In [42]:

```
# import libraries

import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import classification_report,confusion_matrix

import keras
from keras import backend as K
from keras.models import *
from keras.layers import *
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.applications import ResNet50
from keras.applications.vgg16 import VGG16

import tensorflow as tf
import os
from skimage import io

#import os
#os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

Step 1: Brief description of the problem and data

1.1 Data

In this dataset, we are provided with a large number of small pathology images to classify. Files are named with an image id. The train_labels.csv file provides the ground truth for the images in the train folder. We are predicting the labels for the images in the test folder. A positive label indicates that the center 32x32px region of a patch contains at least one pixel of tumor tissue. Tumor tissue in the outer region of the patch does not influence the label.

The dataset was downloaded from https://www.kaggle.com/competitions/histopathologic-cancer-detection/data

1.2 Project Topic

The goal of this project is to identify metastatic cancer in small image patches taken from larger digital pathology scans. To achieve this goal, I will:

- Inspect, Visualize and Clean the data.
- Build two CNN models.
- Run hyperparameter tuning, try different architectures for comparison.
- Analysis and Result.

- Conclusion.

Step 2: EDA - Inspect, Visualize and Clean the Data

2.1 Inspect the data

In [2]:

```
#train_path = '../input/histopathologic-cancer-detection/train/'
#test_path = '../input/histopathologic-cancer-detection/test/'
```

In [3]:

```
train_path = '../Week3/train/'
test_path = '../Week3/test/'
```

In [4]:

```
# take a look at some first train_label rows
#train_df = pd.read_csv('../input/histopathologic-cancer-detection/train_labels.
train_df = pd.read_csv('../Week3/train_labels.csv')
train_df.head()
```

Out[4]:

	id	label
0	f38a6374c348f90b587e046aac6079959adf3835	0
1	c18f2d887b7ae4f6742ee445113fa1aef383ed77	1
2	755db6279dae599ebb4d39a9123cce439965282d	0
3	bc3f0c64fb968ff4a8bd33af6971ecae77c75e08	0
4	068aba587a4950175d04c680d38943fd488d6a9d	0

In [5]:

```
# get a quick description of the data
train_df.describe()
```

Out[5]:

	label
count	220025.000000
mean	0.405031
std	0.490899
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

In [6]:

```
# check null values in data
train_df.isnull().sum()
```

Out[6]:

```
id       0
label    0
```

```
dtype: int64
```

In [7]:

```python
# check for duplicate train_label
train_df.duplicated(keep=False).sum()
```

Out[7]:

```
0
```

In [8]:

```python
# the structure of data also tells us the number of rows, columns and type of da
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220025 entries, 0 to 220024
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   id      220025 non-null  object
 1   label   220025 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 3.4+ MB
```

In [9]:

```python
# get the number of entries in test set
#test_df = pd.DataFrame({'id':os.listdir(test_path)})
test_df = pd.DataFrame({'id':os.listdir('../Week3/test')})
print(len(test_df))
```

```
57458
```

In [10]:

```python
# take a look at some first rows of test set
test_df.head()
```

Out[10]:

	id
0	fd0a060ef9c30c9a83f6b4bfb568db74b099154d.tif
1	1f9ee06f06d329eb7902a2e03ab3835dd0484581.tif
2	19709bec800f372d0b1d085da6933dd3ef108846.tif
3	7a34fc34523063f13f0617f7518a0330f6187bd3.tif
4	93be720ca2b95fe2126cf2e1ed752bd759e9b0ed.tif

In [11]:

```python
# check null values in test set
test_df.isnull().sum()
```

Out[11]:

```
id    0
dtype: int64
```

In [12]:

```python
# check for duplicate in test set
test_df.duplicated(keep=False).sum()
```

Out[12]:

```
0
```

From the output above, we can summarize that:

- There are 220,025 entries and 2 columns in train_label data.
- There is no missing values.
- There is no duplicated entries.
- The id column is object and label column is integer with two values 0 (no_tumor_tissue) and 1 (has_tumor_tissue).
- There are 57,458 entries in test data.

2.2 Visualize the data

In [13]:

```
# calculate the count of each label
train_df['label'].value_counts()
```

Out[13]:

```
0    130908
1     89117
Name: label, dtype: int64
```

In [14]:

```
# calculate the proportion of each label
train_df['label'].value_counts()/len(train_df)*100
```

Out[14]:

```
0    59.496875
1    40.503125
Name: label, dtype: float64
```

In [15]:

```
# plot the count of each label
fig, ax = plt.subplots(figsize=(6,6))
sns.countplot(data=train_df, y='label', ax=ax).set(title='\nFigure 1. The Count

