In [1]:

```
from glob import glob
import pandas as pd
from sklearn.model_selection import train_test_split
import os
import cv2
import random
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from skimage import color, io
from keras_preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import *
from tensorflow.keras import layers, optimizers
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K
from tensorflow.keras.optimizers import Adam
```

In [2]:

```
# ## Obtain all folders (without README and .csv) and keep them in order by starting wi

brain_imgs = []
brain_masks = glob('archive/lgg-mri-segmentation/kaggle_3m/*/*_mask*')

for file in brain_masks:
    brain_imgs.append(file.replace('_mask',''))
```

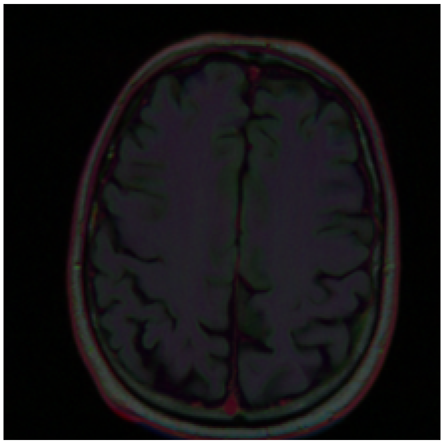
In [3]:

```
has_tumor = []

for mask in brain_masks:
    tumor = np.max(cv2.imread(mask))
    if tumor == 0:
        has_tumor.append(0)
    else:
        has_tumor.append(1)

brain_df = pd.DataFrame(data={"Image File": brain_imgs, "Mask File": brain_masks, "Tumo
```

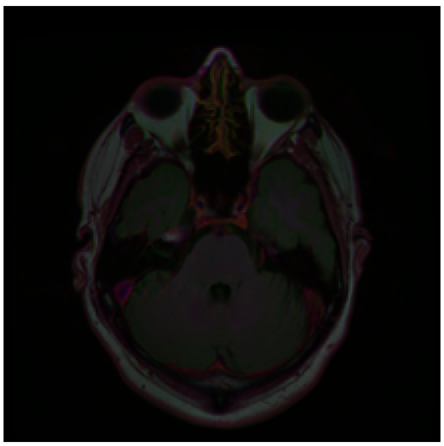
In [4]:

```
for x in range(4):
    i = random.randint(0, len(brain_df)) # select a random index
    plt.figure(x)
    image = cv2.imread(brain_df["Image File"][i])
    mask = cv2.imread(brain_df["Mask File"][i])
    if brain_df['Tumor Detected'][i] == 1:
        indices = np.where(mask==255)
        mask[indices[0], indices[1], :] = [0, 255, 0]
    plt.imshow(image)
    plt.imshow(mask,cmap='jet',alpha=0.5)
    plt.title("Tumor Present: "+ str(brain_df['Tumor Detected'][i]))
    plt.axis('off')
```

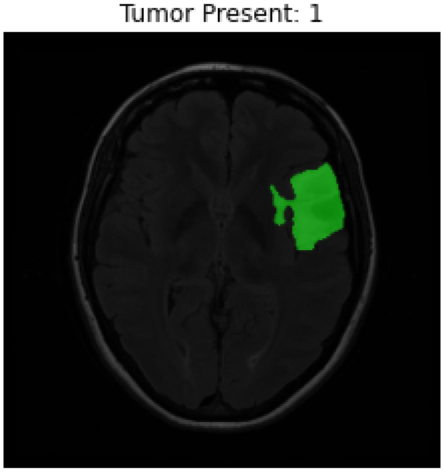
Tumor Present: 0



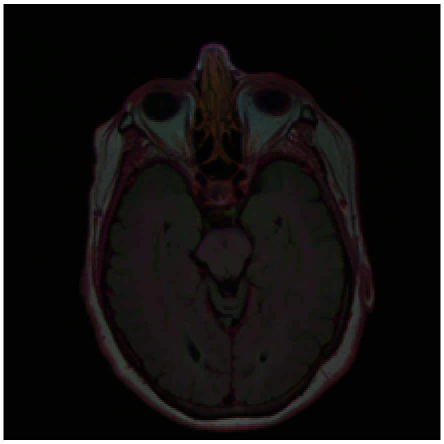
Tumor Present: 0



Tumor Present: 1



Tumor Present: 0



In [5]:
```
brain_train, brain_test = train_test_split(brain_df, test_size = 0.15)
brain_train, brain_val = train_test_split(brain_train, test_size = 0.2)
```

In [6]:
```
print(brain_train.shape)
print(brain_test.shape)
print(brain_val.shape)
```

```
(2671, 3)
(590, 3)
(668, 3)
```

In [7]:
```
#Source for UNet paramaters: https://github.com/zhixuhao/unet/blob/master/data.py
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):
    '''
    can generate image and mask at the same time use the same seed for
    image_datagen and mask_datagen to ensure the transformation for image
    and mask is the same if you want to visualize the results of generator,
    set save_to_dir = "your path"
    '''
    image_datagen = ImageDataGenerator(**aug_dict)
    mask_datagen = ImageDataGenerator(**aug_dict)

    image_generator = image_datagen.flow_from_dataframe(
        data_frame,
        x_col = "Image File",
        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 File",
        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)
```

In [8]:

```
# Loss functions sourced from https://github.com/nabsabraham/focal-tversky-unet/blob/ma

smooth = 100

def dsc(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    score = (2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) + smooth)
    return score

def dice_loss(y_true, y_pred):
    loss = 1 - dsc(y_true, y_pred)
    return loss

def tversky(y_true, y_pred):
    y_true_pos = K.flatten(y_true)
    y_pred_pos = K.flatten(y_pred)
    true_pos = K.sum(y_true_pos * y_pred_pos)
    false_neg = K.sum(y_true_pos * (1-y_pred_pos))
    false_pos = K.sum((1-y_true_pos)*y_pred_pos)
    alpha = 0.7
    return (true_pos + smooth)/(true_pos + alpha*false_neg + (1-alpha)*false_pos + smoo

def tversky_loss(y_true, y_pred):
    return 1 - tversky(y_true,y_pred)

def focal_tversky(y_true,y_pred):
    pt_1 = tversky(y_true, y_pred)
    gamma = 0.75
    return K.pow((1-pt_1), gamma)

def iou(y_true, y_pred):
    intersection = K.sum(y_true * y_pred)
```

```python
    sum_ = K.sum(y_true + y_pred)
    jac = (intersection + smooth) / (sum_ - intersection + smooth)
    return jac
```

