoch 2/10
88/88 [==============================] - 70s 791ms/step - loss: 0.8467
- iou: 0.1279 - dice_coef: 0.2246 - val_loss: 1.1900 - val_iou: 0.0100
- val_dice_coef: 0.0199
Epoch 3/10
88/88 [==============================] - 69s 783ms/step - loss: 0.7537
- iou: 0.1780 - dice_coef: 0.3000 - val_loss: 1.0648 - val_iou: 0.0100
- val_dice_coef: 0.0197
Epoch 4/10
88/88 [==============================] - 69s 773ms/step - loss: 0.7083
- iou: 0.2065 - dice_coef: 0.3386 - val_loss: 0.9609 - val_iou: 0.0515
- val_dice_coef: 0.0966
Epoch 5/10
88/88 [==============================] - 69s 783ms/step - loss: 0.6353
- iou: 0.2540 - dice_coef: 0.4003 - val_loss: 0.6831 - val_iou: 0.2139
- val_dice_coef: 0.3493
Epoch 6/10
88/88 [==============================] - 69s 780ms/step - loss: 0.5807
- iou: 0.2964 - dice_coef: 0.4510 - val_loss: 0.5329 - val_iou: 0.3403
- val_dice_coef: 0.5042
Epoch 7/10
88/88 [==============================] - 70s 785ms/step - loss: 0.5229
```

```
- iou: 0.3484 - dice_coef: 0.5086 - val_loss: 0.5051 - val_iou: 0.3645
- val_dice_coef: 0.5298
Epoch 8/10
88/88 [==============================] - 70s 792ms/step - loss: 0.4598
- iou: 0.3994 - dice_coef: 0.5660 - val_loss: 0.4120 - val_iou: 0.4446
- val_dice_coef: 0.6128
Epoch 9/10
88/88 [==============================] - 69s 784ms/step - loss: 0.4599
- iou: 0.4095 - dice_coef: 0.5721 - val_loss: 0.4312 - val_iou: 0.4236
- val_dice_coef: 0.5892
Epoch 10/10
88/88 [==============================] - 69s 774ms/step - loss: 0.4012
- iou: 0.4604 - dice_coef: 0.6230 - val_loss: 0.3164 - val_iou: 0.5466
- val_dice_coef: 0.7032
```

Visualize the model performance

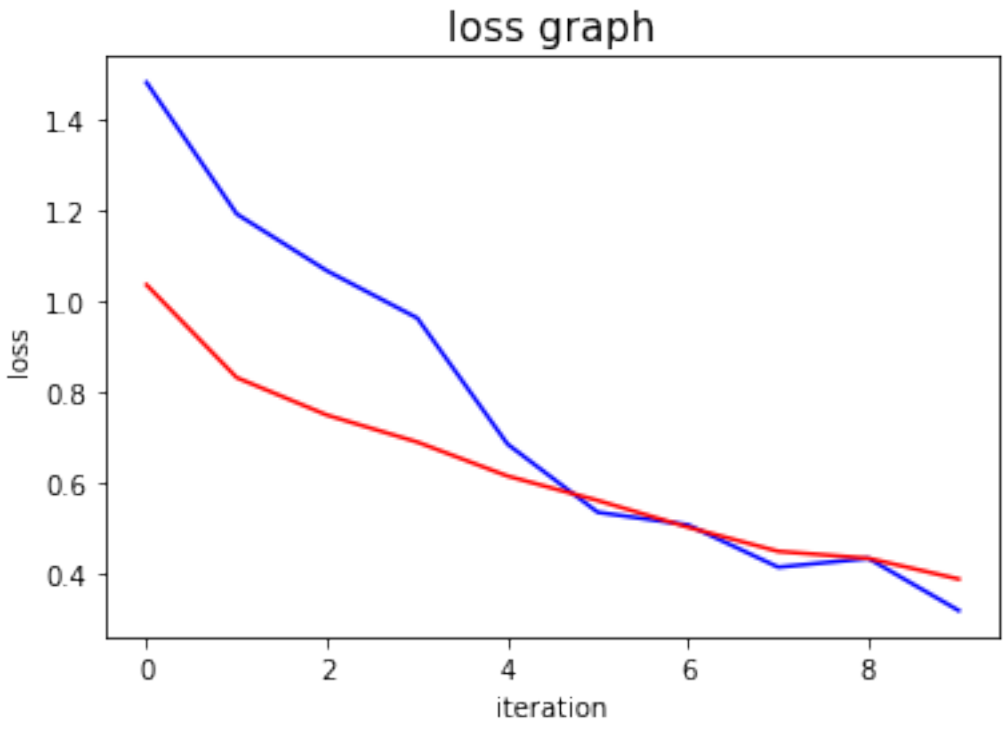
```
plt.figure(figsize=(8,15))
plt.subplot(3,1,1)
plt.plot(model.history.history['loss'], 'b-', label='train_loss')
plt.plot(model.history.history['val_loss'], 'r-', label='val_loss')
plt.legend(loc='best')
plt.title('Loss')

plt.subplot(3,1,2)
plt.plot(model.history.history['iou'], 'b-', label='train_iou')
plt.plot(model.history.history['val_iou'], 'r-', label='val_iou')
plt.legend(loc='best')
plt.title('IoU')

plt.subplot(3,1,3)
plt.plot(model.history.history['dice_coef'], 'b-',
label='train_dice_coef')
plt.plot(model.history.history['val_dice_coef'], 'r-',
label='val_dice_coef')
plt.legend(loc='best')
plt.title('Dice Coef')

Text(0.5, 1.0, 'Dice Coef')
```

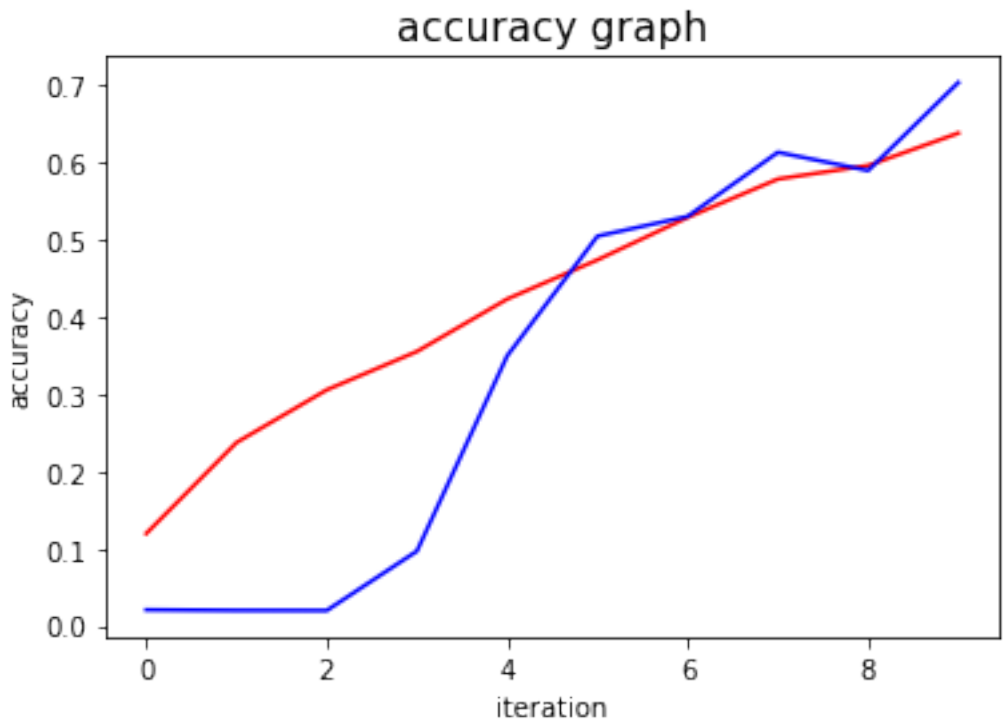

Loss
train_loss
val_loss
1.4
1.2
1.0
0.8
0.6
0.4
0
2
4
6
8


IoU
train_iou
val_iou
0.5
0.4
0.3
0.2
0.1
0.0
0
2
4
6
8


Dice Coef
train_dice_coef
val_dice_coef
0.7
0.6

```
a = history.history

list_traindice = a['dice_coef']
list_testdice = a['val_dice_coef']

list_trainjaccard = a['iou']
list_testjaccard = a['val_iou']