# plot the proportion of each label
labels = train_df['label'].unique().tolist()
counts = train_df['label'].value_counts()
sizes = [counts[v] for v in labels]
fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%0.2f%%')
ax1.axis('equal')
plt.title("\nFigure 2. The Proportion of Each Label\n")
plt.tight_layout()
plt.show()
```

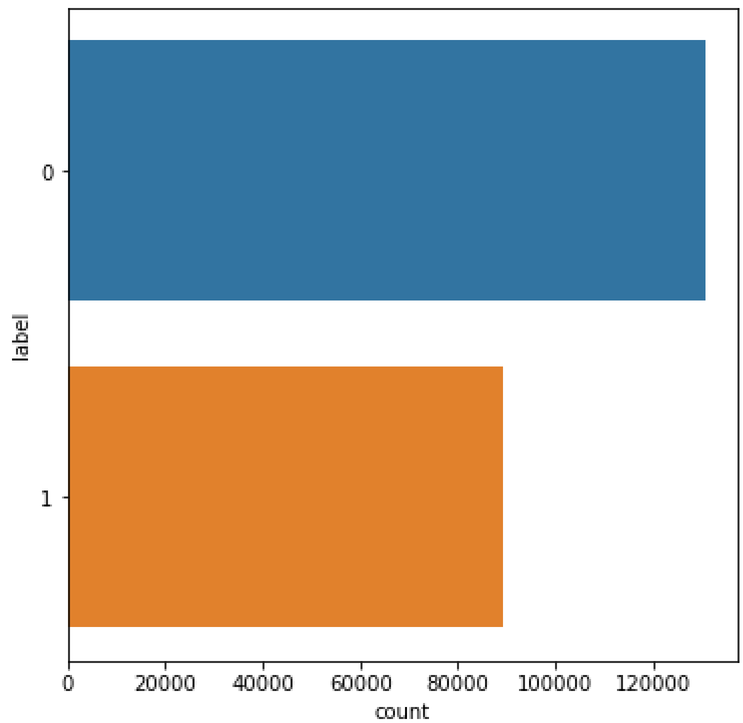


Figure 1. The Count of Each Label

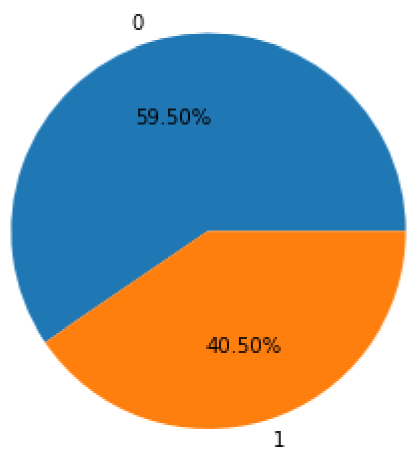


Figure 2. The Proportion of Each Label

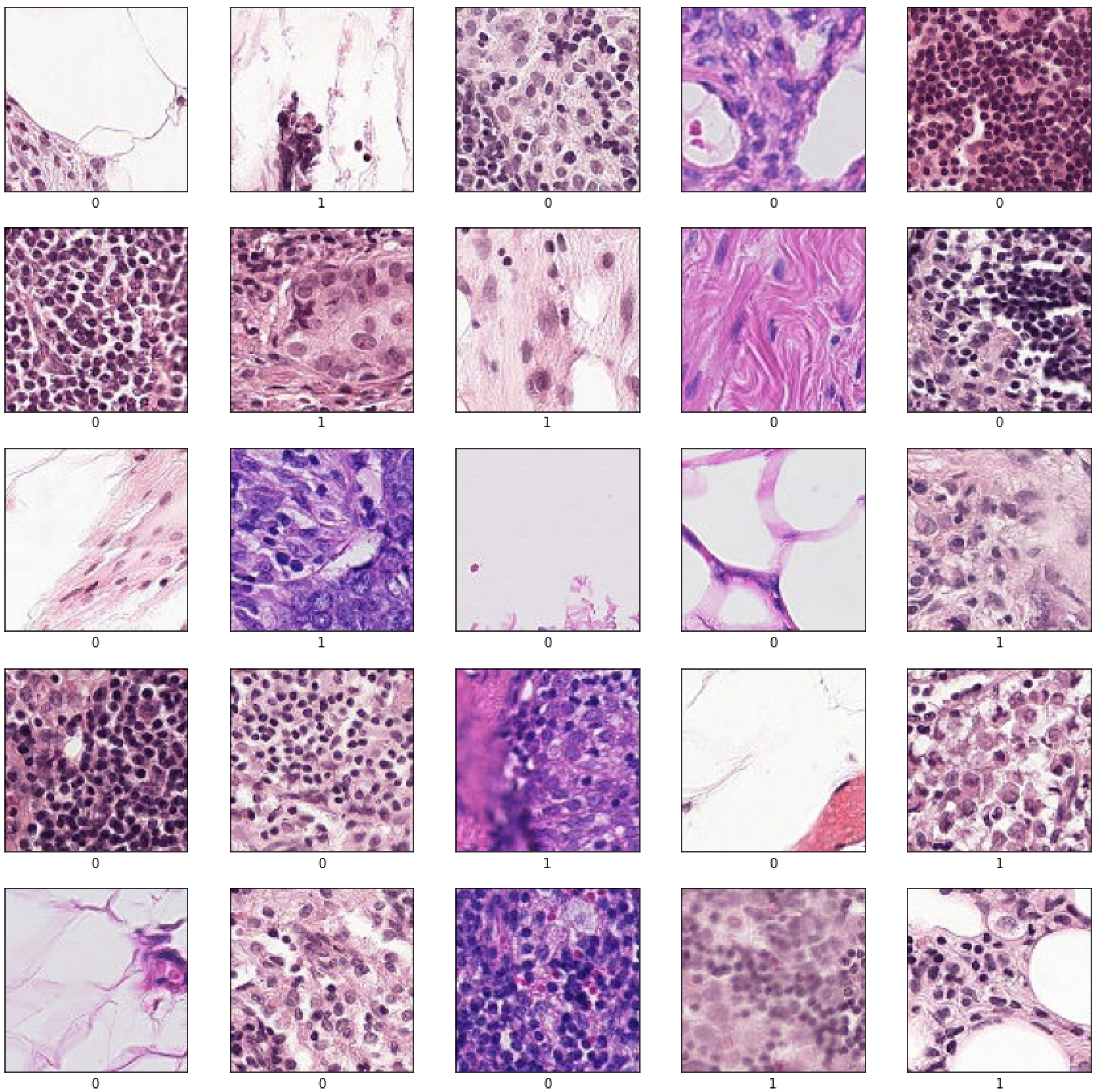
Figure 1 shows the count of each label and figure 2 shows the proportions of each label. Looking at these two figures, we can see that in overall, the number of example for no_tumor_tissue label (0) was larger than the number of example for has_tumor_tissue label (1). And with the proportion of example for no_tumor_tissue label (0) was nearly 60% comparing to 40% of example for has_tumor_tissue (1), I think it will be better if we balance data by reducing the number of samples in label 0 since if one category was severely underrepresentated or, in contrast, overrepresentative in the train data, then it may cause our model to be biased and/or perform poorly on some or all of the test data.

In [16]:

```
# Train image visualisations
def append_tif(string):
    return string + ".tif"
train_df["id"] = train_df["id"].apply(append_tif)
```

```python
train_df["label"] = train_df["label"].astype(str)