In [9]:

```python
##Unet Model from https://github.com/zhixuhao/unet/blob/master/model.py

def unet(input_size):
    inputs = Input(input_size)
    conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = '
    conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = '
    batch1 = BatchNormalization(axis=3)(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(batch1)

    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
    batch2 = BatchNormalization(axis=3)(conv2)
    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
    batch2 = BatchNormalization(axis=3)(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(batch2)

    conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
    batch3 = BatchNormalization(axis=3)(conv3)
    conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
    batch3 = BatchNormalization(axis=3)(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(batch3)

    conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer =
    batch4 = BatchNormalization(axis=3)(conv4)
    conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer =
    batch4 = BatchNormalization(axis=3)(conv4)
    drop4 = Dropout(0.5)(batch4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)

    conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer =
    batch5 = BatchNormalization(axis=3)(conv5)
    conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer =
    batch5 = BatchNormalization(axis=3)(conv5)
    drop5 = Dropout(0.5)(batch5)

    up6 = Conv2DTranspose(512, 2, strides=(2,2),activation = 'relu', padding = 'same',
    merge6 = concatenate([drop4,up6], axis = 3)
    conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer =
    conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer =
    batch6 = BatchNormalization(axis=3)(conv6)

    up7 = Conv2DTranspose(256, 2, strides=(2,2), activation = 'relu', padding = 'same',
    merge7 = concatenate([conv3,up7], axis = 3)
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
    batch7 = BatchNormalization(axis=3)(conv7)

    up8 = Conv2DTranspose(128, 2, strides=(2,2), activation = 'relu', padding = 'same',
    merge8 = concatenate([conv2,up8], axis = 3)
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
    batch8 = BatchNormalization(axis=3)(conv8)

    up9 = Conv2DTranspose(64, 2, strides=(2,2), activation = 'relu', padding = 'same',
    merge9 = concatenate([conv1,up9], axis = 3)
```

```
    conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = '
    conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = '
    batch9 = BatchNormalization(axis=3)(conv9)
    conv10 = Conv2D(1, 1, activation = 'sigmoid')(batch9)

    model = Model(inputs = inputs, outputs = conv10,name="unet")

    return model
```

In [10]:

```
train_generator_args = dict(rotation_range=0.2,
                            width_shift_range=0.05,
                            height_shift_range=0.05,
                            shear_range=0.05,
                            zoom_range=0.05,
                            horizontal_flip=True,
                            fill_mode='nearest')
train_gen = train_generator(brain_train[["Image File","Mask File"]], 32,
                                train_generator_args,
                                target_size=(64, 64))

test_gen = train_generator(brain_test[["Image File","Mask File"]], 32,
                                dict(),
                                target_size=(64, 64))

val_gen = train_generator(brain_val[["Image File","Mask File"]], 32,
                                dict(),
                                target_size=(64, 64))

model = unet((64, 64, 3))
model.summary()
```

```
Model: "unet"
__________________________________________________________________________________________
__________
 Layer (type)                   Output Shape         Param #     Connected to
==========================================================================================
==========
 input_1 (InputLayer)           [(None, 64, 64, 3)]  0           []

 conv2d (Conv2D)                (None, 64, 64, 64)   1792        ['input_1[0][0]']

 conv2d_1 (Conv2D)              (None, 64, 64, 64)   36928       ['conv2d[0][0]']

 batch_normalization (BatchNorm  (None, 64, 64, 64)  256         ['conv2d_1[0][0]']
 alization)

 max_pooling2d (MaxPooling2D)   (None, 32, 32, 64)   0           ['batch_normalization
[0][0]']

 conv2d_2 (Conv2D)              (None, 32, 32, 128)  73856       ['max_pooling2d[0][0]']

 batch_normalization_1 (BatchNo  (None, 32, 32, 128)  512        ['conv2d_2[0][0]']
 rmalization)

 conv2d_3 (Conv2D)              (None, 32, 32, 128)  147584      ['batch_normalization_1
[0][0]']

 batch_normalization_2 (BatchNo  (None, 32, 32, 128)  512        ['conv2d_3[0][0]']
```

```
 rmalization)

 max_pooling2d_1 (MaxPooling2D)  (None, 16, 16, 128)  0           ['batch_normalization_2
[0][0]']

 conv2d_4 (Conv2D)              (None, 16, 16, 256)  295168      ['max_pooling2d_1[0]
[0]']

 batch_normalization_3 (BatchNo  (None, 16, 16, 256)  1024        ['conv2d_4[0][0]']
 rmalization)

 conv2d_5 (Conv2D)              (None, 16, 16, 256)  590080      ['batch_normalization_3
[0][0]']

 batch_normalization_4 (BatchNo  (None, 16, 16, 256)  1024        ['conv2d_5[0][0]']
 rmalization)

 max_pooling2d_2 (MaxPooling2D)  (None, 8, 8, 256)   0           ['batch_normalization_4
[0][0]']

 conv2d_6 (Conv2D)              (None, 8, 8, 512)    1180160     ['max_pooling2d_2[0]
[0]']

 batch_normalization_5 (BatchNo  (None, 8, 8, 512)   2048        ['conv2d_6[0][0]']
 rmalization)

 conv2d_7 (Conv2D)              (None, 8, 8, 512)    2359808     ['batch_normalization_5
[0][0]']

 batch_normalization_6 (BatchNo  (None, 8, 8, 512)   2048        ['conv2d_7[0][0]']
 rmalization)

 dropout (Dropout)              (None, 8, 8, 512)    0           ['batch_normalization_6
[0][0]']

 max_pooling2d_3 (MaxPooling2D)  (None, 4, 4, 512)   0           ['dropout[0][0]']

 conv2d_8 (Conv2D)              (None, 4, 4, 1024)   4719616     ['max_pooling2d_3[0]
[0]']

 batch_normalization_7 (BatchNo  (None, 4, 4, 1024)  4096        ['conv2d_8[0][0]']
 rmalization)

 conv2d_9 (Conv2D)              (None, 4, 4, 1024)   9438208     ['batch_normalization_7
[0][0]']

 batch_normalization_8 (BatchNo  (None, 4, 4, 1024)  4096        ['conv2d_9[0][0]']
 rmalization)

 dropout_1 (Dropout)            (None, 4, 4, 1024)   0           ['batch_normalization_8
[0][0]']

 conv2d_transpose (Conv2DTransp  (None, 8, 8, 512)   2097664     ['dropout_1[0][0]']
 ose)

 concatenate (Concatenate)      (None, 8, 8, 1024)   0           ['dropout[0][0]',
                                                                  'conv2d_transpose[0]
[0]']

 conv2d_10 (Conv2D)             (None, 8, 8, 512)    4719104     ['concatenate[0][0]']
```

```
 conv2d_11 (Conv2D)             (None, 8, 8, 512)    2359808     ['conv2d_10[0][0]']

 batch_normalization_9 (BatchNo  (None, 8, 8, 512)   2048        ['conv2d_11[0][0]']
 rmalization)

 conv2d_transpose_1 (Conv2DTran  (None, 16, 16, 256)  524544     ['batch_normalization_9
[0][0]']
 spose)

 concatenate_1 (Concatenate)    (None, 16, 16, 512)  0           ['conv2d_5[0][0]',
                                                                  'conv2d_transpose_1[0]
[0]']

 conv2d_12 (Conv2D)             (None, 16, 16, 256)  1179904     ['concatenate_1[0][0]']

 conv2d_13 (Conv2D)             (None, 16, 16, 256)  590080      ['conv2d_12[0][0]']

 batch_normalization_10 (BatchN  (None, 16, 16, 256)  1024       ['conv2d_13[0][0]']
 ormalization)

 conv2d_transpose_2 (Conv2DTran  (None, 32, 32, 128)  131200     ['batch_normalization_1
0[0][0]']
 spose)

 concatenate_2 (Concatenate)    (None, 32, 32, 256)  0           ['conv2d_3[0][0]',
                                                                  'conv2d_transpose_2[0]
[0]']

 conv2d_14 (Conv2D)             (None, 32, 32, 128)  295040      ['concatenate_2[0][0]']

 conv2d_15 (Conv2D)             (None, 32, 32, 128)  147584      ['conv2d_14[0][0]']

 batch_normalization_11 (BatchN  (None, 32, 32, 128)  512        ['conv2d_15[0][0]']
 ormalization)

 conv2d_transpose_3 (Conv2DTran  (None, 64, 64, 64)  32832       ['batch_normalization_1
1[0][0]']
 spose)

 concatenate_3 (Concatenate)    (None, 64, 64, 128)  0           ['conv2d_1[0][0]',
                                                                  'conv2d_transpose_3[0]
[0]']

 conv2d_16 (Conv2D)             (None, 64, 64, 64)   73792       ['concatenate_3[0][0]']

 conv2d_17 (Conv2D)             (None, 64, 64, 64)   36928       ['conv2d_16[0][0]']

 batch_normalization_12 (BatchN  (None, 64, 64, 64)  256         ['conv2d_17[0][0]']
 ormalization)

 conv2d_18 (Conv2D)             (None, 64, 64, 1)    65          ['batch_normalization_1
2[0][0]']

==========================================================================================
==========
Total params: 31,051,201
Trainable params: 31,041,473
Non-trainable params: 9,728
```