list_trainloss = a['loss']
list_testloss = a['val_loss']
plt.figure(1)
plt.plot(list_testloss, 'b-')
plt.plot(list_trainloss,'r-')
plt.xlabel('iteration')
plt.ylabel('loss')
plt.title('loss graph', fontsize = 15)
plt.figure(2)
plt.plot(list_traindice, 'r-')
plt.plot(list_testdice, 'b-')
plt.xlabel('iteration')
plt.ylabel('accuracy')
plt.title('accuracy graph', fontsize = 15)
plt.show()
```





Evaluate the model

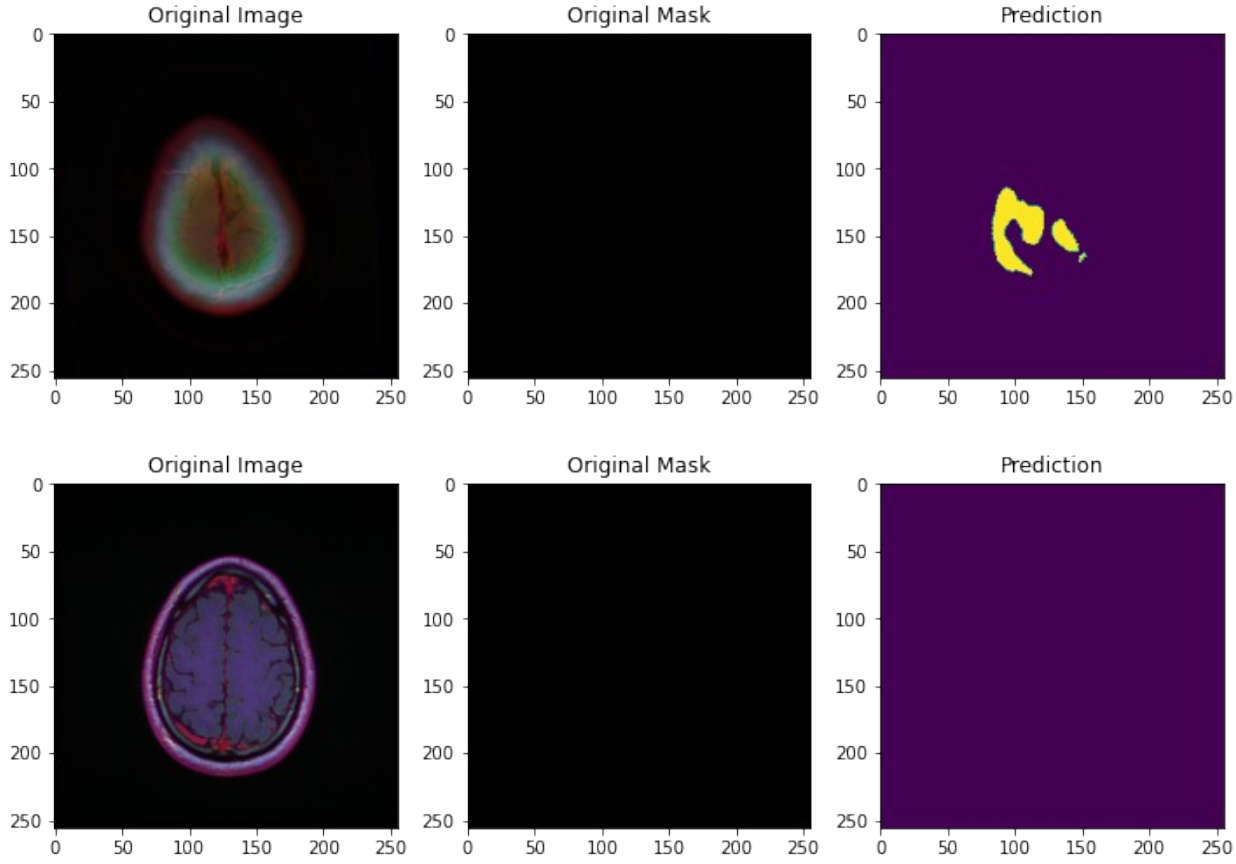
```
test_gen = train_generator(df_test, BATCH_SIZE,
                                dict(),
                                target_size=IMAGE_SIZE)
results = model.evaluate(test_gen, steps=len(df_test) / BATCH_SIZE)
print("Test IOU: ",results[1])
print("Test Dice Coefficent: ",results[2])
```

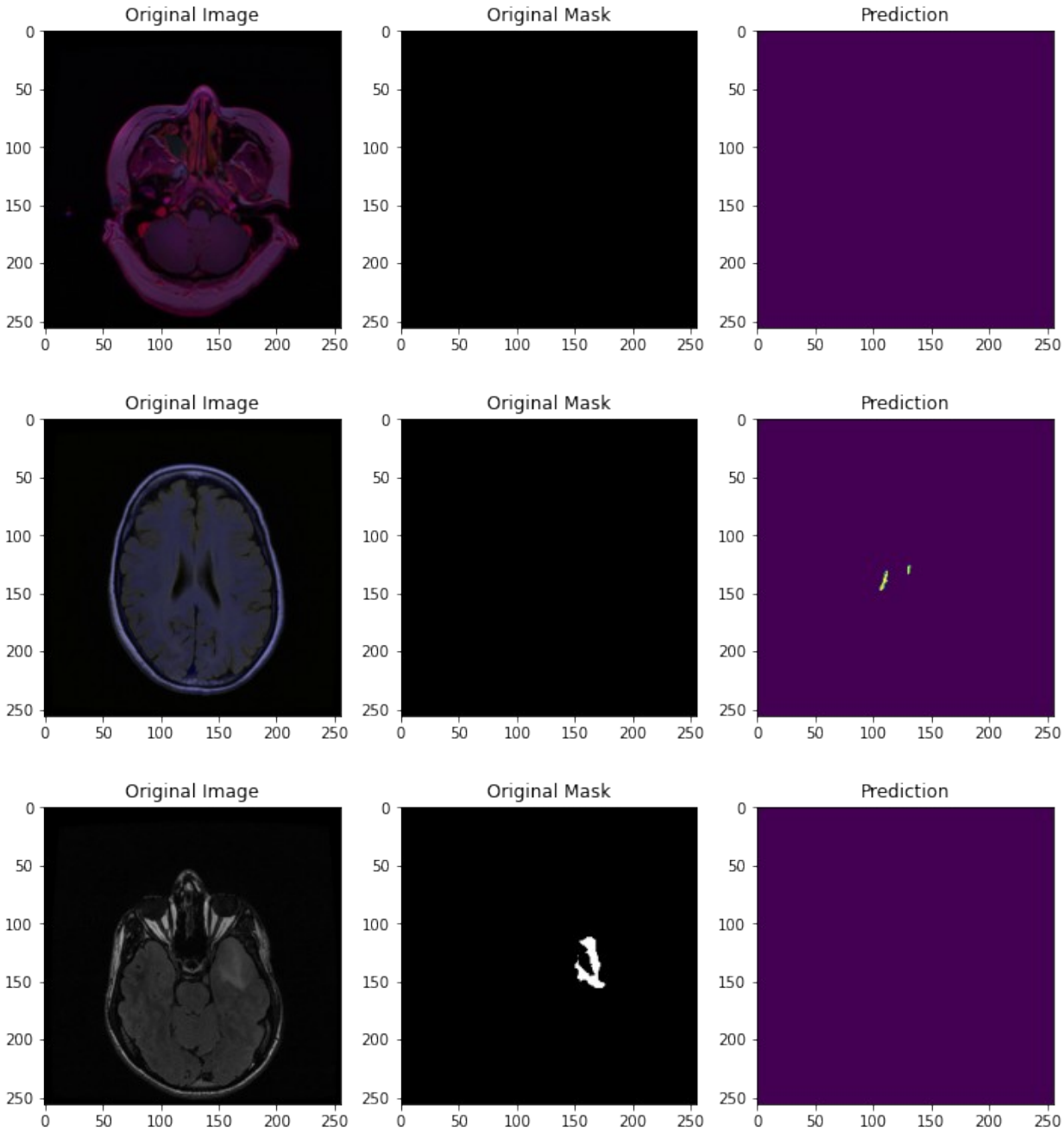
```
Found 590 validated image filenames.
Found 590 validated image filenames.
18/18 [==============================] - 9s 487ms/step - loss: 0.3254
- iou: 0.5431 - dice_coef: 0.7001
Test IOU:  0.5431138277053833
Test Dice Coefficent:  0.7001265287399292
```

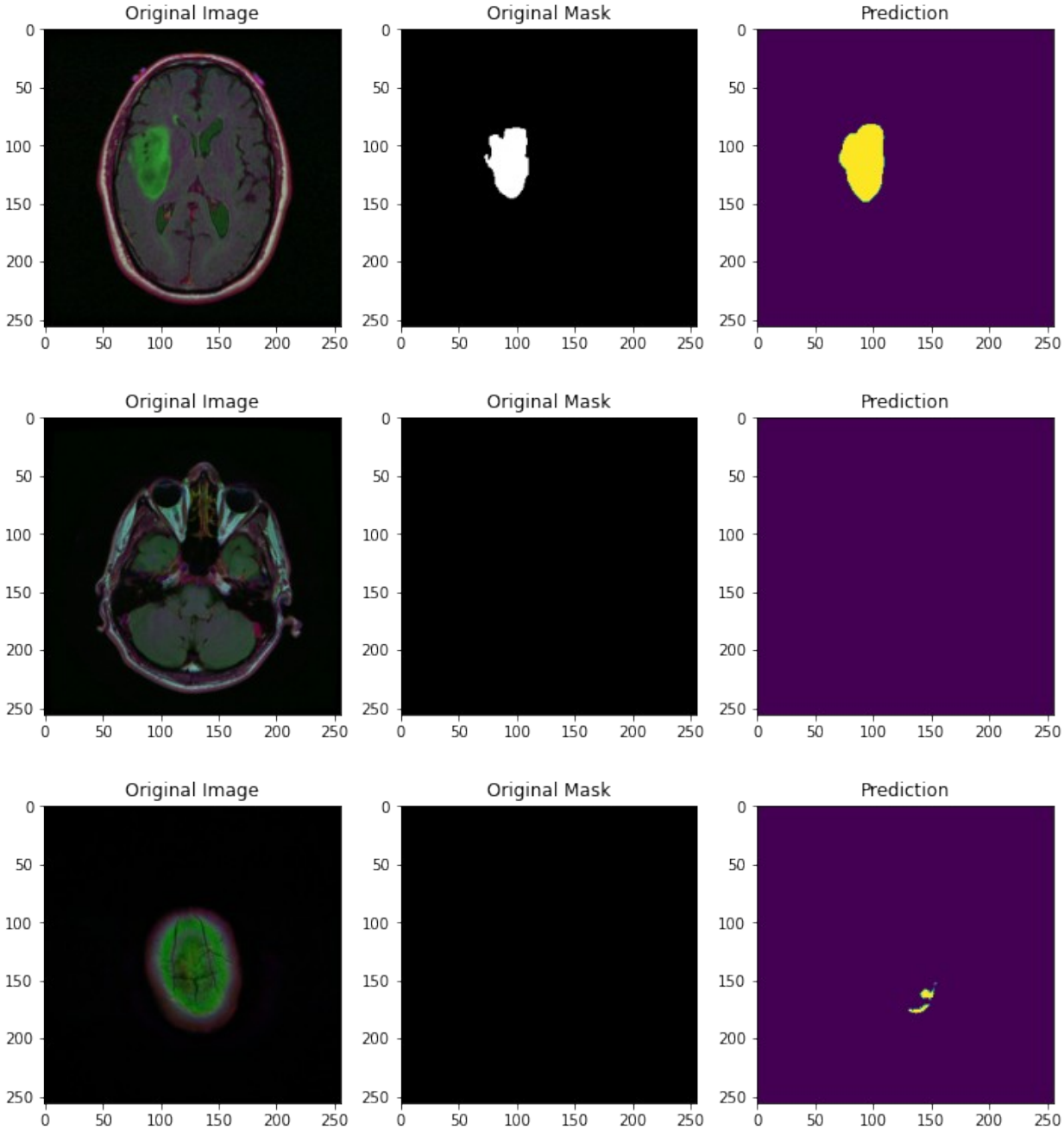
Visualize the Result

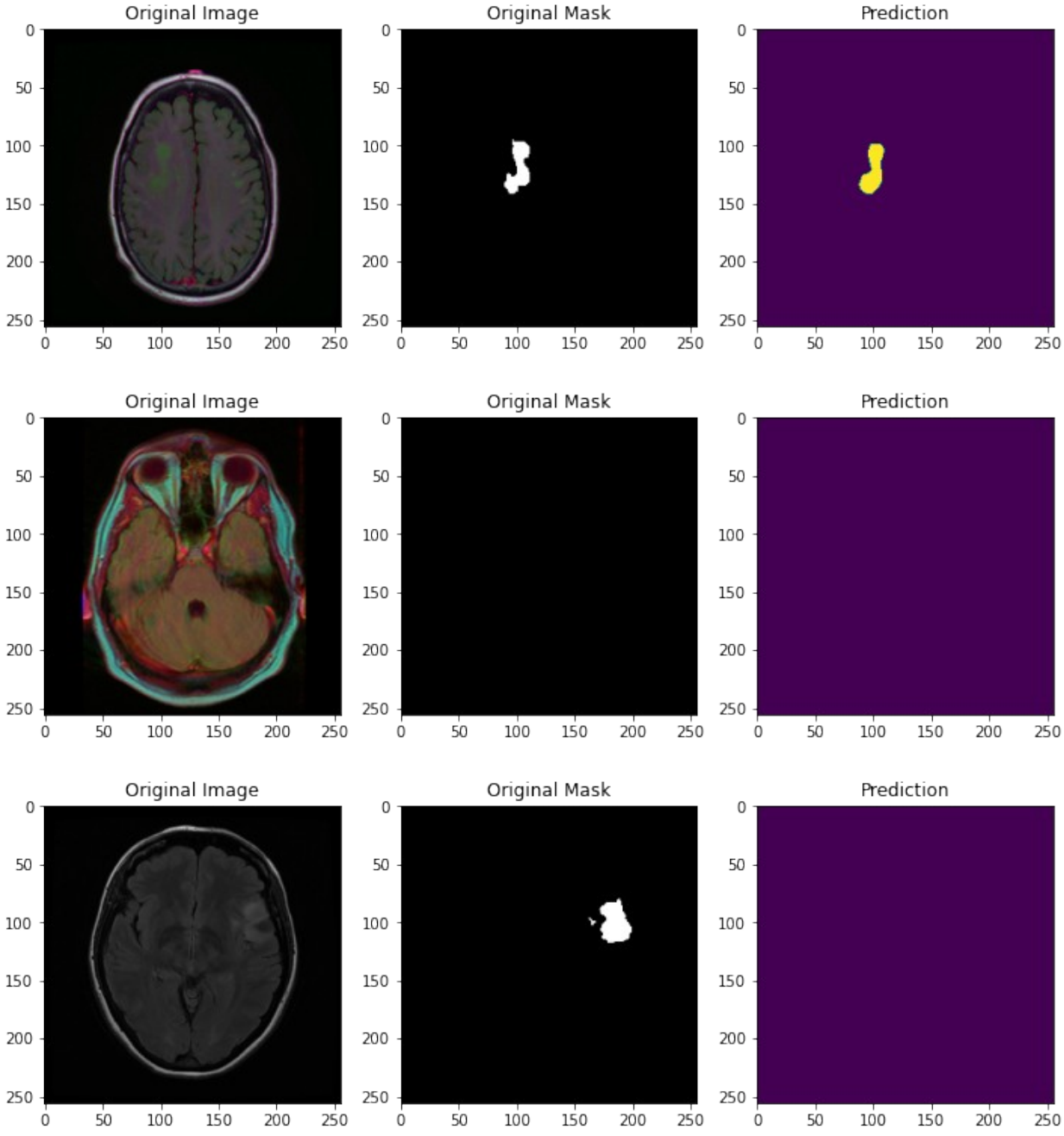
```
for i in range(30):
    index=np.random.randint(1,len(df_test.index))
    img = cv2.imread(df_test['image_path'].iloc[index])
    img = cv2.resize(img ,IMAGE_SIZE)
    img = img / 255
    img = img[np.newaxis, :, :, :]
    pred=model.predict(img)