fig, axes = plt.subplots(5, 5, figsize=(15, 15))
for i, ax in enumerate(axes.flat):
    file = str(train_path + train_df.id[i])
    image = io.imread(file)
    ax.imshow(image)
    ax.set(xticks=[], yticks=[], xlabel = train_df.label[i])
fig.suptitle('\nFigure 3. Train image Visualizations')
plt.show()
```

Figure 3. Train image Visualizations

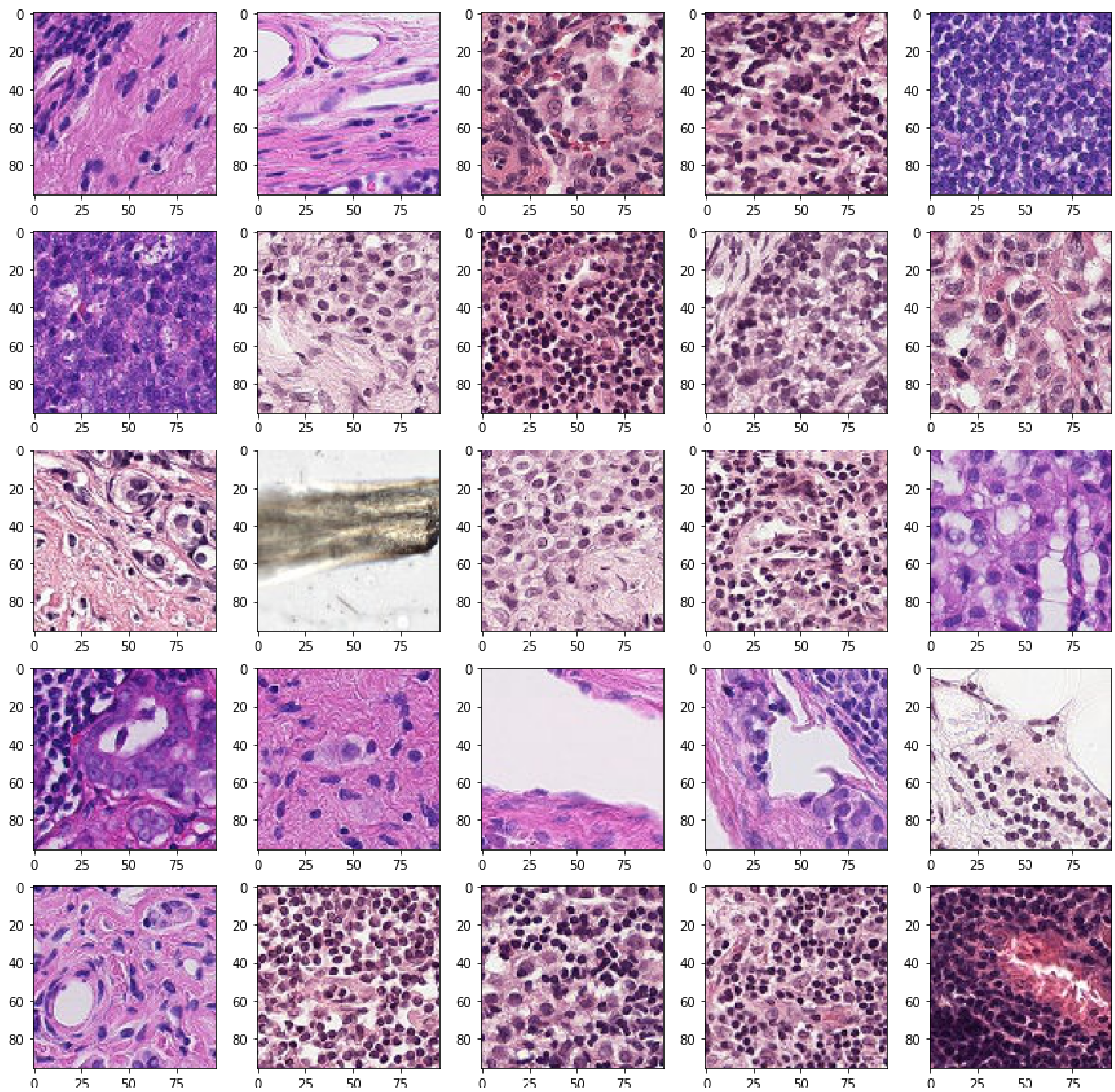


In [17]:

```python
# Test image visualisations
fig, axes = plt.subplots(5, 5, figsize=(15, 15))
for i, ax in enumerate(axes.flat):
```

```python
    file = str(test_path + test_df.id[i])
    image = io.imread(file)
    ax.imshow(image)
fig.suptitle('\nFigure 4. Test image Visualizations')
plt.show()
```

Figure 4. Test image Visualizations



2.3 Clean the data / Data Preprocessing

In [19]:

```python
# check the data format
print(K.image_data_format())
```

channels_last

In [20]:

```python
# set random state
RANDOM_STATE = 42
```

In [21]:

```
# set batch size
BATCH_SIZE = 10
```

In [22]:

```
# set input shape
img_width, img_height = 64, 64
input_shape = (img_width, img_height, 3)
```

Let's balance the data and split train dataset to:

- train set: is the set used for training model
- validation set: is the set used during the model training to adjust the hyperparameters. (20%)

In [23]:

```
# balance the data
SAMPLE = 80000
train1 = train_df[train_df["label"] == "0"].sample(SAMPLE, random_state=RANDOM_S
train2 = train_df[train_df["label"] == "1"].sample(SAMPLE, random_state=RANDOM_S
train_dt = pd.concat([train1, train2], axis=0).reset_index(drop=True)
train_dt["label"].value_counts()
```

Out[23]:

```
0    80000
1    80000
Name: label, dtype: int64
```

In [26]:

```
# split train dataset to train and validation set
train_data, valid_data = train_test_split(train_dt,
                                          random_state=RANDOM_STATE,
                                          test_size=0.2,
                                          shuffle=True, stratify=train_dt["label"])

# check value count in train and validation set
print(train_data["label"].value_counts())
print(valid_data["label"].value_counts())
```

```
0    64000
1    64000
Name: label, dtype: int64
0    16000
1    16000
Name: label, dtype: int64
```

Before we can proceed with building the model:

The first step to working with neural networks is to normalize the dataset, otherwise, it could take a lot longer for the network to converge on a solution.

The usual way of normalizing a dataset is to scale the features, and this is done by subtracting the mean from each feature and dividing by the standard deviation. This will put the features on the same scale somewhere between 0 — 1.

As we are working with 32 x 32 NumPy arrays representing each image and each pixel in the array has an intensity somewhere between 1 — 255, a simpler way of getting all of these images on a scale between 0–1 is to divide each array by 255.

In [27]:

```
datagen = ImageDataGenerator(featurewise_center=False,  # set input mean to 0 ov
                             zoom_range = 0.2, # Randomly zoom image
                             rotation_range = 30,  # randomly rotate images in t
                             width_shift_range=0.1,  # randomly shift images hor
                             height_shift_range=0.1,  # randomly shift images ve
                             horizontal_flip = True,  # randomly flip images
                             rescale=1./255)    # multiply the data by the value
```

In [28]:

```
train_generator = datagen.flow_from_dataframe(
                            dataframe=train_data,
                            directory=train_path,
                            x_col="id",
                            y_col="label",
                            batch_size=BATCH_SIZE,
                            seed=RANDOM_STATE,
                            class_mode="binary",
                            target_size=(64,64))
```

```
Found 128000 validated image filenames belonging to 2 classes.
```

In [29]:

```
validation_generator = datagen.flow_from_dataframe(
                            dataframe=valid_data,
                            directory=train_path,
                            x_col="id",
                            y_col="label",
                            batch_size=BATCH_SIZE,
                            seed=RANDOM_STATE,
                            class_mode="binary",
                            target_size=(64,64))
```

