In [1]:
```
import numpy as np
from tqdm import tqdm
import os
import random
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
import shutil
import tensorflow

import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.layers import *
from tensorflow.keras.models import *
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.compat.v1 import InteractiveSession
from tensorflow.compat.v1 import ConfigProto

config = ConfigProto()
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)
```

In [2]:
```
import keras.layers as KL
import keras.models as KM
import keras
from keras.applications.densenet import DenseNet169
import os
# os.environ['CUDA_DEVICES_VISIBLE'] = '1'
print(keras.__version__)
```

```
2.4.3
```

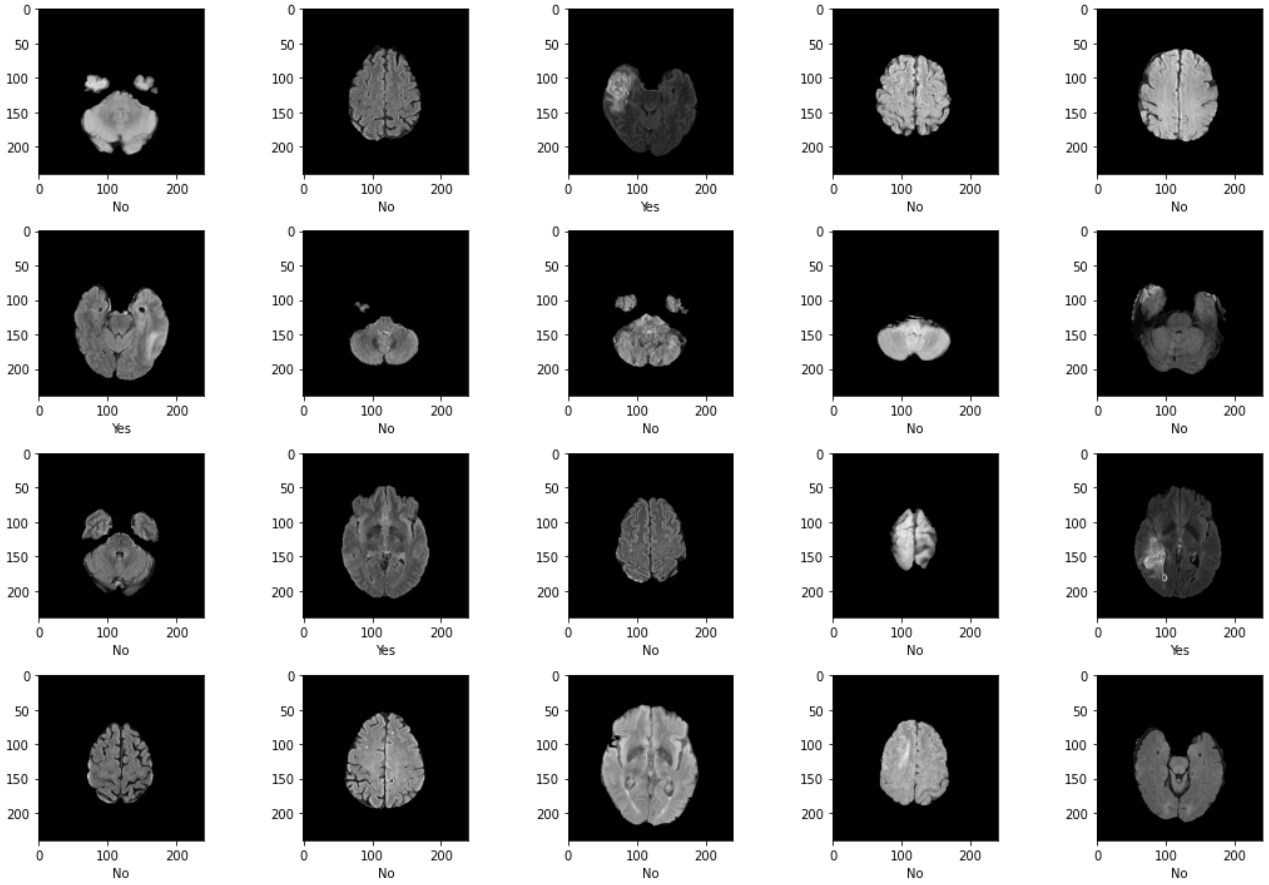
In [ ]:
```
input_image = KL.Input(shape=[512, 512, 3], name="input_image")

resnet50 = DenseNet169(input_tensor=input_image, include_top=True)
# conv5_block1_1_conv
x = resnet50.get_layer('fc1000').output  # block5_pool  max_pool  conv5_block1_1

# resnet50.summery(0)
inputs = [input_image]
outputs = [x]

model = KM.Model(inputs, outputs, name='ctpn')
model.summary()
```