In [11]:

```
K.clear_session()
adam = Adam(learning_rate = 0.01,epsilon=0.1)
model.compile(optimizer = adam,
                  loss = [dice_loss,focal_tversky],
                  metrics = ["binary_accuracy", dsc,tversky,iou]
                 )
```

In [12]:

```
history = model.fit(train_gen,
                 steps_per_epoch=len(brain_train)/128,
                 epochs = 60,
                 validation_data = val_gen,
                 validation_steps = len(brain_val)/128
                 )
```

```
Found 2671 validated image filenames.
Found 2671 validated image filenames.
Epoch 1/60
21/20 [==============================] - ETA: 0s - loss: 0.9796 - binary_accuracy: 0.525
3 - dsc: 0.0204 - tversky: 0.0355 - iou: 0.0111Found 668 validated image filenames.
Found 668 validated image filenames.
20/20 [==============================] - 130s 5s/step - loss: 0.9796 - binary_accuracy:
0.5253 - dsc: 0.0204 - tversky: 0.0355 - iou: 0.0111 - val_loss: 0.9770 - val_binary_acc
uracy: 0.8768 - val_dsc: 0.0230 - val_tversky: 0.0395 - val_iou: 0.0124
Epoch 2/60
20/20 [==============================] - 71s 3s/step - loss: 0.9751 - binary_accuracy:
0.5315 - dsc: 0.0249 - tversky: 0.0426 - iou: 0.0134 - val_loss: 0.9781 - val_binary_acc
uracy: 0.8420 - val_dsc: 0.0219 - val_tversky: 0.0378 - val_iou: 0.0119
Epoch 3/60
20/20 [==============================] - 52s 2s/step - loss: 0.9698 - binary_accuracy:
0.5408 - dsc: 0.0302 - tversky: 0.0510 - iou: 0.0161 - val_loss: 0.9800 - val_binary_acc
uracy: 0.7512 - val_dsc: 0.0200 - val_tversky: 0.0349 - val_iou: 0.0109
Epoch 4/60
20/20 [==============================] - 60s 3s/step - loss: 0.9688 - binary_accuracy:
0.5501 - dsc: 0.0315 - tversky: 0.0534 - iou: 0.0168 - val_loss: 0.9691 - val_binary_acc
uracy: 0.6818 - val_dsc: 0.0309 - val_tversky: 0.0518 - val_iou: 0.0165
Epoch 5/60
20/20 [==============================] - 52s 3s/step - loss: 0.9615 - binary_accuracy:
0.5578 - dsc: 0.0385 - tversky: 0.0645 - iou: 0.0204 - val_loss: 0.9752 - val_binary_acc
uracy: 0.6563 - val_dsc: 0.0248 - val_tversky: 0.0427 - val_iou: 0.0134
Epoch 6/60
20/20 [==============================] - 49s 2s/step - loss: 0.9604 - binary_accuracy:
0.5711 - dsc: 0.0396 - tversky: 0.0664 - iou: 0.0210 - val_loss: 0.9767 - val_binary_acc
uracy: 0.6255 - val_dsc: 0.0233 - val_tversky: 0.0402 - val_iou: 0.0126
Epoch 7/60
20/20 [==============================] - 42s 2s/step - loss: 0.9582 - binary_accuracy:
0.5868 - dsc: 0.0418 - tversky: 0.0698 - iou: 0.0222 - val_loss: 0.9716 - val_binary_acc
uracy: 0.6325 - val_dsc: 0.0284 - val_tversky: 0.0484 - val_iou: 0.0152
Epoch 8/60
20/20 [==============================] - 46s 2s/step - loss: 0.9586 - binary_accuracy:
0.6032 - dsc: 0.0421 - tversky: 0.0705 - iou: 0.0224 - val_loss: 0.9644 - val_binary_acc
uracy: 0.6053 - val_dsc: 0.0356 - val_tversky: 0.0599 - val_iou: 0.0189
Epoch 9/60
```

```
20/20 [==============================] - 52s 3s/step - loss: 0.9545 - binary_accuracy:
0.6175 - dsc: 0.0455 - tversky: 0.0758 - iou: 0.0242 - val_loss: 0.9585 - val_binary_acc
uracy: 0.5993 - val_dsc: 0.0415 - val_tversky: 0.0692 - val_iou: 0.0220
Epoch 10/60
20/20 [==============================] - 51s 2s/step - loss: 0.9547 - binary_accuracy:
0.6354 - dsc: 0.0453 - tversky: 0.0757 - iou: 0.0241 - val_loss: 0.9661 - val_binary_acc
uracy: 0.5930 - val_dsc: 0.0339 - val_tversky: 0.0574 - val_iou: 0.0180
Epoch 11/60
20/20 [==============================] - 79s 4s/step - loss: 0.9550 - binary_accuracy:
0.6531 - dsc: 0.0450 - tversky: 0.0753 - iou: 0.0240 - val_loss: 0.9527 - val_binary_acc
uracy: 0.6069 - val_dsc: 0.0473 - val_tversky: 0.0783 - val_iou: 0.0250
Epoch 12/60
20/20 [==============================] - 96s 5s/step - loss: 0.9436 - binary_accuracy:
0.6813 - dsc: 0.0571 - tversky: 0.0943 - iou: 0.0305 - val_loss: 0.9562 - val_binary_acc
uracy: 0.6109 - val_dsc: 0.0438 - val_tversky: 0.0732 - val_iou: 0.0232
Epoch 13/60
20/20 [==============================] - 93s 4s/step - loss: 0.9432 - binary_accuracy:
0.7096 - dsc: 0.0568 - tversky: 0.0937 - iou: 0.0303 - val_loss: 0.9514 - val_binary_acc
uracy: 0.6152 - val_dsc: 0.0486 - val_tversky: 0.0807 - val_iou: 0.0257
Epoch 14/60
20/20 [==============================] - 51s 2s/step - loss: 0.9400 - binary_accuracy:
0.7371 - dsc: 0.0600 - tversky: 0.0989 - iou: 0.0321 - val_loss: 0.9543 - val_binary_acc
uracy: 0.6354 - val_dsc: 0.0457 - val_tversky: 0.0761 - val_iou: 0.0243
Epoch 15/60
20/20 [==============================] - 51s 2s/step - loss: 0.9386 - binary_accuracy:
0.7625 - dsc: 0.0614 - tversky: 0.1013 - iou: 0.0329 - val_loss: 0.9540 - val_binary_acc
uracy: 0.6674 - val_dsc: 0.0460 - val_tversky: 0.0768 - val_iou: 0.0245
Epoch 16/60
20/20 [==============================] - 51s 2s/step - loss: 0.9349 - binary_accuracy:
0.7825 - dsc: 0.0639 - tversky: 0.1055 - iou: 0.0345 - val_loss: 0.9464 - val_binary_acc
uracy: 0.6961 - val_dsc: 0.0536 - val_tversky: 0.0889 - val_iou: 0.0287
Epoch 17/60
20/20 [==============================] - 51s 2s/step - loss: 0.9326 - binary_accuracy:
0.8009 - dsc: 0.0674 - tversky: 0.1108 - iou: 0.0363 - val_loss: 0.9307 - val_binary_acc
uracy: 0.7262 - val_dsc: 0.0693 - val_tversky: 0.1133 - val_iou: 0.0370
Epoch 18/60
20/20 [==============================] - 51s 2s/step - loss: 0.9208 - binary_accuracy:
0.8153 - dsc: 0.0792 - tversky: 0.1285 - iou: 0.0428 - val_loss: 0.9336 - val_binary_acc
uracy: 0.7726 - val_dsc: 0.0664 - val_tversky: 0.1094 - val_iou: 0.0357