```
Found 32000 validated image filenames belonging to 2 classes.
```

Step 3: Describe Model Architecture

Model 1:

Model is comprised of:

A simple CNN model with 3 Convolutional layers followed by max-pooling layers. A dropout layer is added at the final convolutional layer to avoid overfitting. BatchNormalization normalize the activation of the previous layer at each batch. Sigmoid is used as the activation function for the final layer of the binary classifier. Use binary-entropy loss function for our binary-class classification problem. For simplicity, use accuracy as our evaluation metrics to evaluate the model during training and testing.

- optimization: Adam
- learning rate: 0.0001

- hidden layer activations: relu
- final layer dropout: 0.4
- final layer activation: sigmoid because of the binary classification

In [30]:

```
model = Sequential()
# first convolutional layer
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# second convolutional layer
model.add(Conv2D(64, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# third convolutional layer
model.add(Conv2D(128, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.4))

# Out layer
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896       
                                                                 
 batch_normalization (BatchN  (None, 62, 62, 32)       128       
 ormalization)                                                   
                                                                 
 activation (Activation)     (None, 62, 62, 32)        0         
                                                                 
 max_pooling2d (MaxPooling2D  (None, 31, 31, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 29, 29, 64)        18496     
                                                                 
 batch_normalization_1 (Batc  (None, 29, 29, 64)       256       
 hNormalization)                                                 
                                                                 
 activation_1 (Activation)   (None, 29, 29, 64)        0         
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 14, 14, 64)       0         
 2D)                                                             
```

```
 conv2d_2 (Conv2D)           (None, 12, 12, 128)       73856

 batch_normalization_2 (Batc  (None, 12, 12, 128)      512
 hNormalization)

 activation_2 (Activation)   (None, 12, 12, 128)       0

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 256)               1179904

 batch_normalization_3 (Batc  (None, 256)              1024
 hNormalization)

 activation_3 (Activation)   (None, 256)               0

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 1)                 257

 activation_4 (Activation)   (None, 1)                 0

=================================================================
Total params: 1,275,329
Trainable params: 1,274,369
Non-trainable params: 960
_________________________________________________________________
```

Let's compile the model now using Adam as our optimizer and binary crossentropy as the loss function. We are using a lower learning rate of 0.0001 for a smoother curve.

In [31]:

```
# compile the model
opt = Adam(learning_rate=0.0001)
model.compile(loss='binary_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

In [32]:

```
# let's train our model for 20 epochs
STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
STEP_SIZE_VALID=validation_generator.n//validation_generator.batch_size
history = model.fit(train_generator,
                    epochs = 20 ,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    validation_data = validation_generator,
                    validation_steps=STEP_SIZE_VALID)
```

```
Epoch 1/20
12800/12800 [==============================] - 697s 54ms/step - loss: 0.4683 - a
ccuracy: 0.7861 - val_loss: 0.3623 - val_accuracy: 0.8436
Epoch 2/20
12800/12800 [==============================] - 701s 55ms/step - loss: 0.4092 - a
ccuracy: 0.8194 - val_loss: 0.6046 - val_accuracy: 0.7189
Epoch 3/20
```

```
12800/12800 [==============================] - 713s 56ms/step - loss: 0.3906 - a
ccuracy: 0.8296 - val_loss: 0.5020 - val_accuracy: 0.7549
Epoch 4/20
12800/12800 [==============================] - 729s 57ms/step - loss: 0.3741 - a
ccuracy: 0.8383 - val_loss: 0.3132 - val_accuracy: 0.8693
Epoch 5/20
12800/12800 [==============================] - 725s 57ms/step - loss: 0.3611 - a
ccuracy: 0.8466 - val_loss: 0.4342 - val_accuracy: 0.7918
Epoch 6/20
12800/12800 [==============================] - 729s 57ms/step - loss: 0.3494 - a
ccuracy: 0.8520 - val_loss: 0.3308 - val_accuracy: 0.8572
Epoch 7/20
12800/12800 [==============================] - 701s 55ms/step - loss: 0.3422 - a
ccuracy: 0.8552 - val_loss: 0.4511 - val_accuracy: 0.7863
Epoch 8/20
12800/12800 [==============================] - 4591s 359ms/step - loss: 0.3346 -
accuracy: 0.8604 - val_loss: 0.2900 - val_accuracy: 0.8781
Epoch 9/20
12800/12800 [==============================] - 2214s 173ms/step - loss: 0.3297 -
accuracy: 0.8636 - val_loss: 0.3425 - val_accuracy: 0.8518
Epoch 10/20
12800/12800 [==============================] - 3386s 265ms/step - loss: 0.3237 -
accuracy: 0.8666 - val_loss: 0.3431 - val_accuracy: 0.8586
Epoch 11/20
12800/12800 [==============================] - 708s 55ms/step - loss: 0.3194 - a
ccuracy: 0.8687 - val_loss: 0.5548 - val_accuracy: 0.7660
Epoch 12/20
12800/12800 [==============================] - 709s 55ms/step - loss: 0.3146 - a
ccuracy: 0.8711 - val_loss: 0.2735 - val_accuracy: 0.8902
Epoch 13/20
12800/12800 [==============================] - 708s 55ms/step - loss: 0.3124 - a
ccuracy: 0.8727 - val_loss: 0.3172 - val_accuracy: 0.8640
Epoch 14/20
12800/12800 [==============================] - 703s 55ms/step - loss: 0.3083 - a
ccuracy: 0.8742 - val_loss: 0.2580 - val_accuracy: 0.8932
Epoch 15/20
12800/12800 [==============================] - 713s 56ms/step - loss: 0.3032 - a
ccuracy: 0.8768 - val_loss: 0.2570 - val_accuracy: 0.8950
Epoch 16/20
12800/12800 [==============================] - 709s 55ms/step - loss: 0.2990 - a
ccuracy: 0.8789 - val_loss: 0.3435 - val_accuracy: 0.8635
Epoch 17/20
12800/12800 [==============================] - 701s 55ms/step - loss: 0.2997 - a
ccuracy: 0.8790 - val_loss: 0.2426 - val_accuracy: 0.9014
Epoch 18/20
12800/12800 [==============================] - 698s 55ms/step - loss: 0.2958 - a
ccuracy: 0.8807 - val_loss: 0.3165 - val_accuracy: 0.8632
Epoch 19/20
12800/12800 [==============================] - 699s 55ms/step - loss: 0.2929 - a
ccuracy: 0.8824 - val_loss: 0.2443 - val_accuracy: 0.9023
Epoch 20/20
12800/12800 [==============================] - 703s 55ms/step - loss: 0.2905 - a
ccuracy: 0.8821 - val_loss: 0.2681 - val_accuracy: 0.8869
```

In [36]:

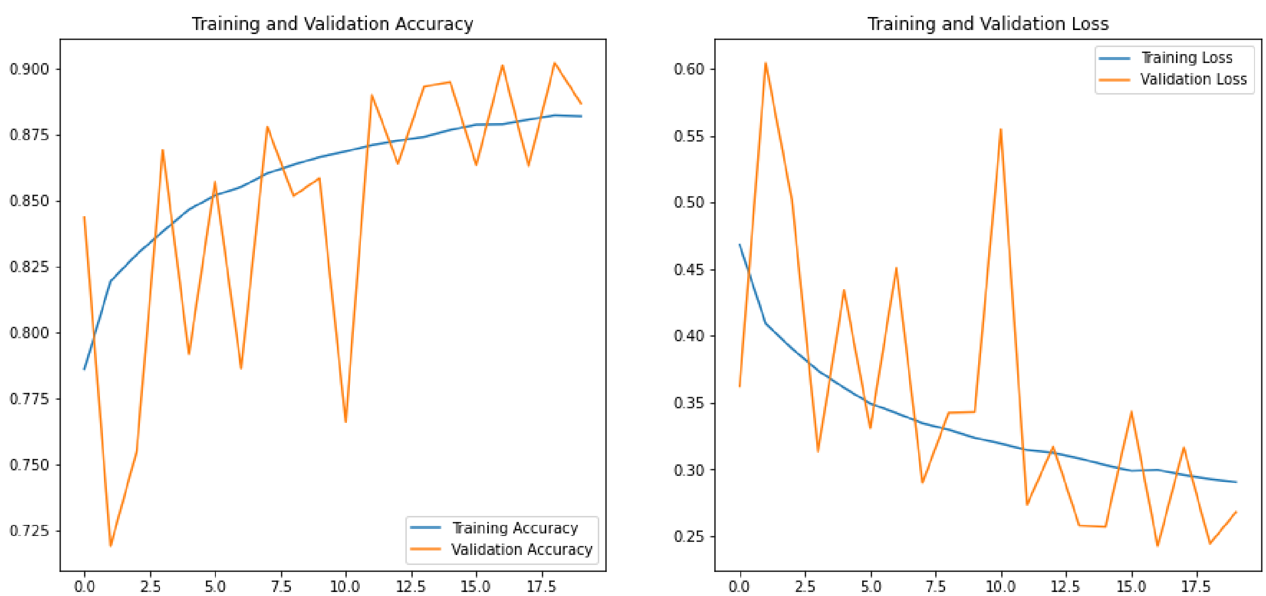
```
model.save("../Week3/my_model1")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit
_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 3 o
f 3). These functions will not be directly callable after loading.
```

```
INFO:tensorflow:Assets written to: ../Week3/my_model1/assets
INFO:tensorflow:Assets written to: ../Week3/my_model1/assets
```

In [37]:

```python
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(20)

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



Model 2

Next, let's use Earlystopping to avoid overfitting by terminating the process early. Since the goal of a training is to minimize the loss. With this, we can set up the metric as:

```
+ monitor : val_loss, value being monitored.
+ mode: min, training will stop when the quantity monitored has
stopped decreasing.
+ patience: 3, number of epochs with no improvement after which
training will be stopped.
```

Moreover, let's use Reduce learning rate when a metric has stopped improving. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

```
+ factor: factor by which the learning rate will be reduced
(new_learning_rate = learning_rate * factor).
+ min_lr: lower bound on the learning rate.
```

A model.fit() training loop will check at end of every epoch whether the loss is no longer decreasing, considering the min_delta and patience if applicable. Once it's found no longer decreasing, model.stop_training is marked True and the training terminates.

In [38]:

```
# define an Earlystopping
checkpoint_filepath = '../Week3/checkpoint'
mp= tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_filepath, save_weight
                               verbose=1, save_best_only=True)
es= tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', patience=3,
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,
                                   verbose=1, mode='min', min_lr=0.00001)
callback=[es, mp, reduce_lr]
```

In [39]:

```
new_model = Sequential()
# first convolutional layer
new_model.add(Conv2D(32, (3, 3), input_shape=input_shape))
new_model.add(BatchNormalization())
new_model.add(Activation('relu'))
new_model.add(MaxPooling2D(pool_size=(2, 2)))

# second convolutional layer
new_model.add(Conv2D(64, (3, 3)))
new_model.add(BatchNormalization())
new_model.add(Activation('relu'))
new_model.add(MaxPooling2D(pool_size=(2, 2)))

# third convolutional layer
new_model.add(Conv2D(128, (3, 3)))
new_model.add(BatchNormalization())
new_model.add(Activation('relu'))
new_model.add(MaxPooling2D(pool_size=(2, 2)))

new_model.add(Flatten())
new_model.add(Dense(256))
new_model.add(BatchNormalization())
new_model.add(Activation('relu'))
new_model.add(Dropout(0.4))

# Out layer
new_model.add(Dense(1))
new_model.add(Activation('sigmoid'))

new_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_3 (Conv2D)           (None, 62, 62, 32)        896       
                                                                 
 batch_normalization_4 (Batc  (None, 62, 62, 32)       128       
 hNormalization)                                                 
                                                                 
 activation_5 (Activation)   (None, 62, 62, 32)        0         
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 31, 31, 32)       0         
 2D)                                                             
                                                                 
 conv2d_4 (Conv2D)           (None, 29, 29, 64)        18496     
                                                                 
 batch_normalization_5 (Batc  (None, 29, 29, 64)       256       
 hNormalization)                                                 
                                                                 
 activation_6 (Activation)   (None, 29, 29, 64)        0         
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 14, 14, 64)       0         
 2D)                                                             
                                                                 
 conv2d_5 (Conv2D)           (None, 12, 12, 128)       73856     
                                                                 
 batch_normalization_6 (Batc  (None, 12, 12, 128)      512       
 hNormalization)                                                 
                                                                 
 activation_7 (Activation)   (None, 12, 12, 128)       0         
                                                                 
 max_pooling2d_5 (MaxPooling  (None, 6, 6, 128)        0         
 2D)                                                             
                                                                 
 flatten_1 (Flatten)         (None, 4608)              0         
                                                                 
 dense_2 (Dense)             (None, 256)               1179904   
                                                                 
 batch_normalization_7 (Batc  (None, 256)              1024      
 hNormalization)                                                 
                                                                 
 activation_8 (Activation)   (None, 256)               0         
                                                                 
 dropout_1 (Dropout)         (None, 256)               0         
                                                                 
 dense_3 (Dense)             (None, 1)                 257       
                                                                 
 activation_9 (Activation)   (None, 1)                 0         
                                                                 
=================================================================
Total params: 1,275,329
Trainable params: 1,274,369
Non-trainable params: 960
_________________________________________________________________
```

In [40]:
```python
# compile new model
new_model.compile(loss='binary_crossentropy',
              optimizer=Adam(learning_rate=0.0001),
              metrics=['accuracy'])
```

In [41]:
```python
# let's train our model for 20 epochs
new_history = new_model.fit(train_generator,
                    epochs = 20 ,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    validation_data = validation_generator,
                    validation_steps=STEP_SIZE_VALID,
                    callbacks=callback)
```

