VGG16-Final Working

February 9, 2020

In [1]:
```
import matplotlib.pyplot as plt
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential
from keras.layers import Dropout, Flatten, Dense
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing import image
from keras.models import Model
from keras.layers import Dense, Flatten
from keras import backend as K
import numpy as np
import pandas as pd
import os
from sklearn.metrics import classification_report, confusion_matrix
import sklearn.metrics as metrics
import sklearn
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
%matplotlib inline
```

Using TensorFlow backend.

In [2]:
```
# create the base pre-trained model
# build the VGG16 network
base_model = applications.VGG16(weights='imagenet', include_top=False,
                                input_shape=(150,150,3))
print('Model loaded.')
base_model.summary()
```

WARNING: Logging before flag parsing goes to stderr.
W0209 11:56:58.908277 139932506134336 deprecation_wrapper.py:119] From /home/mlab/anaconda3/lib/python

W0209 11:56:58.918303 139932506134336 deprecation_wrapper.py:119] From /home/mlab/anaconda3/lib/python

W0209 11:56:58.920446 139932506134336 deprecation_wrapper.py:119] From /home/mlab/anaconda3/lib/python

W0209 11:56:58.938584 139932506134336 deprecation_wrapper.py:119] From /home/mlab/anaconda3/lib/python

W0209 11:56:59.213879 139932506134336 deprecation_wrapper.py:119] From /home/mlab/anaconda3/lib/python

W0209 11:56:59.214377 139932506134336 deprecation_wrapper.py:119] From /home/mlab/anaconda3/lib/python

Model loaded.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808

```
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)       0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

In [3]:
```
# this is the model we will train
model =Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(256,activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0
_________________________________________________________________
```

In [4]:
```
print('Number of trainable weights before freezing: ', len(model.trainable_weights))
## to freesze all convolutional layers in pretrained network method 1
# base_model.trainable=False
```

```
Number of trainable weights before freezing:  30
```

In [5]:
```
# def recall_m(y_true, y_pred):
#     true_positives = K.sum(K.round(K.clip(y_true * y_pred,0,1)))
#     possible_positives = K.sum(K.round(K.clip(y_true,0,1)))
#     recall = true_positives / (possible_positives + K.epsilon())
#     return recall
# def precision_m(y_true, ypred):
#     true_positives = K.sum(K.round(K.clip(y_true * y_pred,0,1)))
```

```
#     predicted_positives = K.sum(K.round(K.clip(y_pred,0,1)))
#     precision = true_positives/(predicted_positives+K.epsilon())
#     return precision

# first: train only the top layers (which were randomly initialized)
# i.e. freeze all convolutional pretrained layers method 2
for layer in base_model.layers:
    layer.trainable = False
print('After freezing: ', len(model.trainable_weights))
# compile the model (should be done *after* setting layers to non-trainable)
model.compile(optimizer=optimizers.Adam(lr=1e-4),metrics=['acc'], loss='binary_crossentropy')
```

```
W0209 11:57:08.031374 139932506134336 deprecation_wrapper.py:119] From /home/mlab/anaconda3/lib/python

W0209 11:57:08.038577 139932506134336 deprecation.py:323] From /home/mlab/anaconda3/lib/python3.7/site-p
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where


After freezing:  4
```

```
In [6]: train_data_dir = '/home/mlab/Documents/brats_hl_data/train'
        validation_data_dir = '/home/mlab/Documents/brats_hl_data/val'
        # 44938
        # 5616
        nb_train_samples = 44938
        nb_validation_samples = 5616
        epochs = 8
        batch_size = 128
        # prepare data augmentation configuration
        train_datagen = ImageDataGenerator(
            rescale=1. / 255,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True)

        test_datagen = ImageDataGenerator(rescale=1. / 255)

        train_generator = train_datagen.flow_from_directory(
            train_data_dir,
            target_size=(150, 150),
            batch_size=batch_size,
            class_mode='binary')

        validation_generator = test_datagen.flow_from_directory(
            validation_data_dir,
            target_size=(150, 150),
```

```
        batch_size=batch_size,
        class_mode='binary')
```

```
Found 44938 images belonging to 2 classes.
Found 5616 images belonging to 2 classes.
```

```
In [7]: true_classes = train_generator.classes
        print(true_classes)
        class_labels = list(train_generator.class_indices.keys())
        print(class_labels)
```