In [8]:
```
#%%

#uncomment if using linux/macos
!rm -rf Train Val
!mkdir Train Val Train/Yes Train/No Val/Yes Val/No

#uncomment if using windows
#!rmdir Train Val /s /q
#!md Train Val Train\Yes Train\No Val\Yes Val\No


img_path = 'Dataset/'
```

```
train_list = []
val_list = []
for CLASS in os.listdir(img_path):
    if not CLASS.startswith('.'):
        all_files = os.listdir(img_path + CLASS)
        files = [item for item in all_files if "img" in item]
        random.shuffle(files)
        img_num = len(files)
        for (n, file_name) in enumerate(files):
            img = os.path.join(img_path,CLASS,file_name)
            seg = os.path.join(img_path,CLASS,file_name.split('_')[0]+'_seg.npy'
            # 80% of images will be used for training, change the number here
            # to use different number of images for training your model.
            if n < 0.8*img_num:
                shutil.copy(img, os.path.join('Train/',CLASS,file_name))
                train_list.append(os.path.join('Train/',CLASS,file_name))
                shutil.copy(seg, os.path.join('Train/',CLASS,file_name.split('_'
            else:
                shutil.copy(img, os.path.join('Val/',CLASS,file_name))
                val_list.append(os.path.join('Val/',CLASS,file_name))
                shutil.copy(seg, os.path.join('Val/',CLASS,file_name.split('_')[
```

In [9]:

```
def plot_samples(img_path,n=20):
    files_list = []
    labels_list = []
    for path, subdirs, all_files in os.walk(img_path):
        files = [item for item in all_files if "img" in item]
        for name in files:
            files_list.append(os.path.join(path, name))
            labels_list.append(path.split('/')[1])
    imgs_lbls = list(zip(files_list, labels_list))
    random.shuffle(imgs_lbls)
    files_list, labels_list = zip(*imgs_lbls)
    j = 5
    i = int(n/j)
    plt.figure(figsize=(15,10))
    k = 1
    for file, lbl in zip(files_list[:n], labels_list[:n]):
        img = np.load(file)
        plt.subplot(i,j,k)
        plt.imshow(img[:,:,0], cmap='gray')
        plt.xlabel(lbl)
        k += 1
    plt.tight_layout()
    plt.show()

plot_samples(img_path)
```