```
Epoch 1/20
12800/12800 [==============================] - ETA: 0s - loss: 0.4709 - accurac
y: 0.7863
Epoch 1: val_loss improved from inf to 0.44097, saving model to ../Week3/checkpo
int
12800/12800 [==============================] - 708s 55ms/step - loss: 0.4709 - a
ccuracy: 0.7863 - val_loss: 0.4410 - val_accuracy: 0.8045 - lr: 1.0000e-04
Epoch 2/20
12800/12800 [==============================] - ETA: 0s - loss: 0.4076 - accurac
y: 0.8214
Epoch 2: val_loss improved from 0.44097 to 0.34267, saving model to ../Week3/che
ckpoint
12800/12800 [==============================] - 708s 55ms/step - loss: 0.4076 - a
ccuracy: 0.8214 - val_loss: 0.3427 - val_accuracy: 0.8535 - lr: 1.0000e-04
Epoch 3/20
12800/12800 [==============================] - ETA: 0s - loss: 0.3863 - accurac
y: 0.8316
Epoch 3: val_loss improved from 0.34267 to 0.33191, saving model to ../Week3/che
ckpoint
12800/12800 [==============================] - 708s 55ms/step - loss: 0.3863 - a
ccuracy: 0.8316 - val_loss: 0.3319 - val_accuracy: 0.8586 - lr: 1.0000e-04
Epoch 4/20
12799/12800 [============================>.] - ETA: 0s - loss: 0.3667 - accurac
y: 0.8425
Epoch 4: val_loss improved from 0.33191 to 0.32389, saving model to ../Week3/che
ckpoint
12800/12800 [==============================] - 709s 55ms/step - loss: 0.3666 - a
ccuracy: 0.8425 - val_loss: 0.3239 - val_accuracy: 0.8609 - lr: 1.0000e-04
Epoch 5/20
12799/12800 [============================>.] - ETA: 0s - loss: 0.3577 - accurac
y: 0.8490
Epoch 5: val_loss did not improve from 0.32389
12800/12800 [==============================] - 726s 57ms/step - loss: 0.3578 - a
ccuracy: 0.8490 - val_loss: 0.3934 - val_accuracy: 0.8189 - lr: 1.0000e-04
Epoch 6/20
12799/12800 [============================>.] - ETA: 0s - loss: 0.3457 - accurac
y: 0.8536
Epoch 6: val_loss did not improve from 0.32389
12800/12800 [==============================] - 714s 56ms/step - loss: 0.3458 - a
ccuracy: 0.8536 - val_loss: 0.3250 - val_accuracy: 0.8552 - lr: 1.0000e-04
Epoch 7/20
12800/12800 [==============================] - ETA: 0s - loss: 0.3351 - accurac
y: 0.8591
Epoch 7: val_loss did not improve from 0.32389
```

```
Epoch 7: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
12800/12800 [==============================] - 719s 56ms/step - loss: 0.3351 - a
ccuracy: 0.8591 - val_loss: 0.4340 - val_accuracy: 0.8200 - lr: 1.0000e-04
Epoch 7: early stopping
```

In [43]:

```
new_model.save("../Week3/my_model2")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit
_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 3 o
f 3). These functions will not be directly callable after loading.
INFO:tensorflow:Assets written to: ../Week3/my_model2/assets
```

```
INFO:tensorflow:Assets written to: ../Week3/my_model2/assets
```

In [45]:

```
new_acc = new_history.history['accuracy']
new_val_acc = new_history.history['val_accuracy']
new_loss = new_history.history['loss']
new_val_loss = new_history.history['val_loss']
epochs_range = range(7)

plt.figure(figsize=(15, 15))
plt.subplot(2, 2, 1)
plt.plot(epochs_range, new_acc, label='Training Accuracy')
plt.plot(epochs_range, new_val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(epochs_range, new_loss, label='Training Loss')
plt.plot(epochs_range, new_val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Step 4: Results and Analysis

In [46]:

```
# check what index keras has internally assigned to each label
print(validation_generator.class_indices)
```

```
{'0': 0, '1': 1}
```

In [47]:

```
# Get the true labels
y_true = validation_generator.classes
```

Since our dataset is not heavily imbalanced then I would like to use ROC AUC as an evaluation metric for binary classification problem. The Receiver Operator Characteristic (ROC) is a probability curve that plots the TPR against FPR at various threshold values and essentially separates the 'signal' from the 'noise.' In other words, it shows the performance of a classification model at all classification thresholds. The Area Under the Curve (AUC) is the measure of the ability of a binary classifier to distinguish between classes and is used as a summary of the ROC curve.

The higher the AUC, the better the model's performance at distinguishing between the positive and negative classes.

Model 1

In [48]:

```
val_loss1, val_acc1 = model.evaluate(validation_generator)
print('val_loss_model1:', val_loss1)
print('val_acc_model1:', val_acc1)
```

```
3200/3200 [==============================] - 102s 32ms/step - loss: 0.2670 - acc
uracy: 0.8864
val_loss_model1: 0.26701804995536804
val_acc_model1: 0.8864062428474426
```

In [49]:

```
# predict validation dataset
predictions1 = model.predict(validation_generator, verbose=1)
predictions1
```

```
3200/3200 [==============================] - 101s 32ms/step
```

Out[49]:

```
array([[0.03428283],
       [0.14390497],
       [0.05418937],
       ...,
       [0.1562683 ],
       [0.99227035],
       [0.23418935]], dtype=float32)
```

In [50]:

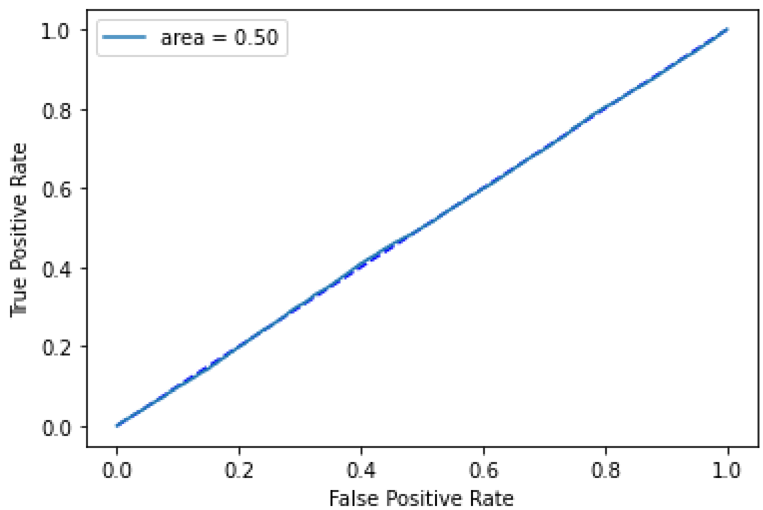
```
# calculate auc_score
fpr1, tpr1, thresholds1 = roc_curve(y_true, predictions1, pos_label=1)
auc_score1 = auc(fpr1, tpr1)
auc_score1
```