```
[0 0 0 ... 1 1 1]
['high', 'low']
```

```
In [8]: # train the model on the new data for a few epochs
        history =model.fit_generator(train_generator,
                                     steps_per_epoch=nb_train_samples//batch_size,
                                     epochs=epochs,
                                     validation_data=validation_generator,
                                     validation_steps=nb_validation_samples//batch_size)
```

```
Epoch 1/8
351/351 [==============================] - 1358s 4s/step - loss: 0.5562 - acc: 0.6998 - val_lo
Epoch 2/8
351/351 [==============================] - 1354s 4s/step - loss: 0.4820 - acc: 0.7620 - val_lo
Epoch 3/8
351/351 [==============================] - 1353s 4s/step - loss: 0.4489 - acc: 0.7841 - val_lo
Epoch 4/8
351/351 [==============================] - 1358s 4s/step - loss: 0.4282 - acc: 0.7939 - val_lo
Epoch 5/8
351/351 [==============================] - 1355s 4s/step - loss: 0.4059 - acc: 0.8099 - val_lo
Epoch 6/8
351/351 [==============================] - 1353s 4s/step - loss: 0.3907 - acc: 0.8192 - val_lo
Epoch 7/8
351/351 [==============================] - 1346s 4s/step - loss: 0.3817 - acc: 0.8231 - val_lo
Epoch 8/8
351/351 [==============================] - 1353s 4s/step - loss: 0.3601 - acc: 0.8367 - val_lo
```

```
In [9]: true_classes_1 = validation_generator.classes
        print(true_classes)
        class_labels_1 = list(validation_generator.class_indices.keys())
        print(class_labels_1)
```

```
[0 0 0 ... 1 1 1]
['high', 'low']
```

In [10]:
```
#Confution Matrix and Classification Report
Y_pred = model.predict_generator(validation_generator, nb_validation_samples // batch_size+1)
```

In [58]:
```
# y_pred = np.argmax(Y_pred, axis=1)
y_pred = (Y_pred<0.475).astype(np.int)

# print('Confusion Matrix')
# print(confusion_matrix(true_classes_1, y_pred))
# print('Classification Report')
# print(classification_report(validation_generator.classes, y_pred,
#                             target_names=class_labels_1))
```

In [59]:
```
# print(validation_generator.classes)
```

In [60]:
```
confusion_matrix = metrics.confusion_matrix(true_classes_1,y_pred)
print(confusion_matrix)
```

```
[[1503 1272]
 [1599 1242]]
```

In [61]:
```
report= sklearn.metrics.classification_report(true_classes_1, y_pred,
                                              target_names = class_labels_1)
print(report)
```

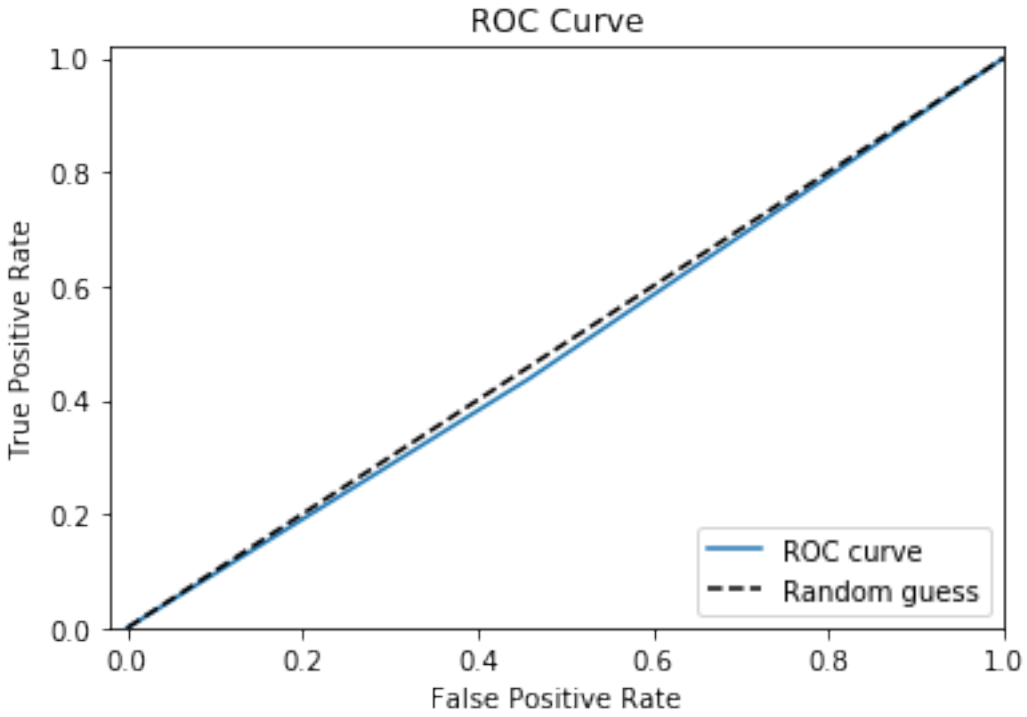
```
              precision    recall  f1-score   support