Epoch 19/60
20/20 [==============================] - 51s 2s/step - loss: 0.9235 - binary_accuracy:
0.8317 - dsc: 0.0765 - tversky: 0.1249 - iou: 0.0415 - val_loss: 0.9057 - val_binary_acc
uracy: 0.8107 - val_dsc: 0.0943 - val_tversky: 0.1516 - val_iou: 0.0511
Epoch 20/60
20/20 [==============================] - 50s 2s/step - loss: 0.9039 - binary_accuracy:
0.8449 - dsc: 0.0983 - tversky: 0.1566 - iou: 0.0538 - val_loss: 0.8906 - val_binary_acc
uracy: 0.8512 - val_dsc: 0.1094 - val_tversky: 0.1740 - val_iou: 0.0600
Epoch 21/60
20/20 [==============================] - 49s 2s/step - loss: 0.8973 - binary_accuracy:
0.8561 - dsc: 0.1027 - tversky: 0.1634 - iou: 0.0561 - val_loss: 0.8917 - val_binary_acc
uracy: 0.8916 - val_dsc: 0.1083 - val_tversky: 0.1733 - val_iou: 0.0602
Epoch 22/60
20/20 [==============================] - 41s 2s/step - loss: 0.8988 - binary_accuracy:
0.8674 - dsc: 0.1012 - tversky: 0.1619 - iou: 0.0554 - val_loss: 0.8548 - val_binary_acc
uracy: 0.8867 - val_dsc: 0.1452 - val_tversky: 0.2247 - val_iou: 0.0811
Epoch 23/60
20/20 [==============================] - 51s 2s/step - loss: 0.8946 - binary_accuracy:
0.8782 - dsc: 0.1054 - tversky: 0.1682 - iou: 0.0580 - val_loss: 0.8436 - val_binary_acc
uracy: 0.8923 - val_dsc: 0.1564 - val_tversky: 0.2384 - val_iou: 0.0876
Epoch 24/60
```

```
20/20 [==============================] - 50s 2s/step - loss: 0.8868 - binary_accuracy:
0.8914 - dsc: 0.1145 - tversky: 0.1810 - iou: 0.0635 - val_loss: 0.8634 - val_binary_acc
uracy: 0.9319 - val_dsc: 0.1366 - val_tversky: 0.2129 - val_iou: 0.0775
Epoch 25/60
20/20 [==============================] - 49s 2s/step - loss: 0.8752 - binary_accuracy:
0.9005 - dsc: 0.1248 - tversky: 0.1949 - iou: 0.0696 - val_loss: 0.8078 - val_binary_acc
uracy: 0.9252 - val_dsc: 0.1922 - val_tversky: 0.2840 - val_iou: 0.1108
Epoch 26/60
20/20 [==============================] - 76s 4s/step - loss: 0.8637 - binary_accuracy:
0.9050 - dsc: 0.1363 - tversky: 0.2114 - iou: 0.0764 - val_loss: 0.8045 - val_binary_acc
uracy: 0.9618 - val_dsc: 0.1955 - val_tversky: 0.2824 - val_iou: 0.1168
Epoch 27/60
20/20 [==============================] - 51s 2s/step - loss: 0.8432 - binary_accuracy:
0.9226 - dsc: 0.1568 - tversky: 0.2392 - iou: 0.0883 - val_loss: 0.6748 - val_binary_acc
uracy: 0.9749 - val_dsc: 0.3252 - val_tversky: 0.4201 - val_iou: 0.2053
Epoch 28/60
20/20 [==============================] - 88s 4s/step - loss: 0.8552 - binary_accuracy:
0.9257 - dsc: 0.1416 - tversky: 0.2198 - iou: 0.0805 - val_loss: 0.6442 - val_binary_acc
uracy: 0.9764 - val_dsc: 0.3558 - val_tversky: 0.4311 - val_iou: 0.2274
Epoch 29/60
20/20 [==============================] - 51s 2s/step - loss: 0.8356 - binary_accuracy:
0.9489 - dsc: 0.1644 - tversky: 0.2511 - iou: 0.0941 - val_loss: 0.6919 - val_binary_acc
uracy: 0.9742 - val_dsc: 0.3081 - val_tversky: 0.4102 - val_iou: 0.1935
Epoch 30/60
20/20 [==============================] - 80s 4s/step - loss: 0.7617 - binary_accuracy:
0.9650 - dsc: 0.2383 - tversky: 0.3385 - iou: 0.1416 - val_loss: 0.5929 - val_binary_acc
uracy: 0.9925 - val_dsc: 0.4071 - val_tversky: 0.4304 - val_iou: 0.2807
Epoch 31/60
20/20 [==============================] - 98s 5s/step - loss: 0.7040 - binary_accuracy:
0.9802 - dsc: 0.2960 - tversky: 0.4005 - iou: 0.1823 - val_loss: 0.7829 - val_binary_acc
uracy: 0.9917 - val_dsc: 0.2171 - val_tversky: 0.2277 - val_iou: 0.1582
Epoch 32/60
20/20 [==============================] - 96s 5s/step - loss: 0.5634 - binary_accuracy:
0.9866 - dsc: 0.4322 - tversky: 0.5186 - iou: 0.2905 - val_loss: 0.7511 - val_binary_acc
uracy: 0.9899 - val_dsc: 0.2489 - val_tversky: 0.2386 - val_iou: 0.1756
Epoch 33/60
20/20 [==============================] - 98s 5s/step - loss: 0.4869 - binary_accuracy:
0.9904 - dsc: 0.5131 - tversky: 0.5663 - iou: 0.3669 - val_loss: 0.9338 - val_binary_acc
uracy: 0.9890 - val_dsc: 0.0662 - val_tversky: 0.0926 - val_iou: 0.0660
Epoch 34/60
20/20 [==============================] - 98s 5s/step - loss: 0.4510 - binary_accuracy:
0.9903 - dsc: 0.5490 - tversky: 0.5895 - iou: 0.4011 - val_loss: 0.9211 - val_binary_acc
uracy: 0.9900 - val_dsc: 0.0789 - val_tversky: 0.1078 - val_iou: 0.0774
Epoch 35/60
20/20 [==============================] - 98s 5s/step - loss: 0.3477 - binary_accuracy:
0.9934 - dsc: 0.6523 - tversky: 0.6616 - iou: 0.5004 - val_loss: 0.9313 - val_binary_acc
uracy: 0.9893 - val_dsc: 0.0687 - val_tversky: 0.0965 - val_iou: 0.0684
Epoch 36/60
20/20 [==============================] - 97s 5s/step - loss: 0.3740 - binary_accuracy:
0.9926 - dsc: 0.6220 - tversky: 0.6489 - iou: 0.4749 - val_loss: 0.9286 - val_binary_acc
uracy: 0.9892 - val_dsc: 0.0714 - val_tversky: 0.0972 - val_iou: 0.0691
Epoch 37/60
20/20 [==============================] - 98s 5s/step - loss: 0.3415 - binary_accuracy:
0.9932 - dsc: 0.6585 - tversky: 0.6614 - iou: 0.5120 - val_loss: 0.6830 - val_binary_acc
uracy: 0.9883 - val_dsc: 0.3170 - val_tversky: 0.2793 - val_iou: 0.2119
Epoch 38/60
20/20 [==============================] - 98s 5s/step - loss: 0.3586 - binary_accuracy:
0.9936 - dsc: 0.6414 - tversky: 0.6610 - iou: 0.4964 - val_loss: 0.5958 - val_binary_acc
uracy: 0.9939 - val_dsc: 0.4042 - val_tversky: 0.3761 - val_iou: 0.2949
Epoch 39/60
```

```
20/20 [==============================] - 55s 3s/step - loss: 0.3023 - binary_accuracy:
0.9941 - dsc: 0.6977 - tversky: 0.7026 - iou: 0.5545 - val_loss: 0.5427 - val_binary_acc
uracy: 0.9937 - val_dsc: 0.4573 - val_tversky: 0.4225 - val_iou: 0.3362