Out[50]:

```
0.50013395703125
```

In [52]:

```
plt.plot([0,1], [0,1], linestyle='--', color='blue')
plt.plot(fpr1, tpr1, label='area = {:.2f}'.format(auc_score1))
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```



We can print out the classification report to see the precision and accuracy.

In [53]:

```
# Get the prediction binary
y_pred1 = np.where(predictions1 > 0.5, 1, 0)
```

In [54]:

```
# print out the classification report
print(classification_report(y_true, y_pred1, target_names = ['no_tumor_tissue (C
```

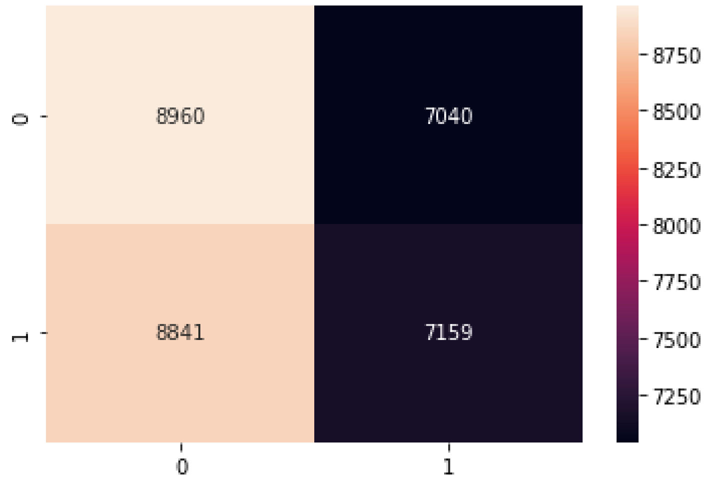
```
                          precision    recall  f1-score   support

 no_tumor_tissue (Class 0)       0.50      0.56      0.53     16000
has_tumor_tissue (Class 1)       0.50      0.45      0.47     16000

                  accuracy                           0.50     32000
                 macro avg       0.50      0.50      0.50     32000
              weighted avg       0.50      0.50      0.50     32000
```

In [55]:

```
# print out the confusion matrix
cm1 = confusion_matrix(y_true, y_pred1)
sns.heatmap(cm1, annot=True, fmt=".0f")
```

Out[55]:

```
<AxesSubplot:>
```



Model 2

In [56]:

```
# the best epoch will be used.
new_model.load_weights('../Week3/checkpoint')
val_loss2, val_acc2 = new_model.evaluate(validation_generator)
print('val_loss_model2:', val_loss2)
print('val_acc_model2:', val_acc2)
```

```
3200/3200 [==============================] - 101s 32ms/step - loss: 0.3265 - acc
uracy: 0.8601
val_loss_model2: 0.32647979259490967
val_acc_model2: 0.8601250052452087
```

In [57]:

```
# predict validation dataset
predictions2 = new_model.predict(validation_generator)
predictions2
```

```
3200/3200 [==============================] - 102s 32ms/step
```

Out[57]:

```
array([[0.432638  ],
       [0.18982337],
       [0.81373936],
       ...,
       [0.98624164],
       [0.12874842],
       [0.6048674 ]], dtype=float32)
```

In [58]:

```
# calculate auc_score
fpr2, tpr2, thresholds2 = roc_curve(y_true, predictions2, pos_label=1)
auc_score2 = auc(fpr2, tpr2)
auc_score2
```

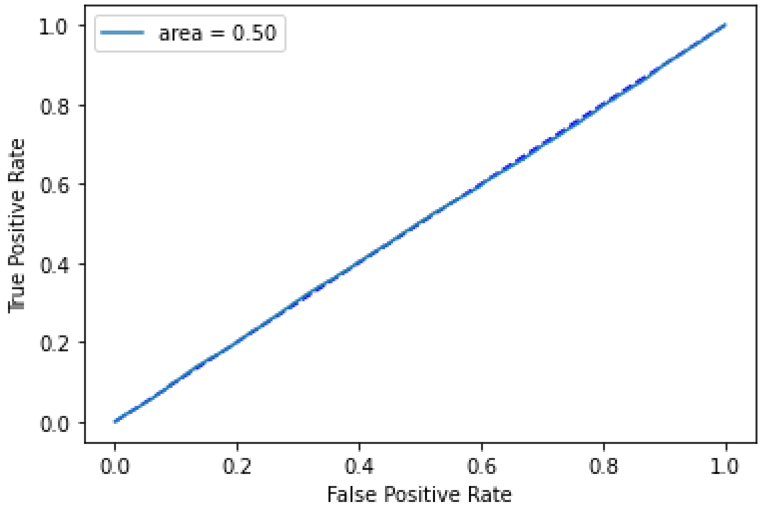
Out[58]:

```
0.499510451171875
```

In [59]:

```
plt.plot([0,1], [0,1], linestyle='--', color='blue')
plt.plot(fpr2, tpr2, label='area = {:.2f}'.format(auc_score2))
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
```

```
# show the plot
plt.show()
```



In [60]:

```
# Get the prediction binary
y_pred2 = np.where(predictions2 > 0.5, 1, 0)
```

In [61]:

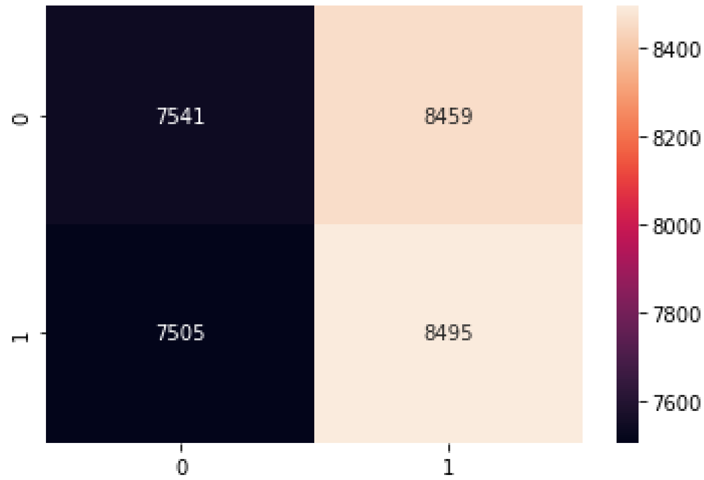
```
# print out the classification report
print(classification_report(y_true, y_pred2, target_names = ['no_tumor_tissue (C
```