    plt.figure(figsize=(12,12))
```

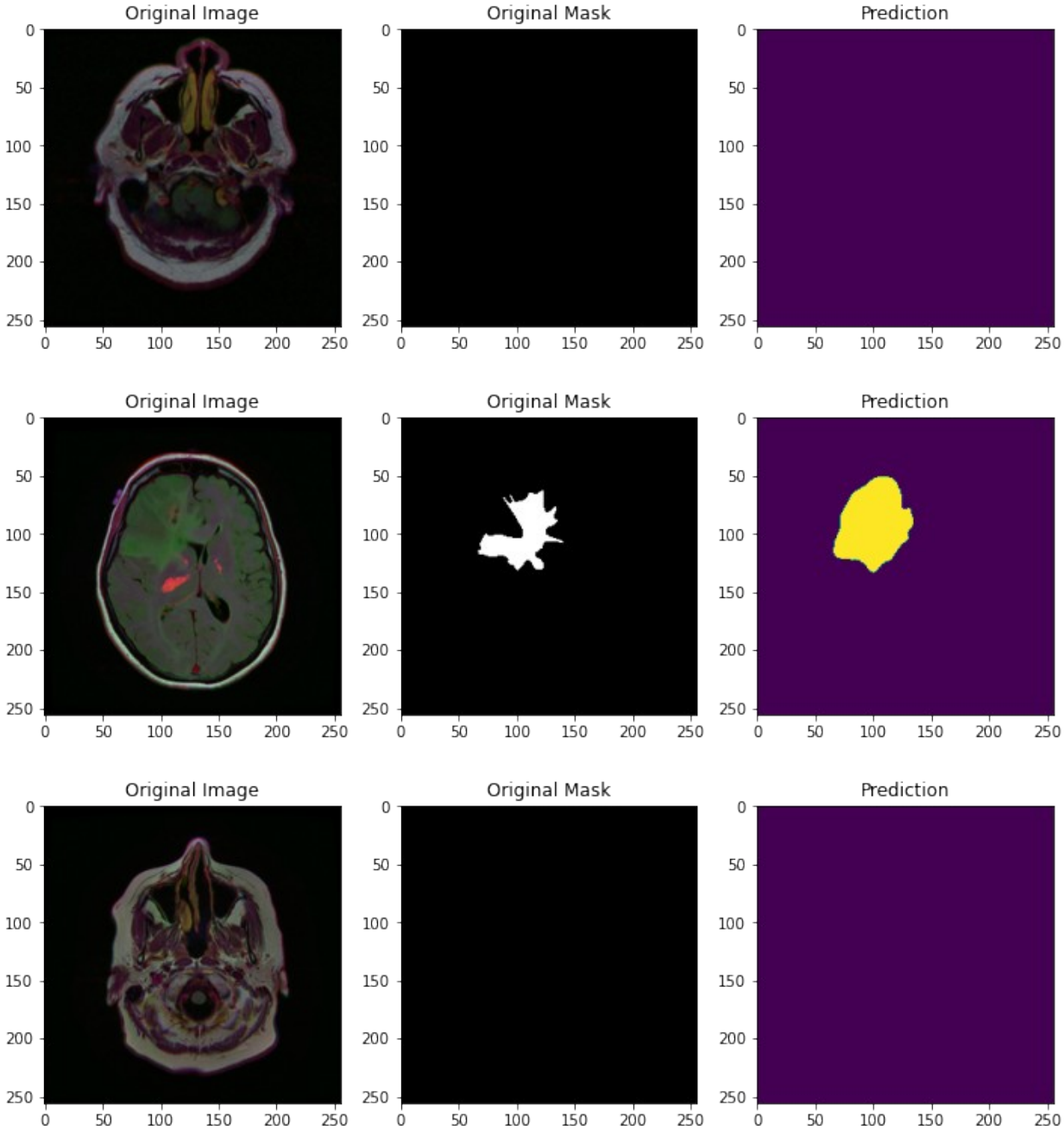
```
    plt.subplot(1,3,1)
    plt.imshow(np.squeeze(img))
    plt.title('Original Image')
    plt.subplot(1,3,2)