Epoch 40/60
20/20 [==============================] - 50s 2s/step - loss: 0.3300 - binary_accuracy:
0.9934 - dsc: 0.6730 - tversky: 0.6807 - iou: 0.5278 - val_loss: 0.5228 - val_binary_acc
uracy: 0.9922 - val_dsc: 0.4772 - val_tversky: 0.4341 - val_iou: 0.3476
Epoch 41/60
20/20 [==============================] - 53s 3s/step - loss: 0.2982 - binary_accuracy:
0.9948 - dsc: 0.7018 - tversky: 0.7043 - iou: 0.5628 - val_loss: 0.5748 - val_binary_acc
uracy: 0.9909 - val_dsc: 0.4252 - val_tversky: 0.3783 - val_iou: 0.2953
Epoch 42/60
20/20 [==============================] - 78s 4s/step - loss: 0.3279 - binary_accuracy:
0.9938 - dsc: 0.6721 - tversky: 0.6860 - iou: 0.5238 - val_loss: 0.4768 - val_binary_acc
uracy: 0.9940 - val_dsc: 0.5232 - val_tversky: 0.4951 - val_iou: 0.3957
Epoch 43/60
20/20 [==============================] - 51s 2s/step - loss: 0.2932 - binary_accuracy:
0.9937 - dsc: 0.7068 - tversky: 0.6979 - iou: 0.5643 - val_loss: 0.3079 - val_binary_acc
uracy: 0.9952 - val_dsc: 0.6921 - val_tversky: 0.6692 - val_iou: 0.5554
Epoch 44/60
20/20 [==============================] - 51s 2s/step - loss: 0.3361 - binary_accuracy:
0.9935 - dsc: 0.6669 - tversky: 0.6772 - iou: 0.5276 - val_loss: 0.9020 - val_binary_acc
uracy: 0.9884 - val_dsc: 0.0980 - val_tversky: 0.1152 - val_iou: 0.0833
Epoch 45/60
20/20 [==============================] - 52s 2s/step - loss: 0.2983 - binary_accuracy:
0.9952 - dsc: 0.7017 - tversky: 0.7098 - iou: 0.5640 - val_loss: 0.7274 - val_binary_acc
uracy: 0.9909 - val_dsc: 0.2726 - val_tversky: 0.2547 - val_iou: 0.1933
Epoch 46/60
20/20 [==============================] - 52s 3s/step - loss: 0.3226 - binary_accuracy:
0.9931 - dsc: 0.6774 - tversky: 0.6840 - iou: 0.5323 - val_loss: 0.9083 - val_binary_acc
uracy: 0.9887 - val_dsc: 0.0917 - val_tversky: 0.1090 - val_iou: 0.0787
Epoch 47/60
20/20 [==============================] - 84s 4s/step - loss: 0.2702 - binary_accuracy:
0.9946 - dsc: 0.7298 - tversky: 0.7337 - iou: 0.5937 - val_loss: 0.3243 - val_binary_acc
uracy: 0.9948 - val_dsc: 0.6757 - val_tversky: 0.6372 - val_iou: 0.5319
Epoch 48/60
20/20 [==============================] - 98s 5s/step - loss: 0.2682 - binary_accuracy:
0.9943 - dsc: 0.7343 - tversky: 0.7321 - iou: 0.5982 - val_loss: 0.3175 - val_binary_acc
uracy: 0.9929 - val_dsc: 0.6825 - val_tversky: 0.7140 - val_iou: 0.5328
Epoch 49/60
20/20 [==============================] - 99s 5s/step - loss: 0.2768 - binary_accuracy:
0.9943 - dsc: 0.7232 - tversky: 0.7265 - iou: 0.5856 - val_loss: 0.3645 - val_binary_acc
uracy: 0.9943 - val_dsc: 0.6355 - val_tversky: 0.5986 - val_iou: 0.4903
Epoch 50/60
20/20 [==============================] - 72s 3s/step - loss: 0.2637 - binary_accuracy:
0.9953 - dsc: 0.7363 - tversky: 0.7493 - iou: 0.6003 - val_loss: 0.2914 - val_binary_acc
uracy: 0.9951 - val_dsc: 0.7086 - val_tversky: 0.6840 - val_iou: 0.5675
Epoch 51/60
20/20 [==============================] - 91s 4s/step - loss: 0.2573 - binary_accuracy:
0.9946 - dsc: 0.7427 - tversky: 0.7474 - iou: 0.6072 - val_loss: 0.3018 - val_binary_acc
uracy: 0.9929 - val_dsc: 0.6982 - val_tversky: 0.7243 - val_iou: 0.5535
Epoch 52/60
20/20 [==============================] - 64s 3s/step - loss: 0.2591 - binary_accuracy:
0.9949 - dsc: 0.7443 - tversky: 0.7511 - iou: 0.6114 - val_loss: 0.5905 - val_binary_acc
uracy: 0.9933 - val_dsc: 0.4095 - val_tversky: 0.3755 - val_iou: 0.2968
Epoch 53/60
20/20 [==============================] - 89s 4s/step - loss: 0.2623 - binary_accuracy:
0.9952 - dsc: 0.7377 - tversky: 0.7320 - iou: 0.6021 - val_loss: 0.3675 - val_binary_acc
uracy: 0.9932 - val_dsc: 0.6325 - val_tversky: 0.5847 - val_iou: 0.4904
Epoch 54/60
```

```
20/20 [==============================] - 86s 4s/step - loss: 0.2367 - binary_accuracy:
0.9956 - dsc: 0.7633 - tversky: 0.7608 - iou: 0.6338 - val_loss: 0.2639 - val_binary_acc
uracy: 0.9944 - val_dsc: 0.7361 - val_tversky: 0.7600 - val_iou: 0.5960
Epoch 55/60
20/20 [==============================] - 85s 4s/step - loss: 0.2505 - binary_accuracy:
0.9945 - dsc: 0.7495 - tversky: 0.7576 - iou: 0.6206 - val_loss: 0.2535 - val_binary_acc
uracy: 0.9948 - val_dsc: 0.7465 - val_tversky: 0.7321 - val_iou: 0.6099
Epoch 56/60
20/20 [==============================] - 97s 5s/step - loss: 0.2316 - binary_accuracy:
0.9955 - dsc: 0.7534 - tversky: 0.7624 - iou: 0.6335 - val_loss: 0.3096 - val_binary_acc
uracy: 0.9917 - val_dsc: 0.6904 - val_tversky: 0.7432 - val_iou: 0.5405
Epoch 57/60
20/20 [==============================] - 98s 5s/step - loss: 0.2569 - binary_accuracy:
0.9951 - dsc: 0.7431 - tversky: 0.7496 - iou: 0.6100 - val_loss: 0.2949 - val_binary_acc
uracy: 0.9943 - val_dsc: 0.7051 - val_tversky: 0.6967 - val_iou: 0.5653
Epoch 58/60
20/20 [==============================] - 99s 5s/step - loss: 0.2239 - binary_accuracy:
0.9956 - dsc: 0.7761 - tversky: 0.7784 - iou: 0.6497 - val_loss: 0.3739 - val_binary_acc
uracy: 0.9936 - val_dsc: 0.6261 - val_tversky: 0.6613 - val_iou: 0.4939
Epoch 59/60
20/20 [==============================] - 98s 5s/step - loss: 0.2397 - binary_accuracy:
0.9950 - dsc: 0.7603 - tversky: 0.7752 - iou: 0.6273 - val_loss: 0.4516 - val_binary_acc
uracy: 0.9924 - val_dsc: 0.5484 - val_tversky: 0.4951 - val_iou: 0.4055
Epoch 60/60
20/20 [==============================] - 96s 5s/step - loss: 0.2503 - binary_accuracy:
0.9949 - dsc: 0.7478 - tversky: 0.7467 - iou: 0.6145 - val_loss: 0.2739 - val_binary_acc
uracy: 0.9949 - val_dsc: 0.7261 - val_tversky: 0.7437 - val_iou: 0.5871
```