In [10]:
```
class DataGenerator(tensorflow.keras.utils.Sequence):
    'Generates data for Keras'
    def __init__(self, list_IDs, batch_size=32, dim=(240,240), n_channels=3,
                 n_classes=2, shuffle=True):
        'Initialization'
        self.dim = dim
        self.batch_size = batch_size
        self.list_IDs = list_IDs
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.shuffle = shuffle
        self.on_epoch_end()

    def __len__(self):
        'Denotes the number of batches per epoch'
        return int(np.floor(len(self.list_IDs) / self.batch_size))

    def __getitem__(self, index):
        'Generate one batch of data'
        # Generate indexes of the batch
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]

        # Find list of IDs
        list_IDs_temp = [self.list_IDs[k] for k in indexes]

        # Generate data
        X, y = self.__data_generation(list_IDs_temp)

        return X, y

    def on_epoch_end(self):
        'Updates indexes after each epoch'
```

```
        self.indexes = np.arange(len(self.list_IDs))
        if self.shuffle == True:
            np.random.shuffle(self.indexes)

    def __data_generation(self, list_IDs_temp):
        'Generates data containing batch_size samples' # X : (n_samples, *dim, n
        # Initialization
        X = np.empty((self.batch_size, *self.dim, self.n_channels))
        y = np.empty((self.batch_size), dtype=int)

        # Generate data
        for i, ID in enumerate(list_IDs_temp):
            # Store sample
            # Add data augmentation here
            X[i,] = np.load(ID)

            # Store class
            y[i] = min(1,np.sum(np.load(ID.split('_')[0]+'_seg.npy')))

        return X, tensorflow.keras.utils.to_categorical(y, num_classes=self.n_cl
```

In [11]:
```
train_generator = DataGenerator(train_list)
validation_generator = DataGenerator(val_list)
img_size = (240,240)
```

In [14]:
```
base_model = VGG16(
    #uncomment if you want to train your network from scratch.
    weights = None,
    include_top=False,
    input_shape=img_size+(3,)
)
```

In [16]:
```
base_model =DenseNet169(
#uncomment if you want to train your network from scratch.
    weights = None,
    include_top=False,
    input_shape=img_size+(3,)
)
```

In [17]:
```
num_classes = 2

model = Sequential()
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(num_classes, activation='sigmoid'))

# uncomment here if you want to finetune the top layer(classifier) of a pretrain
# model.layers[0].trainable = False

model.compile(
    loss='binary_crossentropy',
    optimizer=Adam(lr=1e-4),
    metrics=['accuracy']
)

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet169 (Model)          (None, 7, 7, 1664)        12642880
_________________________________________________________________
flatten_2 (Flatten)          (None, 81536)             0
_________________________________________________________________
dropout_2 (Dropout)          (None, 81536)             0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 163074
=================================================================
Total params: 12,805,954
Trainable params: 12,647,554
Non-trainable params: 158,400
_________________________________________________________________
```

In [18]:

```
num_epochs = 30
earlystopping = EarlyStopping(
    monitor='accuracy',
    mode='max',
    patience=20
)

history = model.fit(
    train_generator,
    epochs=num_epochs,
    validation_data=validation_generator,
    callbacks=[earlystopping]
)
```