plt.imshow(np.squeeze(cv2.imread(df_test['mask_path'].iloc[index])))
    plt.title('Original Mask')
    plt.subplot(1,3,3)
    plt.imshow(np.squeeze(pred) > .5)
    plt.title('Prediction')
    plt.show()
```

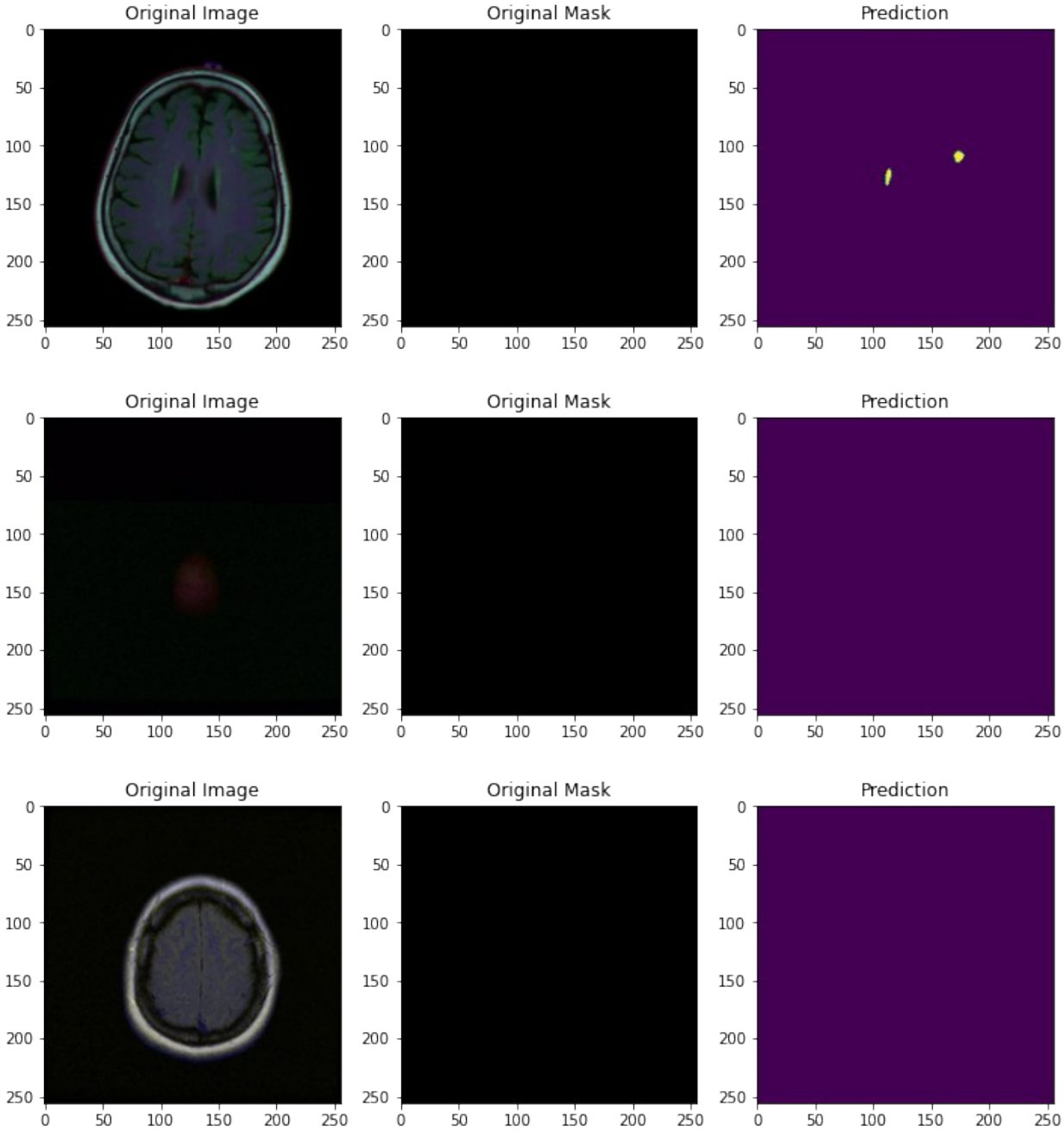


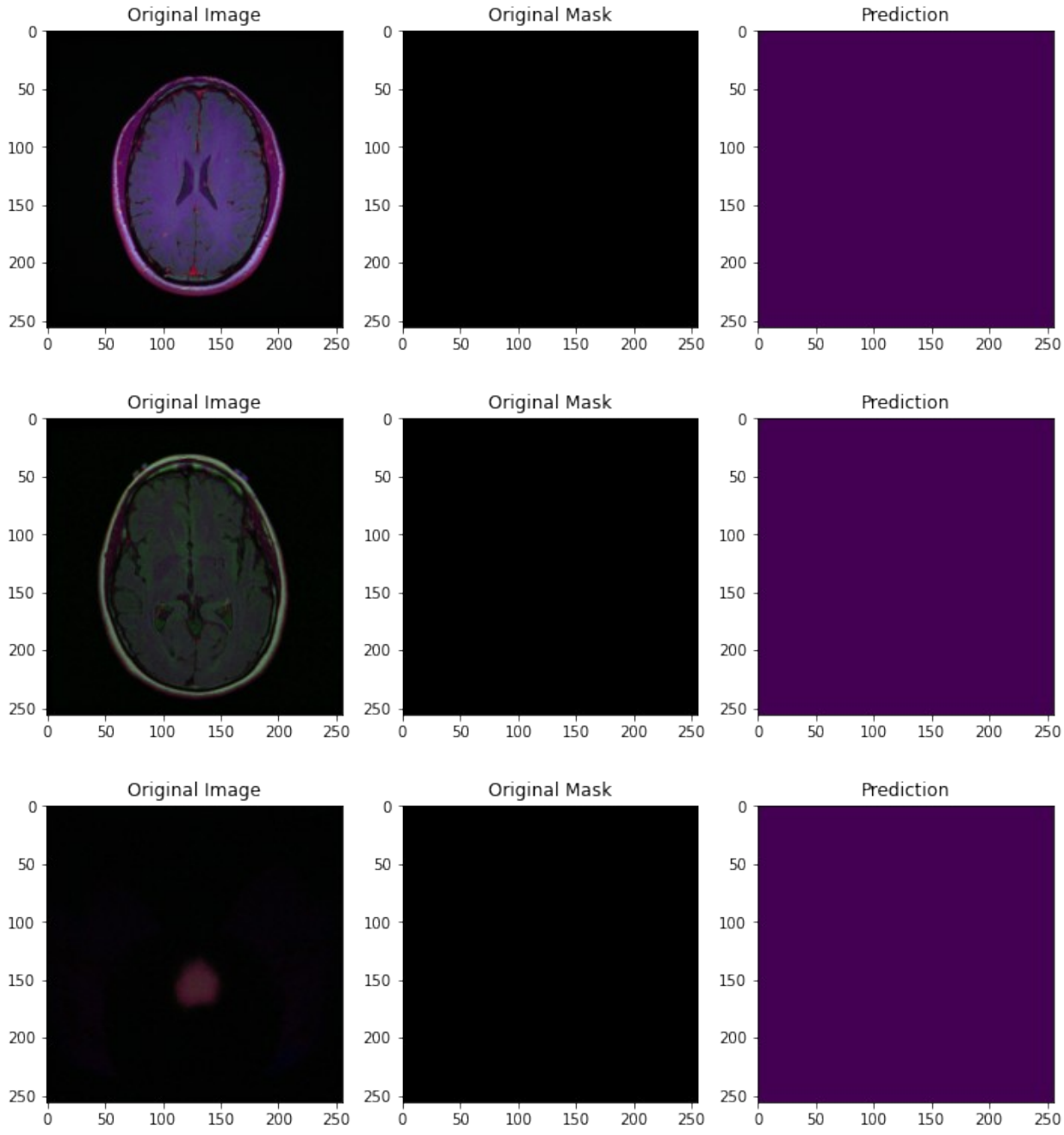

Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction

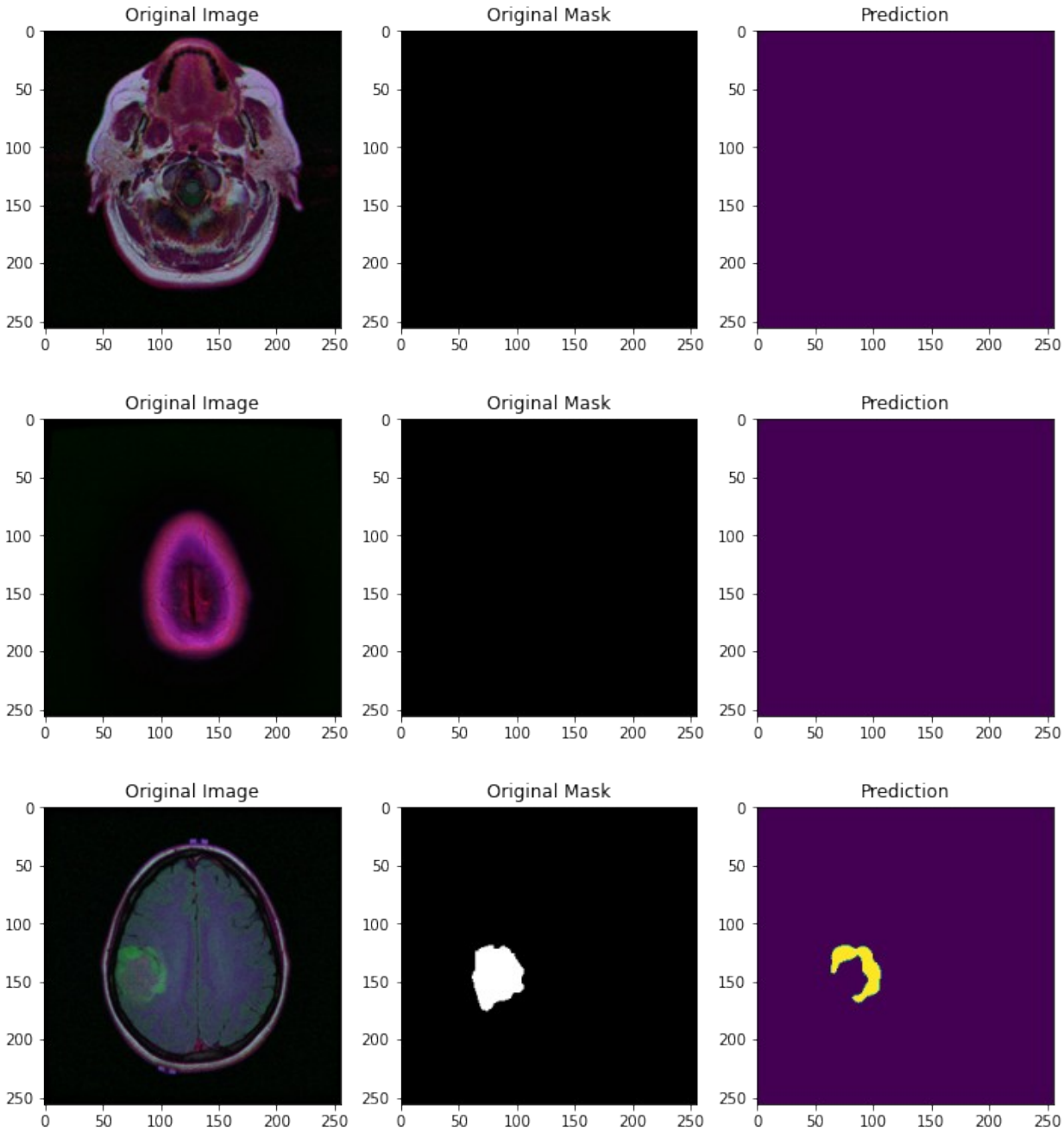

Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction

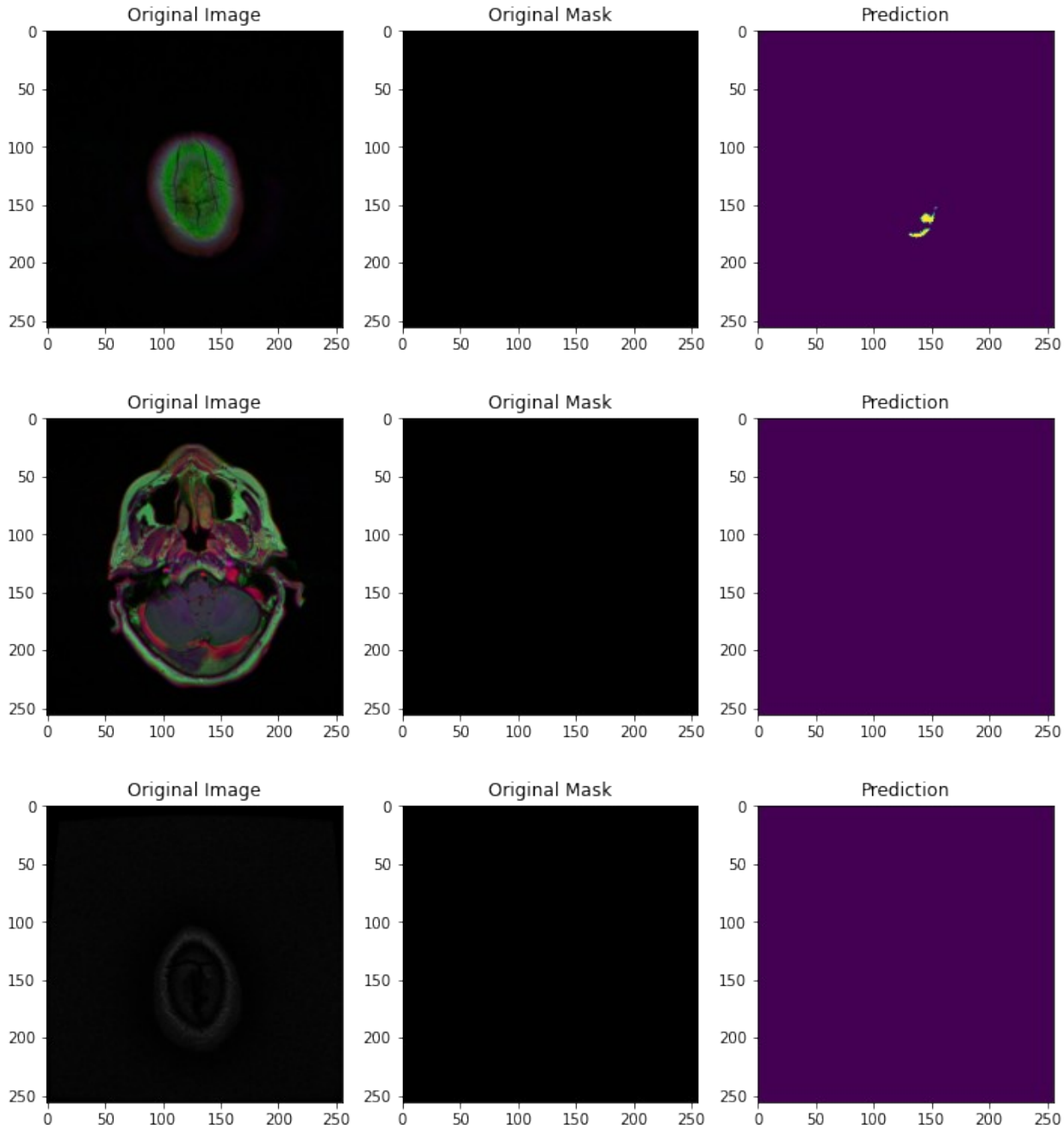

Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction

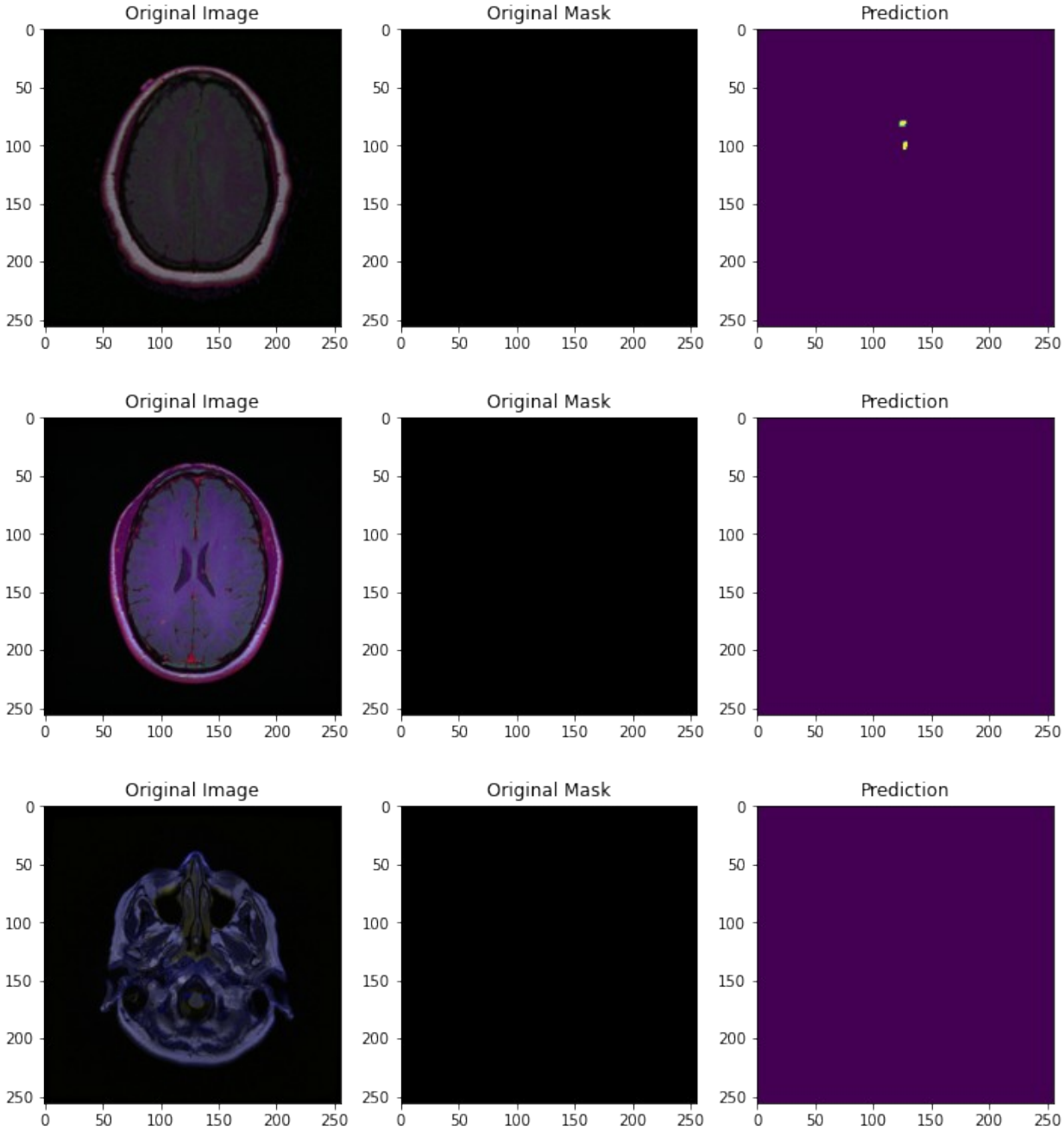

Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction

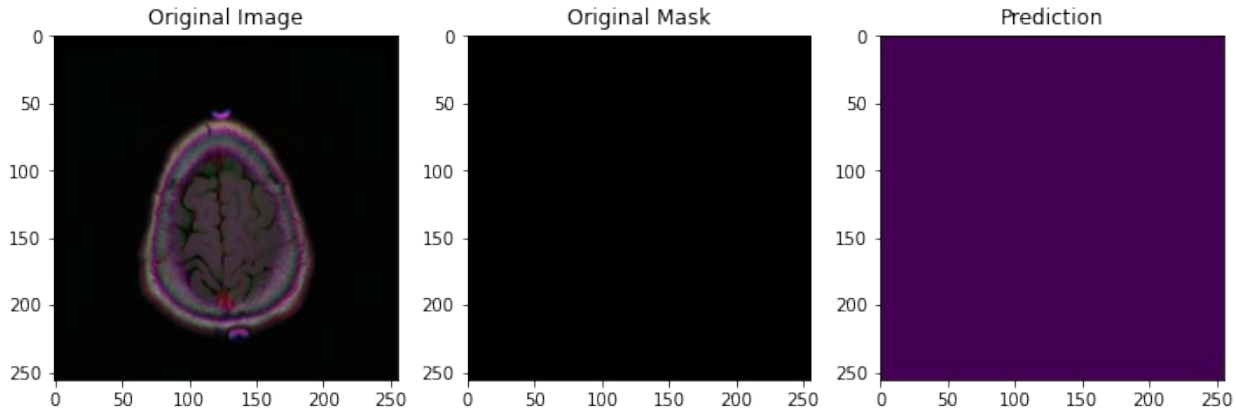

Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction


Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction


Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction


Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction
0
50
100
150
200
250


Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction
Original Image
Original Mask
Prediction