In [13]:

```python
results = model.evaluate(test_gen, steps=len(brain_test)/32)
```

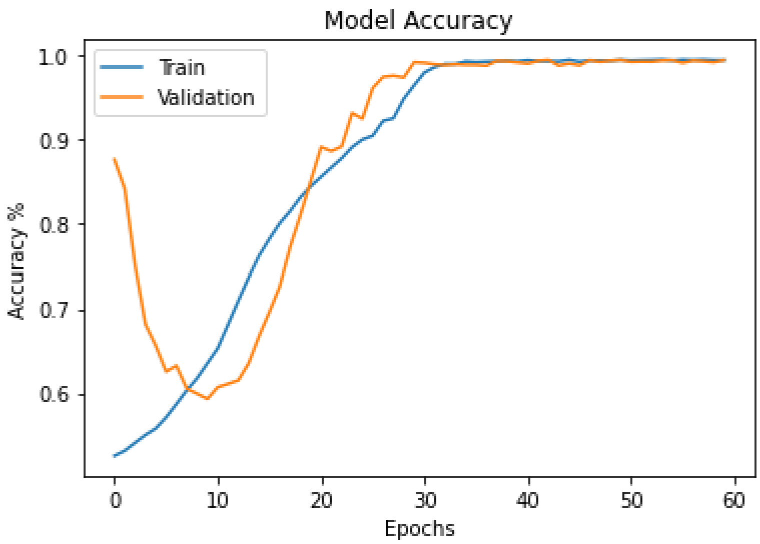
```
Found 590 validated image filenames.
Found 590 validated image filenames.
18/18 [==============================] - 29s 2s/step - loss: 0.2576 - binary_accuracy:
0.9950 - dsc: 0.7344 - tversky: 0.7390 - iou: 0.6029
```

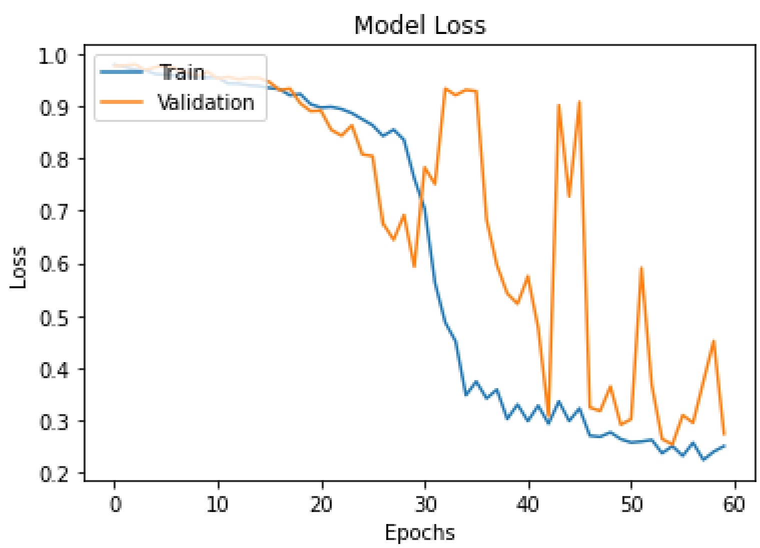
In [17]:

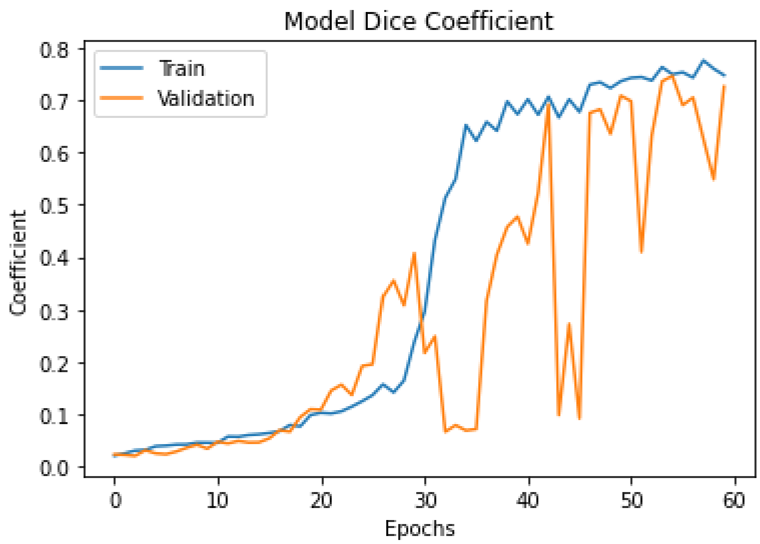
```python
plt.figure(1)
plt.plot(history.history["binary_accuracy"])
plt.plot(history.history["val_binary_accuracy"])
plt.title("Model Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy %")
plt.legend(["Train","Validation"],loc="upper left")

plt.figure(2)
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("Model Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(["Train","Validation"],loc="upper left")

plt.figure(3)
plt.plot(history.history["dsc"])
plt.plot(history.history["val_dsc"])
plt.title("Model Dice Coefficient")
plt.xlabel("Epochs")
plt.ylabel("Coefficient")
plt.legend(["Train","Validation"],loc="upper left")
```