```
                          precision    recall  f1-score   support

 no_tumor_tissue (Class 0)       0.50      0.47      0.49     16000
has_tumor_tissue (Class 1)       0.50      0.53      0.52     16000

                  accuracy                           0.50     32000
                 macro avg       0.50      0.50      0.50     32000
              weighted avg       0.50      0.50      0.50     32000
```

In [62]:

```
# print out the confusion matrix
cm2 = confusion_matrix(y_true, y_pred2)
sns.heatmap(cm2, annot=True, fmt=".0f")
```

Out[62]: <AxesSubplot:>



Predict test data and print out the submission

In [63]:
```
test_datagen = ImageDataGenerator(rescale=1./255)
```

In [64]:
```
test_generator = test_datagen.flow_from_dataframe(
                            dataframe=test_df,
                            directory=test_path,
                            x_col="id",
                            y_col=None,
                            batch_size=BATCH_SIZE,
                            shuffle=False,
                            seed=RANDOM_STATE,
                            class_mode=None,
                            target_size=(64,64))
```

```
Found 57458 validated image filenames.
```

In [65]:
```
# predict validation dataset
t_predictions = new_model.predict(test_generator, verbose=1)
t_predictions
```

Out[65]:
```
5746/5746 [==============================] - 108s 19ms/step
array([[0.13753963],
       [0.0328943 ],
       [0.17902558],
       ...,
       [0.03642212],
       [0.02054871],
       [0.4093984 ]], dtype=float32)
```

In [67]:
```
# Get the new prediction binary
test_pred = np.where(t_predictions > 0.5, 1, 0)
```

In [68]:
```
# create submission dataframe
test_predictions = np.transpose(test_pred)[0]
submission = pd.DataFrame()
submission['id'] = test_df['id'].apply(lambda x: x.split('.')[0])
```

```
submission['label'] = test_predictions
submission.head()
```

Out[68]:

	id	label
0	fd0a060ef9c30c9a83f6b4bfb568db74b099154d	0
1	1f9ee06f06d329eb7902a2e03ab3835dd0484581	0
2	19709bec800f372d0b1d085da6933dd3ef108846	0
3	7a34fc34523063f13f0617f7518a0330f6187bd3	0
4	93be720ca2b95fe2126cf2e1ed752bd759e9b0ed	0

In [69]:

```
# view test prediction counts
submission['label'].value_counts()
```

Out[69]:

```
0    43117
1    14341
Name: label, dtype: int64
```

In [70]:

```
# plot the count of each label
fig, ax = plt.subplots(figsize=(6,6))
sns.countplot(data=submission, y='label', ax=ax).set(title='\nFigure 5. The Coun

# plot the proportion of each label
labels = submission['label'].unique().tolist()
counts = submission['label'].value_counts()
sizes = [counts[v] for v in labels]
fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%0.2f%%')
ax1.axis('equal')
plt.title("\nFigure 6. The Proportion of Each Label\n")
plt.tight_layout()
plt.show()
```

Figure 5. The Count of Each Label

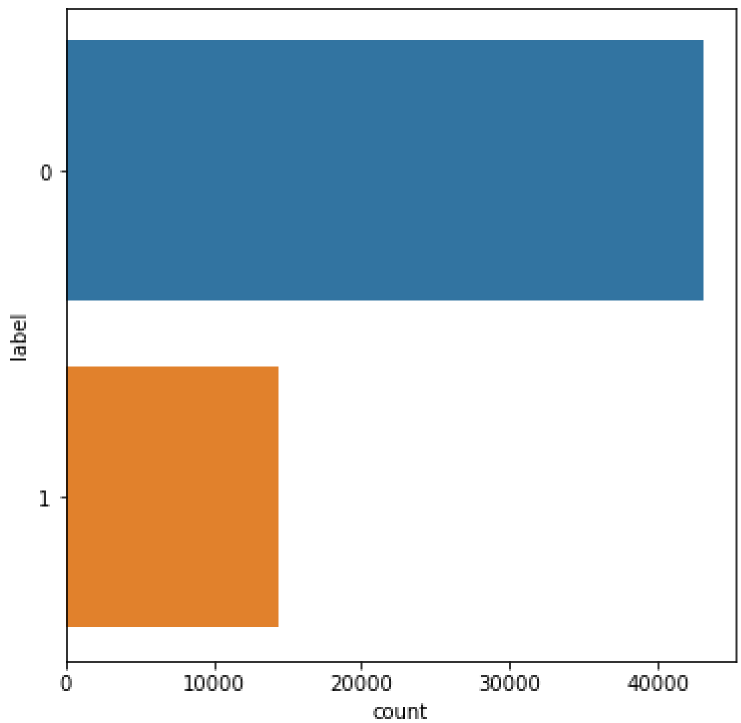
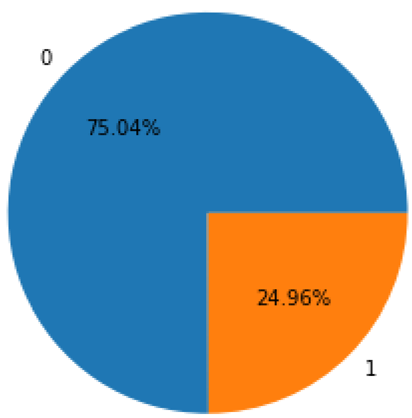


Figure 6. The Proportion of Each Label



In [71]:
```
# convert to csv to submit to competition
#submission.to_csv('submission.csv', index=False)
submission.to_csv('../Week3/submission.csv', index=False)
```

Step 5: Conclusion

In [78]:
```
compare_table = pd.DataFrame({"Model": ["Model1", "Model2"],
                        "val_acc": [round(val_acc1, 3), round(val_acc2, 3)],
                        "val_loss": [round(val_loss1, 3), round(val_loss2, 3)],
                        "AUC": [round(auc_score1, 2), round(auc_score2, 2)]})
compare_table
```

Out[78]:

	Model	val_acc	val_loss	AUC
0	Model1	0.886	0.267	0.5
1	Model2	0.860	0.326	0.5

Model 1 has the higher validation accuracy and lower validation loss compare to model2, However AUC score of two models are just the same although it took more time to run model1 than model2 because model2 used Earlystopping and Reduce Learning Rate to optimize the model. I think these two models might be overfitting, so besides these two models, I tried building some models with different learning rate and different values of dense, drop out. For example, when I chose a learning rate like 0.00001, I observed that the model just ran and ended up with an early stop at epoch 4 because of the learning rate was too small, so it was stuck at epoch 4. But due to the limitation of time and memory, I could just build these simple CNN models and get AUC of 0.5. Hence, I believe that there are many ways could improve the result such as run this model by increasing the number of epochs or trying to test with many different parameters might get better results.

In [ ]:

```
```