```
pred = np.where(pred > 0.5 , 1 , 0)
pred
```

```
array([[[[0],
         [0],
         [0],
         ...,
         [0],
         [0],
         [0]],

        [[0],
         [0],
         [0],
         ...,
         [0],
         [0],
         [0]],

        [[0],
         [0],
         [0],
         ...,
         [0],
         [0],
         [0]],

        ...,

        [[0],
         [0],
         [0],
         ...,
         [0],
         [0],
         [0]],
```

```
        [[0],
         [0],
         [0],
         ...,
         [0],
         [0],
         [0]],

        [[0],
         [0],
         [0],
         ...,
         [0],
         [0],
         [0]]]])
```

```
pred = np.squeeze(pred) > .5
pred
```

```
array([[False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       ...,
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False]])
```

```
cor , incor = 0 , 0
for i in range(len(df_test.index)):
    index=i
    img = cv2.imread(df_test['image_path'].iloc[index])
    img = cv2.resize(img ,IMAGE_SIZE)
    img = img / 255
    img = img[np.newaxis, :, :, :]
    pred = (np.squeeze(model.predict(img)) > 0.5)
    mask = np.squeeze(cv2.imread(df_test['mask_path'].iloc[index]))
    print(f"Prediction {i}: " , pred.any() == mask.any())
    if pred.any() == mask.any():
        cor += 1
    else:
        incor += 1
```

```
Prediction 0:  True
Prediction 1:  True
Prediction 2:  False
Prediction 3:  False
Prediction 4:  True
Prediction 5:  True
Prediction 6:  True
Prediction 7:  True
```

```
Prediction 8:  True
Prediction 9:  True
Prediction 10:  True
Prediction 11:  False
Prediction 12:  True
Prediction 13:  True
Prediction 14:  True
Prediction 15:  True
Prediction 16:  True
Prediction 17:  False
Prediction 18:  True
Prediction 19:  True
Prediction 20:  True
Prediction 21:  True
Prediction 22:  True
Prediction 23:  True
Prediction 24:  True
Prediction 25:  True
Prediction 26:  True
Prediction 27:  True
Prediction 28:  True
Prediction 29:  True
Prediction 30:  True
Prediction 31:  True
Prediction 32:  True
Prediction 33:  True
Prediction 34:  True
Prediction 35:  True
Prediction 36:  True
Prediction 37:  True
Prediction 38:  True
Prediction 39:  False
Prediction 40:  True
Prediction 41:  True
Prediction 42:  True
Prediction 43:  True
Prediction 44:  False
Prediction 45:  True
Prediction 46:  False
Prediction 47:  True
Prediction 48:  True
Prediction 49:  True
Prediction 50:  False
Prediction 51:  False
Prediction 52:  True
Prediction 53:  False
Prediction 54:  True
Prediction 55:  True
Prediction 56:  False
```

```
Prediction 57:  True
Prediction 58:  True
Prediction 59:  True
Prediction 60:  True
Prediction 61:  True
Prediction 62:  True
Prediction 63:  True
Prediction 64:  True
Prediction 65:  True
Prediction 66:  True
Prediction 67:  True
Prediction 68:  False
Prediction 69:  True
Prediction 70:  True
Prediction 71:  True
Prediction 72:  True
Prediction 73:  True
Prediction 74:  True
Prediction 75:  True
Prediction 76:  True
Prediction 77:  True
Prediction 78:  True
Prediction 79:  True
Prediction 80:  True
Prediction 81:  True
Prediction 82:  True
Prediction 83:  True
Prediction 84:  True
Prediction 85:  True
Prediction 86:  True
Prediction 87:  False
Prediction 88:  True
Prediction 89:  True
Prediction 90:  True
Prediction 91:  True
Prediction 92:  True
Prediction 93:  False
Prediction 94:  True
Prediction 95:  True
Prediction 96:  True
Prediction 97:  True
Prediction 98:  False
Prediction 99:  True
Prediction 100:  True
Prediction 101:  True
Prediction 102:  True
Prediction 103:  True
Prediction 104:  True
Prediction 105:  True
```