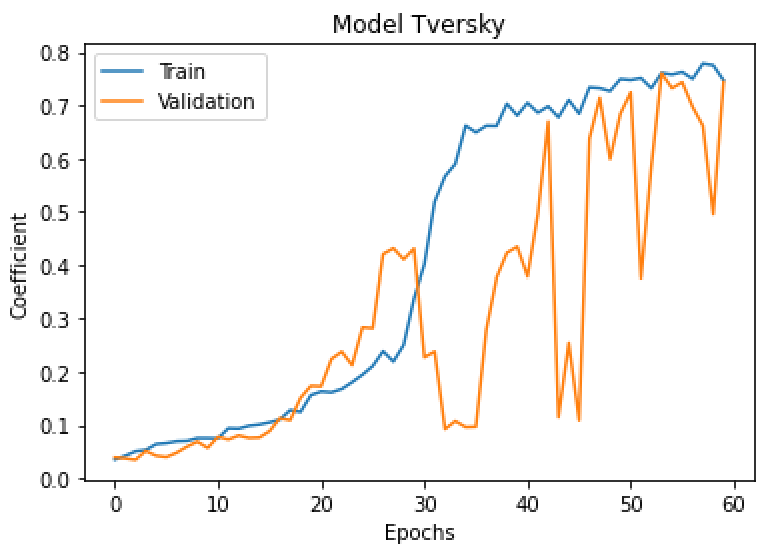
```python
plt.figure(4)
plt.plot(history.history["tversky"])
plt.plot(history.history["val_tversky"])
plt.title("Model Tversky")
plt.xlabel("Epochs")
plt.ylabel("Coefficient")
plt.legend(["Train","Validation"],loc="upper left")

plt.figure(5)
plt.plot(history.history["iou"])
plt.plot(history.history["val_iou"])
plt.title("Model IOU")
plt.xlabel("Epochs")
plt.ylabel("Coefficient")
plt.legend(["Train","Validation"],loc="upper left")
```

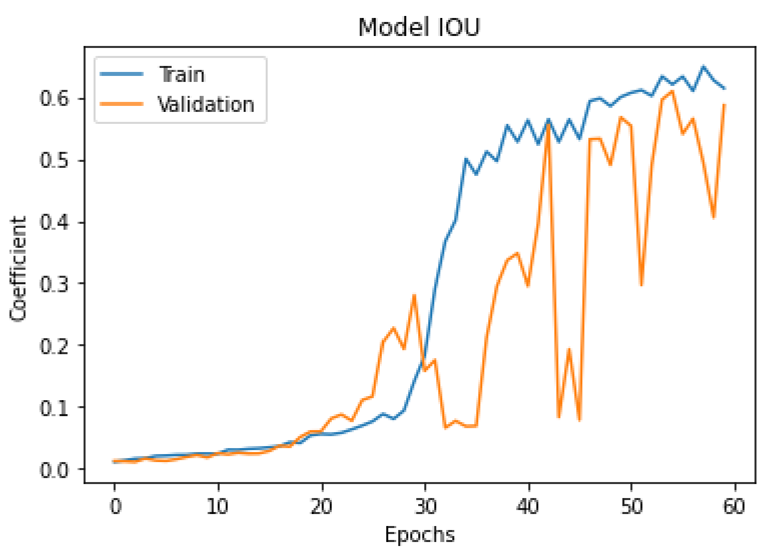
Out[17]: <matplotlib.legend.Legend at 0x1f347a8bc10>