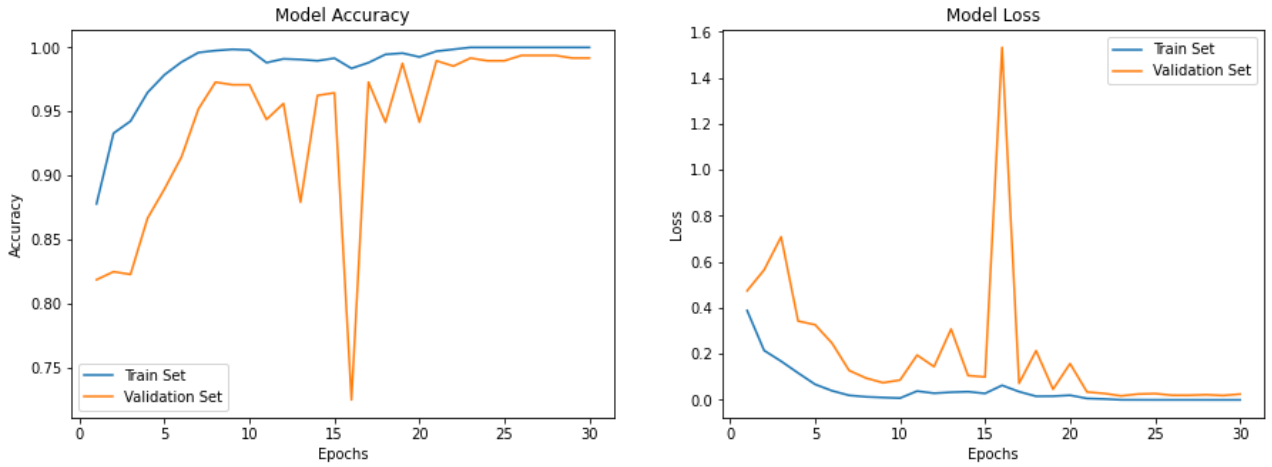
```
Epoch 1/30
63/63 [==============================] - 30s 477ms/step - loss: 0.3883 - accurac
y: 0.8780 - val_loss: 0.4743 - val_accuracy: 0.8188
Epoch 2/30
63/63 [==============================] - 28s 440ms/step - loss: 0.2147 - accurac
y: 0.9330 - val_loss: 0.5649 - val_accuracy: 0.8250
Epoch 3/30
63/63 [==============================] - 28s 441ms/step - loss: 0.1681 - accurac
y: 0.9425 - val_loss: 0.7085 - val_accuracy: 0.8229
Epoch 4/30
63/63 [==============================] - 28s 442ms/step - loss: 0.1170 - accurac
y: 0.9648 - val_loss: 0.3425 - val_accuracy: 0.8667
Epoch 5/30
63/63 [==============================] - 28s 441ms/step - loss: 0.0676 - accurac
y: 0.9787 - val_loss: 0.3267 - val_accuracy: 0.8896
Epoch 6/30
63/63 [==============================] - 28s 440ms/step - loss: 0.0396 - accurac
y: 0.9886 - val_loss: 0.2462 - val_accuracy: 0.9146
Epoch 7/30
63/63 [==============================] - 28s 442ms/step - loss: 0.0198 - accurac
y: 0.9960 - val_loss: 0.1279 - val_accuracy: 0.9521
Epoch 8/30
63/63 [==============================] - 28s 442ms/step - loss: 0.0134 - accurac
y: 0.9975 - val_loss: 0.0946 - val_accuracy: 0.9729
Epoch 9/30
63/63 [==============================] - 28s 442ms/step - loss: 0.0103 - accurac
y: 0.9985 - val_loss: 0.0746 - val_accuracy: 0.9708
Epoch 10/30
63/63 [==============================] - 28s 442ms/step - loss: 0.0078 - accurac
y: 0.9980 - val_loss: 0.0863 - val_accuracy: 0.9708
Epoch 11/30
63/63 [==============================] - 28s 442ms/step - loss: 0.0384 - accurac
y: 0.9881 - val_loss: 0.1947 - val_accuracy: 0.9438
```

```
Epoch 12/30
63/63 [==============================] - 28s 442ms/step - loss: 0.0290 - accurac
y: 0.9911 - val_loss: 0.1444 - val_accuracy: 0.9563
Epoch 13/30
63/63 [==============================] - 28s 442ms/step - loss: 0.0334 - accurac
y: 0.9906 - val_loss: 0.3078 - val_accuracy: 0.8792
Epoch 14/30
63/63 [==============================] - 28s 442ms/step - loss: 0.0357 - accurac
y: 0.9896 - val_loss: 0.1057 - val_accuracy: 0.9625
Epoch 15/30
63/63 [==============================] - 28s 442ms/step - loss: 0.0281 - accurac
y: 0.9916 - val_loss: 0.0995 - val_accuracy: 0.9646
Epoch 16/30
63/63 [==============================] - 28s 442ms/step - loss: 0.0632 - accurac
y: 0.9836 - val_loss: 1.5317 - val_accuracy: 0.7250
Epoch 17/30
63/63 [==============================] - 28s 442ms/step - loss: 0.0359 - accurac
y: 0.9881 - val_loss: 0.0718 - val_accuracy: 0.9729
Epoch 18/30
63/63 [==============================] - 28s 442ms/step - loss: 0.0157 - accurac
y: 0.9945 - val_loss: 0.2137 - val_accuracy: 0.9417
Epoch 19/30
63/63 [==============================] - 28s 442ms/step - loss: 0.0162 - accurac
y: 0.9955 - val_loss: 0.0462 - val_accuracy: 0.9875
Epoch 20/30
63/63 [==============================] - 28s 442ms/step - loss: 0.0202 - accurac