```
Prediction 106:  True
Prediction 107:  True
Prediction 108:  True
Prediction 109:  True
Prediction 110:  True
Prediction 111:  True
Prediction 112:  True
Prediction 113:  True
Prediction 114:  False
Prediction 115:  True
Prediction 116:  True
Prediction 117:  True
Prediction 118:  True
Prediction 119:  True
Prediction 120:  False
Prediction 121:  True
Prediction 122:  True
Prediction 123:  True
Prediction 124:  True
Prediction 125:  True
Prediction 126:  True
Prediction 127:  True
Prediction 128:  True
Prediction 129:  False
Prediction 130:  True
Prediction 131:  True
Prediction 132:  True
Prediction 133:  True
Prediction 134:  True
Prediction 135:  True
Prediction 136:  True
Prediction 137:  True
Prediction 138:  True
Prediction 139:  True
Prediction 140:  True
Prediction 141:  True
Prediction 142:  False
Prediction 143:  True
Prediction 144:  True
Prediction 145:  True
Prediction 146:  True
Prediction 147:  True
Prediction 148:  False
Prediction 149:  False
Prediction 150:  True
Prediction 151:  True
Prediction 152:  True
Prediction 153:  True
Prediction 154:  True
```

```
Prediction 155:  True
Prediction 156:  True
Prediction 157:  True
Prediction 158:  True
Prediction 159:  True
Prediction 160:  True
Prediction 161:  True
Prediction 162:  True
Prediction 163:  True
Prediction 164:  True
Prediction 165:  True
Prediction 166:  True
Prediction 167:  True
Prediction 168:  True
Prediction 169:  True
Prediction 170:  True
Prediction 171:  True
Prediction 172:  True
Prediction 173:  True
Prediction 174:  True
Prediction 175:  True
Prediction 176:  True
Prediction 177:  True
Prediction 178:  True
Prediction 179:  True
Prediction 180:  True
Prediction 181:  False
Prediction 182:  True
Prediction 183:  True
Prediction 184:  True
Prediction 185:  True
Prediction 186:  True
Prediction 187:  True
Prediction 188:  True
Prediction 189:  True
Prediction 190:  True
Prediction 191:  True
Prediction 192:  True
Prediction 193:  True
Prediction 194:  True
Prediction 195:  False
Prediction 196:  True
Prediction 197:  True
Prediction 198:  True
Prediction 199:  True
Prediction 200:  False
Prediction 201:  True
Prediction 202:  True
Prediction 203:  True
```

```
Prediction 204:  False
Prediction 205:  False
Prediction 206:  True
Prediction 207:  True
Prediction 208:  True
Prediction 209:  True
Prediction 210:  True
Prediction 211:  True
Prediction 212:  True
Prediction 213:  False
Prediction 214:  True
Prediction 215:  True
Prediction 216:  True
Prediction 217:  True
Prediction 218:  True
Prediction 219:  True
Prediction 220:  True
Prediction 221:  True
Prediction 222:  True
Prediction 223:  True
Prediction 224:  True
Prediction 225:  False
Prediction 226:  False
Prediction 227:  True
Prediction 228:  True
Prediction 229:  True
Prediction 230:  True
Prediction 231:  False
Prediction 232:  False
Prediction 233:  False
Prediction 234:  True
Prediction 235:  False
Prediction 236:  True
Prediction 237:  True
Prediction 238:  True
Prediction 239:  True
Prediction 240:  True
Prediction 241:  True
Prediction 242:  True
Prediction 243:  True
Prediction 244:  True
Prediction 245:  True
Prediction 246:  True
Prediction 247:  True
Prediction 248:  True
Prediction 249:  True
Prediction 250:  True
Prediction 251:  True
Prediction 252:  True
```

```
Prediction 253:  False
Prediction 254:  True
Prediction 255:  True
Prediction 256:  True
Prediction 257:  True
Prediction 258:  True
Prediction 259:  True
Prediction 260:  True
Prediction 261:  False
Prediction 262:  False
Prediction 263:  True
Prediction 264:  True
Prediction 265:  False
Prediction 266:  True
Prediction 267:  False
Prediction 268:  True
Prediction 269:  True
Prediction 270:  True
Prediction 271:  True
Prediction 272:  True
Prediction 273:  True
Prediction 274:  True
Prediction 275:  False
Prediction 276:  True
Prediction 277:  True
Prediction 278:  True
Prediction 279:  True
Prediction 280:  True
Prediction 281:  True
Prediction 282:  True
Prediction 283:  False
Prediction 284:  True
Prediction 285:  False
Prediction 286:  True
Prediction 287:  True
Prediction 288:  True
Prediction 289:  True
Prediction 290:  True
Prediction 291:  True
Prediction 292:  True
Prediction 293:  True
Prediction 294:  True
Prediction 295:  True
Prediction 296:  False
Prediction 297:  True
Prediction 298:  True
Prediction 299:  True
Prediction 300:  True
Prediction 301:  True
```

```
Prediction 302:  True
Prediction 303:  True
Prediction 304:  True
Prediction 305:  True
Prediction 306:  True
Prediction 307:  False
Prediction 308:  True
Prediction 309:  True
Prediction 310:  True
Prediction 311:  True
Prediction 312:  True
Prediction 313:  True
Prediction 314:  True
Prediction 315:  True
Prediction 316:  False
Prediction 317:  True
Prediction 318:  True
Prediction 319:  True
Prediction 320:  True
Prediction 321:  True
Prediction 322:  True
Prediction 323:  True
Prediction 324:  True
Prediction 325:  True
Prediction 326:  True
Prediction 327:  True
Prediction 328:  False
Prediction 329:  True
Prediction 330:  True
Prediction 331:  True
Prediction 332:  True
Prediction 333:  True
Prediction 334:  True
Prediction 335:  True
Prediction 336:  True
Prediction 337:  True
Prediction 338:  True
Prediction 339:  False
Prediction 340:  True
Prediction 341:  True
Prediction 342:  True
Prediction 343:  True
Prediction 344:  True
Prediction 345:  False
Prediction 346:  True
Prediction 347:  True
Prediction 348:  True
Prediction 349:  True
Prediction 350:  True
```

```
Prediction 351:  True
Prediction 352:  True
Prediction 353:  True
Prediction 354:  True
Prediction 355:  True
Prediction 356:  True
Prediction 357:  True
Prediction 358:  True
Prediction 359:  True
Prediction 360:  True
Prediction 361:  True
Prediction 362:  True
Prediction 363:  True
Prediction 364:  True
Prediction 365:  True
Prediction 366:  True
Prediction 367:  True
Prediction 368:  True
Prediction 369:  True
Prediction 370:  False
Prediction 371:  False
Prediction 372:  True
Prediction 373:  True
Prediction 374:  True
Prediction 375:  True
Prediction 376:  True
Prediction 377:  True
Prediction 378:  True
Prediction 379:  True
Prediction 380:  True
Prediction 381:  True
Prediction 382:  True
Prediction 383:  True
Prediction 384:  True
Prediction 385:  True
Prediction 386:  True
Prediction 387:  True
Prediction 388:  True
Prediction 389:  True
Prediction 390:  False
Prediction 391:  True
Prediction 392:  True
Prediction 393:  True
Prediction 394:  True
Prediction 395:  True
Prediction 396:  True
Prediction 397:  True
Prediction 398:  True
Prediction 399:  True
```

```
Prediction 400:  False
Prediction 401:  True
Prediction 402:  True
Prediction 403:  True
Prediction 404:  True
Prediction 405:  True
Prediction 406:  True
Prediction 407:  True
Prediction 408:  True
Prediction 409:  True
Prediction 410:  True
Prediction 411:  True
Prediction 412:  True
Prediction 413:  True
Prediction 414:  True
Prediction 415:  True
Prediction 416:  False
Prediction 417:  True
Prediction 418:  True
Prediction 419:  True
Prediction 420:  True
Prediction 421:  True
Prediction 422:  False
Prediction 423:  True
Prediction 424:  True
Prediction 425:  True
Prediction 426:  True
Prediction 427:  True
Prediction 428:  True
Prediction 429:  True
Prediction 430:  True
Prediction 431:  True
Prediction 432:  True
Prediction 433:  True
Prediction 434:  False
Prediction 435:  True
Prediction 436:  True
Prediction 437:  True
Prediction 438:  True
Prediction 439:  True
Prediction 440:  True
Prediction 441:  False
Prediction 442:  True
Prediction 443:  True
Prediction 444:  True
Prediction 445:  True
Prediction 446:  True
Prediction 447:  True
Prediction 448:  True
```

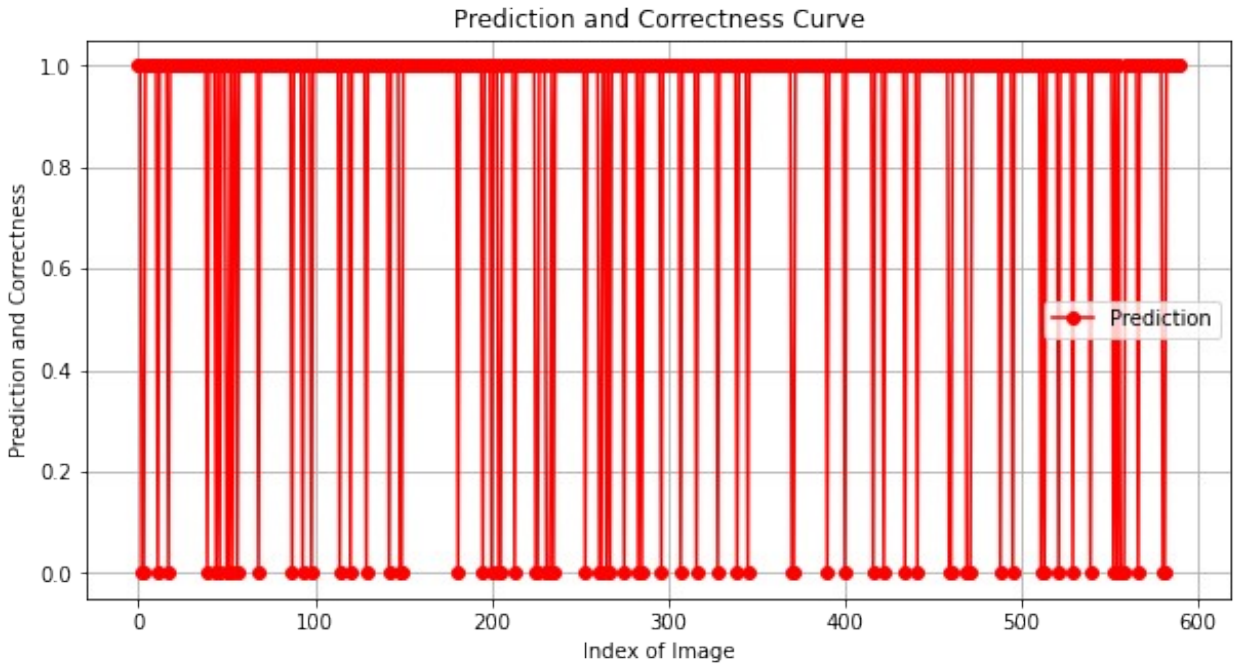
```
Prediction 449:  True
Prediction 450:  True
Prediction 451:  True
Prediction 452:  True
Prediction 453:  True
Prediction 454:  True
Prediction 455:  True
Prediction 456:  True
Prediction 457:  True
Prediction 458:  True
Prediction 459:  False
Prediction 460:  False
Prediction 461:  True
Prediction 462:  True
Prediction 463:  True
Prediction 464:  True
Prediction 465:  True
Prediction 466:  True
Prediction 467:  True
Prediction 468:  True
Prediction 469:  False
Prediction 470:  False
Prediction 471:  False
Prediction 472:  True
Prediction 473:  True
Prediction 474:  True
Prediction 475:  True
Prediction 476:  True
Prediction 477:  True
Prediction 478:  True
Prediction 479:  True
Prediction 480:  True
Prediction 481:  True
Prediction 482:  True
Prediction 483:  True
Prediction 484:  True
Prediction 485:  True
Prediction 486:  True
Prediction 487:  True
Prediction 488:  False
Prediction 489:  True
Prediction 490:  True
Prediction 491:  True
Prediction 492:  True
Prediction 493:  True
Prediction 494:  True
Prediction 495:  False
Prediction 496:  True
Prediction 497:  True
Prediction 498:  True
```