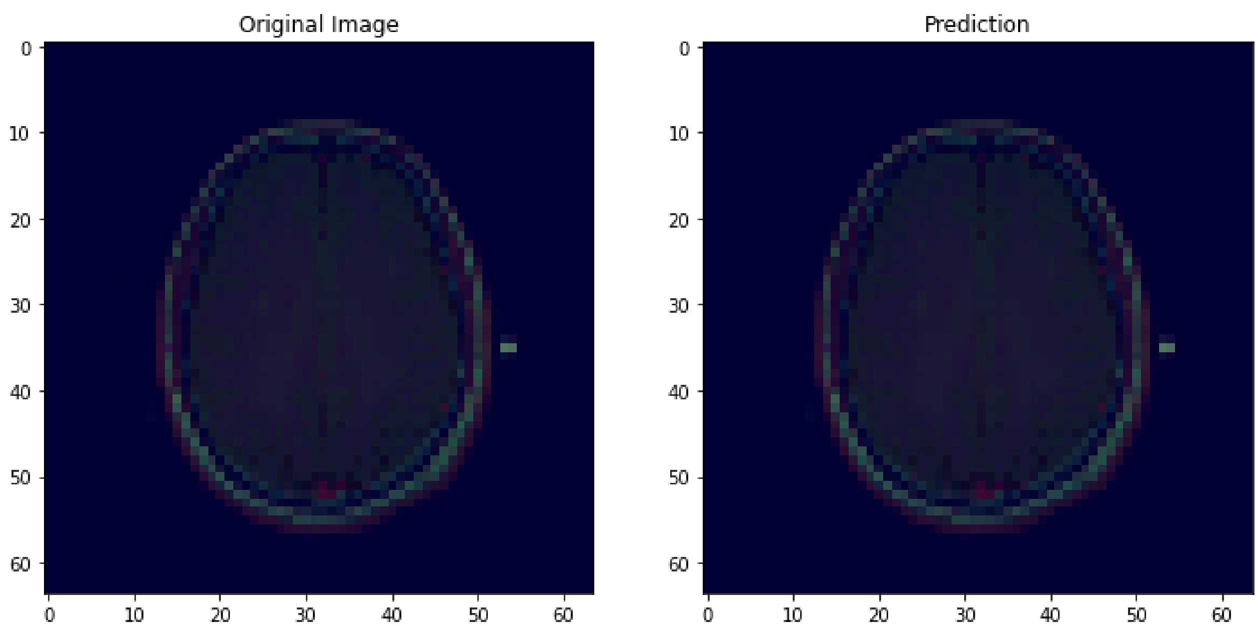



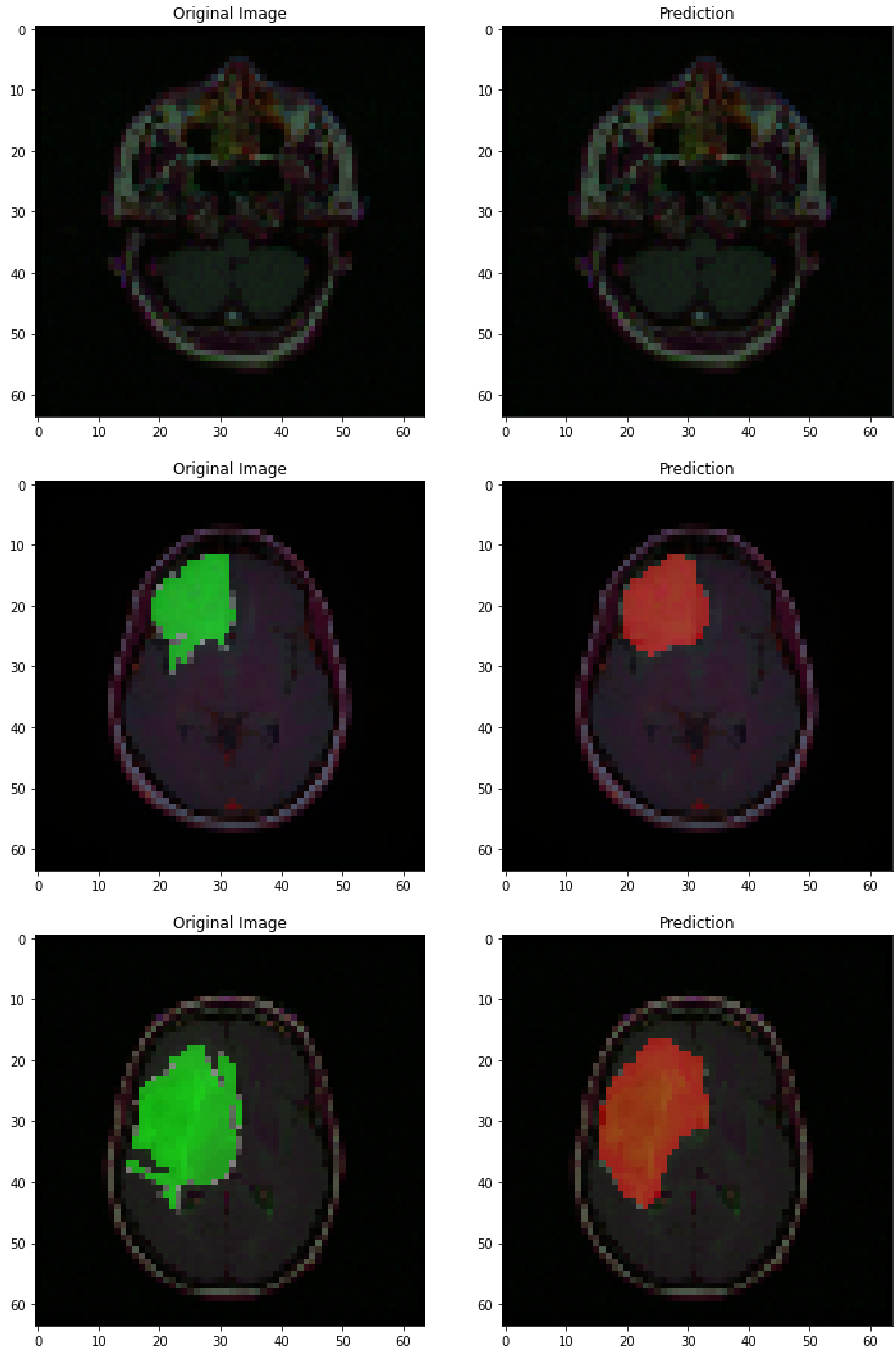



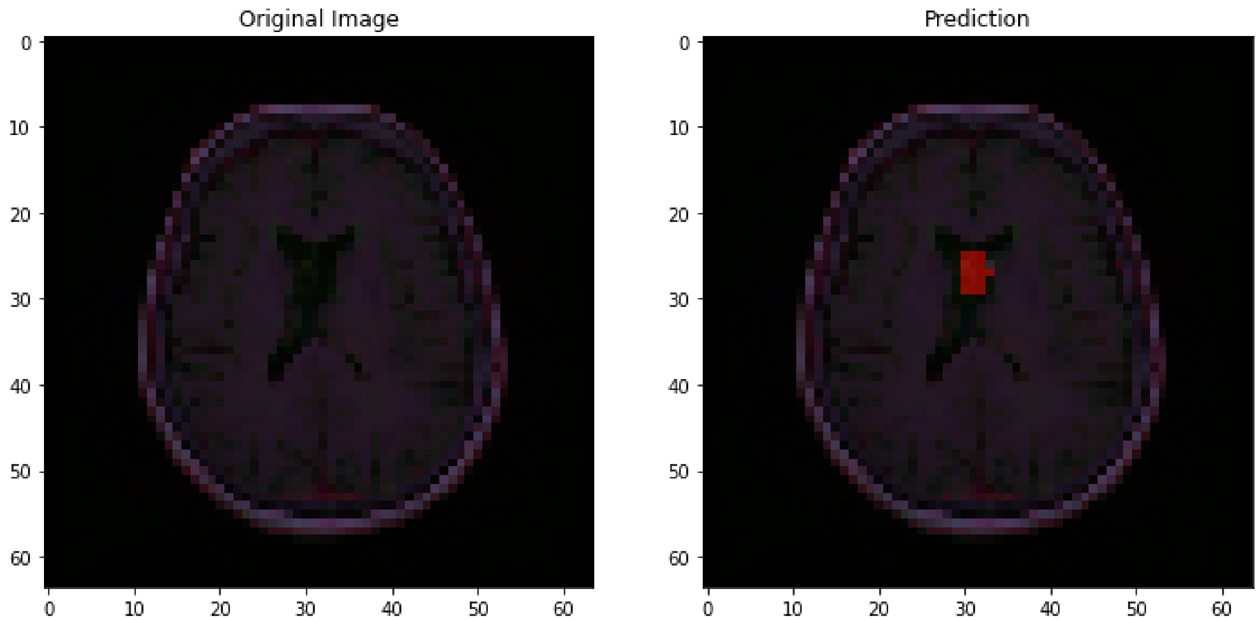

In [25]:

```
for i in range(5):
    index=np.random.randint(1,len(brain_test.index))
    img = cv2.imread(brain_test['Image File'].iloc[index])
    img = cv2.resize(img ,(64, 64))
    img = img / 255
    img = img[np.newaxis, :, :, :]
```

```
pred=model.predict(img)

plt.figure(figsize=(12,12))
plt.subplot(1,2,1)
plt.imshow(np.squeeze(img))
plt.title('Original Image')
mask = cv2.imread(brain_test['Mask File'].iloc[index])
mask = cv2.resize(mask,(64,64))
indices = np.where(mask==255)
mask[indices[0], indices[1], :] = [0, 255, 0]
plt.imshow(np.squeeze(mask),cmap='jet',alpha=0.5)
plt.subplot(1,2,2)
plt.imshow(np.squeeze(img))
pred = np.repeat(pred, 3, axis=-1)
pred = np.squeeze(pred)
indices = np.where(pred>.5)
pred[indices[0], indices[1], :] = [1, 0, 0]
plt.imshow(pred,cmap='jet',alpha=0.5)

plt.title('Prediction')
plt.show()
```




Original Image
Prediction
Original Image
Prediction
Original Image
Prediction



Original Image
Prediction
0
10
20
30
40
50
60