y: 0.9926 - val_loss: 0.1577 - val_accuracy: 0.9417
Epoch 21/30
63/63 [==============================] - 28s 441ms/step - loss: 0.0063 - accurac
y: 0.9970 - val_loss: 0.0343 - val_accuracy: 0.9896
Epoch 22/30
63/63 [==============================] - 28s 441ms/step - loss: 0.0040 - accurac
y: 0.9985 - val_loss: 0.0281 - val_accuracy: 0.9854
Epoch 23/30
63/63 [==============================] - 28s 442ms/step - loss: 4.0771e-04 - acc
uracy: 1.0000 - val_loss: 0.0174 - val_accuracy: 0.9917
Epoch 24/30
63/63 [==============================] - 28s 442ms/step - loss: 2.5055e-04 - acc
uracy: 1.0000 - val_loss: 0.0254 - val_accuracy: 0.9896
Epoch 25/30
63/63 [==============================] - 28s 441ms/step - loss: 1.3710e-04 - acc
uracy: 1.0000 - val_loss: 0.0274 - val_accuracy: 0.9896
Epoch 26/30
63/63 [==============================] - 28s 441ms/step - loss: 1.6784e-04 - acc
uracy: 1.0000 - val_loss: 0.0202 - val_accuracy: 0.9937
Epoch 27/30
63/63 [==============================] - 28s 441ms/step - loss: 1.6833e-04 - acc
uracy: 1.0000 - val_loss: 0.0201 - val_accuracy: 0.9937
Epoch 28/30
63/63 [==============================] - 28s 442ms/step - loss: 1.2176e-04 - acc
uracy: 1.0000 - val_loss: 0.0225 - val_accuracy: 0.9937
Epoch 29/30
63/63 [==============================] - 28s 442ms/step - loss: 1.0489e-04 - acc
uracy: 1.0000 - val_loss: 0.0192 - val_accuracy: 0.9917
Epoch 30/30
63/63 [==============================] - 28s 442ms/step - loss: 8.7275e-05 - acc
uracy: 1.0000 - val_loss: 0.0251 - val_accuracy: 0.9917
```

In [19]:

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(1, len(history.epoch) + 1)
plt.figure(figsize=(15,5))
```

```
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Train Set')
plt.plot(epochs_range, val_acc, label='Validation Set')
plt.legend(loc="best")
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Train Set')
plt.plot(epochs_range, val_loss, label='Validation Set')
plt.legend(loc="best")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Model Loss')
plt.show()
```



In [14]:
```
model.save('trained_model.h5')
```

In [ ]:
```
test_dir = 'Test/'
#load your model here
model = load_model('trained_model.h5')
test_list = []
for CLASS in os.listdir(test_dir):
    if not CLASS.startswith('.'):
        all_files = os.listdir(test_dir + CLASS)
        files = [item for item in all_files if "img" in item]
        for file_name in files:
            test_list.append(test_dir + CLASS + '/' + file_name)
test_generator = DataGenerator(test_list, batch_size=1)

predictions = []
y_test = []
for i in range(test_generator.__len__()):
    x_test, y = test_generator.__getitem__(i)
    y_test.append(y[0][1])
    prediction = model.predict(x_test)
    predictions.append(np.int(prediction[0][1]>0.5))
accuracy = accuracy_score(y_test, predictions)
print('Test Accuracy = %.2f' % accuracy)
```