```
Prediction 499:  True
Prediction 500:  True
Prediction 501:  True
Prediction 502:  True
Prediction 503:  True
Prediction 504:  True
Prediction 505:  True
Prediction 506:  True
Prediction 507:  True
Prediction 508:  True
Prediction 509:  True
Prediction 510:  True
Prediction 511:  False
Prediction 512:  True
Prediction 513:  False
Prediction 514:  True
Prediction 515:  True
Prediction 516:  True
Prediction 517:  True
Prediction 518:  True
Prediction 519:  True
Prediction 520:  True
Prediction 521:  False
Prediction 522:  True
Prediction 523:  True
Prediction 524:  True
Prediction 525:  True
Prediction 526:  True
Prediction 527:  True
Prediction 528:  True
Prediction 529:  False
Prediction 530:  True
Prediction 531:  True
Prediction 532:  True
Prediction 533:  True
Prediction 534:  True
Prediction 535:  True
Prediction 536:  True
Prediction 537:  True
Prediction 538:  True
Prediction 539:  False
Prediction 540:  True
Prediction 541:  True
Prediction 542:  True
Prediction 543:  True
Prediction 544:  True
Prediction 545:  True
Prediction 546:  True
Prediction 547:  True
```

```
Prediction 548:  True
Prediction 549:  True
Prediction 550:  True
Prediction 551:  True
Prediction 552:  False
Prediction 553:  True
Prediction 554:  False
Prediction 555:  True
Prediction 556:  False
Prediction 557:  False
Prediction 558:  False
Prediction 559:  True
Prediction 560:  True
Prediction 561:  True
Prediction 562:  True
Prediction 563:  True
Prediction 564:  True
Prediction 565:  True
Prediction 566:  False
Prediction 567:  True
Prediction 568:  True
Prediction 569:  True
Prediction 570:  True
Prediction 571:  True
Prediction 572:  True
Prediction 573:  True
Prediction 574:  True
Prediction 575:  True
Prediction 576:  True
Prediction 577:  True
Prediction 578:  True
Prediction 579:  True
Prediction 580:  False
Prediction 581:  False
Prediction 582:  True
Prediction 583:  True
Prediction 584:  True
Prediction 585:  True
Prediction 586:  True
Prediction 587:  True
Prediction 588:  True
Prediction 589:  True
```

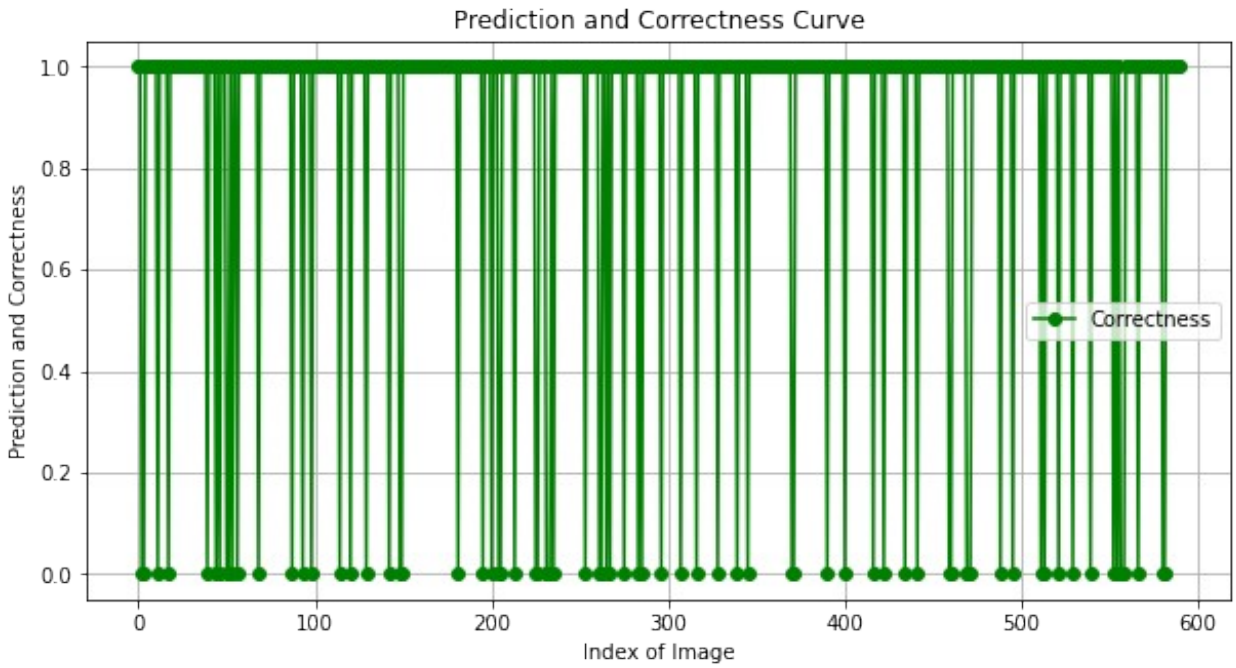
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Initialize lists to store prediction results and correctness
prediction_results = []
correctness = []
```

```
# Assuming df_test is your DataFrame containing image paths and mask
paths
for i in range(len(df_test.index)):
    index = i
    img = cv2.imread(df_test['image_path'].iloc[index])
    img = cv2.resize(img, IMAGE_SIZE)
    img = img / 255
    img = img[np.newaxis, :, :, :]
    pred = (np.squeeze(model.predict(img)) > 0.5)
    mask = np.squeeze(cv2.imread(df_test['mask_path'].iloc[index]))

    # Check if prediction matches the ground truth mask
    prediction_correct = pred.any() == mask.any()

    # Append prediction result and correctness
    prediction_results.append(prediction_correct)
    correctness.append(1 if prediction_correct else 0)

# Plotting the curve
plt.figure(figsize=(10, 5))
plt.plot(prediction_results, label='Prediction', marker='o',
linestyle='-', color='red')  # False predictions in red
#plt.plot(correctness, label='Correctness', marker='o', linestyle='-',
color='green')      # Correct predictions in green
plt.xlabel('Index of Image')
plt.ylabel('Prediction and Correctness')
plt.title('Prediction and Correctness Curve')
plt.legend()
plt.grid(True)
plt.show()
```



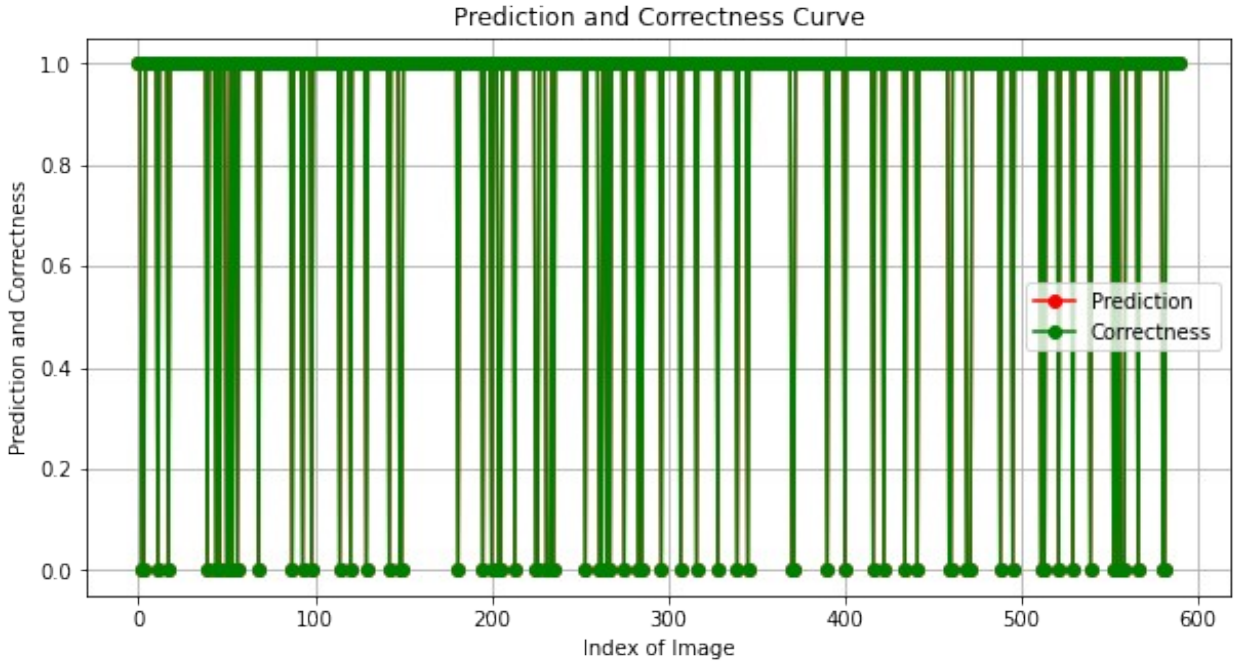

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Initialize lists to store prediction results and correctness
prediction_results = []
correctness = []

# Assuming df_test is your DataFrame containing image paths and mask
paths
for i in range(len(df_test.index)):
    index = i
    img = cv2.imread(df_test['image_path'].iloc[index])
    img = cv2.resize(img, IMAGE_SIZE)
    img = img / 255
    img = img[np.newaxis, :, :, :]
    pred = (np.squeeze(model.predict(img)) > 0.5)
    mask = np.squeeze(cv2.imread(df_test['mask_path'].iloc[index]))

    # Check if prediction matches the ground truth mask
    prediction_correct = pred.any() == mask.any()

    # Append prediction result and correctness
    prediction_results.append(prediction_correct)
    correctness.append(1 if prediction_correct else 0)

# Plotting the curve
plt.figure(figsize=(10, 5))
#plt.plot(prediction_results, label='Prediction', marker='o',
```

```
linestyle='-', color='red')  # False predictions in red
plt.plot(correctness, label='Correctness', marker='o', linestyle='-',
color='green')      # Correct predictions in green
plt.xlabel('Index of Image')
plt.ylabel('Prediction and Correctness')
plt.title('Prediction and Correctness Curve')
plt.legend()
plt.grid(True)
plt.show()
```




```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Initialize lists to store prediction results and correctness
prediction_results = []
correctness = []

# Assuming df_test is your DataFrame containing image paths and mask
paths
for i in range(len(df_test.index)):
    index = i
    img = cv2.imread(df_test['image_path'].iloc[index])
    img = cv2.resize(img, IMAGE_SIZE)
    img = img / 255
    img = img[np.newaxis, :, :, :]
    pred = (np.squeeze(model.predict(img)) > 0.5)
    mask = np.squeeze(cv2.imread(df_test['mask_path'].iloc[index]))
```

```
    # Check if prediction matches the ground truth mask
    prediction_correct = pred.any() == mask.any()

    # Append prediction result and correctness
    prediction_results.append(prediction_correct)
    correctness.append(1 if prediction_correct else 0)

# Plotting the curve
plt.figure(figsize=(10, 5))
plt.plot(prediction_results, label='Prediction', marker='o',
linestyle='-', color='red')  # False predictions in red
plt.plot(correctness, label='Correctness', marker='o', linestyle='-',
color='green')      # Correct predictions in green
plt.xlabel('Index of Image')
plt.ylabel('Prediction and Correctness')
plt.title('Prediction and Correctness Curve')
plt.legend()
plt.grid(True)
plt.show()
```



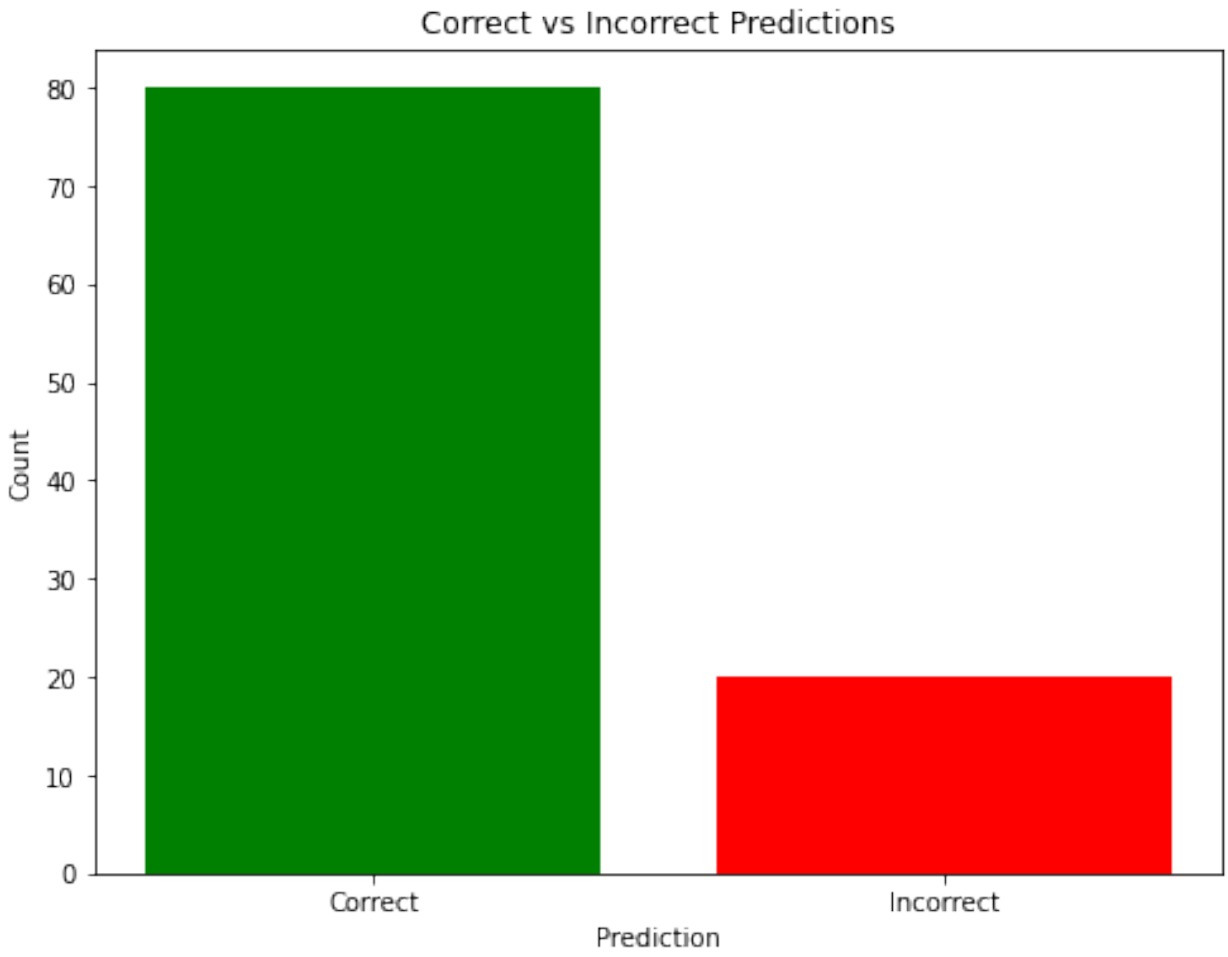

```
print("Correct = " , cor , "Incorrect = " , incor )
print("Accuracy = " , (cor/(cor+incor)) * 100)
```

```
Correct =  515 Incorrect =  75
Accuracy =  87.28813559322035
```

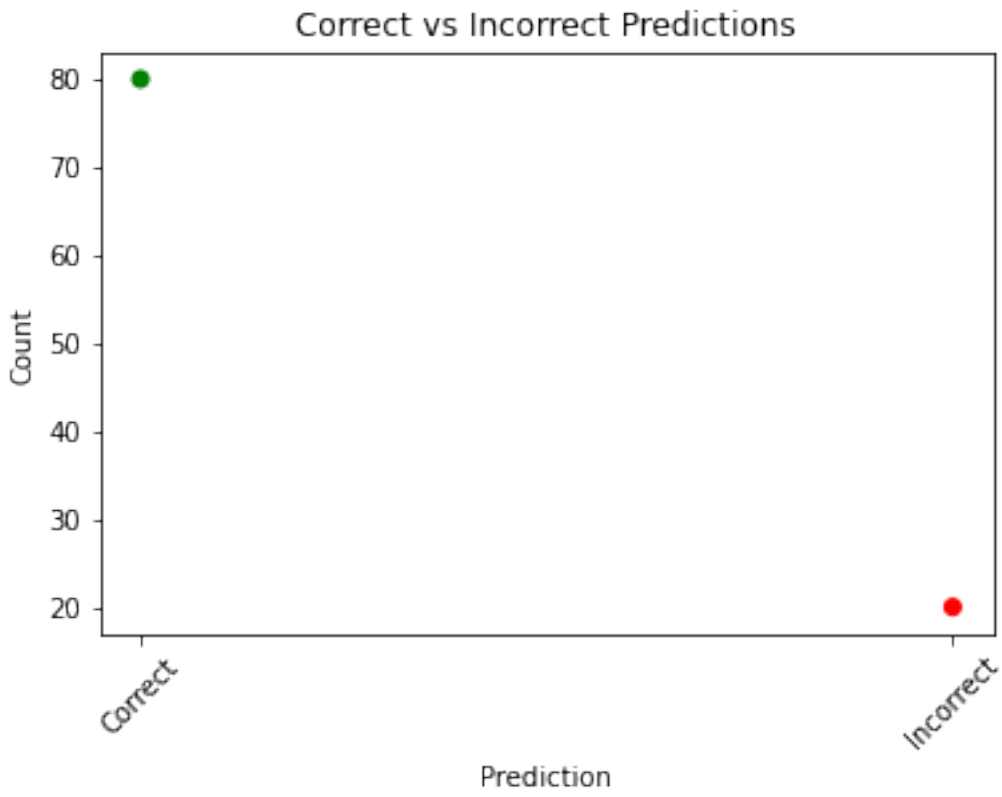
```
import matplotlib.pyplot as plt
```

```
# Assuming cor and incor are the counts of correct and incorrect
predictions, respectively
cor = 80  # Example value
incor = 20  # Example value

# Calculate accuracy
accuracy = (cor / (cor + incor)) * 100

# Print correct and incorrect counts and accuracy
print("Correct =", cor, "Incorrect =", incor)
print("Accuracy =", accuracy)

# Plot a curve
# Here, you can plot any curve you want, for example, a bar plot
showing correct and incorrect counts
plt.figure(figsize=(8, 6))
plt.bar(["Correct", "Incorrect"], [cor, incor], color=['green',
'red'])
plt.xlabel('Prediction')
plt.ylabel('Count')
plt.title('Correct vs Incorrect Predictions')
plt.show()
```

```
Correct = 80 Incorrect = 20
Accuracy = 80.0
```




```
import matplotlib.pyplot as plt

# Assuming cor and incor are initialized earlier
cor, incor = 80, 20  # Example values

# Plotting the scatter plot
plt.figure(figsize=(6, 4))
plt.scatter(["Correct", "Incorrect"], [cor, incor], color=['green',
'red'])
plt.xlabel('Prediction')
plt.ylabel('Count')
plt.title('Correct vs Incorrect Predictions')
plt.xticks(rotation=45)
plt.show()

# Print correct and incorrect counts and accuracy
print("Correct =", cor, "Incorrect =", incor)
print("Accuracy =", (cor / (cor + incor)) * 100)
```




```
Correct = 80 Incorrect = 20
Accuracy = 80.0
```