        high       0.48      0.54      0.51      2775
         low       0.49      0.44      0.46      2841

   micro avg       0.49      0.49      0.49      5616
   macro avg       0.49      0.49      0.49      5616
weighted avg       0.49      0.49      0.49      5616
```

In [62]:
```
fpr, tpr, thresholds = roc_curve(validation_generator.classes, y_pred)

# create plot
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--', label='Random guess')
_ = plt.xlabel('False Positive Rate')
_ = plt.ylabel('True Positive Rate')
_ = plt.title('ROC Curve')
_ = plt.xlim([-0.02, 1])
_ = plt.ylim([0, 1.02])
_ = plt.legend(loc="lower right")
```



In [63]: roc_auc_score(validation_generator.classes, y_pred)

Out[63]: 0.48939581609064187

In [64]:
```
batchX, batchy = train_generator.next()
_, accuracy = model.evaluate(batchX, batchy)
print('Accuracy training: %.2f' % (accuracy*100))
batchXv, batchyv = validation_generator.next()
_, accuracy = model.evaluate(batchXv, batchyv)
print('Accuracy val: %.2f' % (accuracy*100))
```

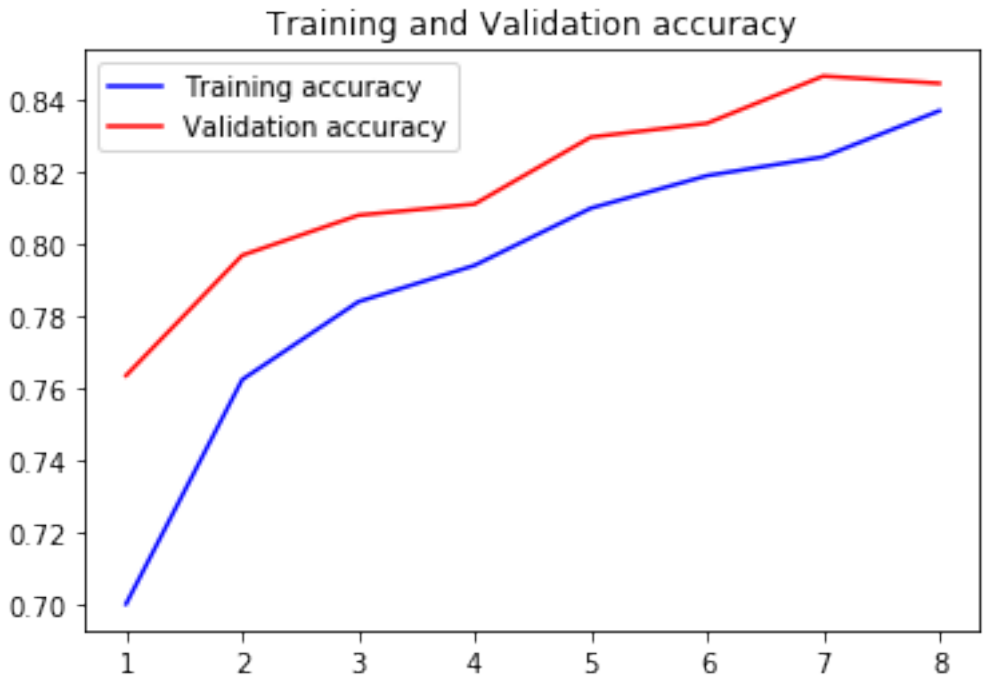
```
128/128 [==============================] - 3s 26ms/step
Accuracy training: 82.03
128/128 [==============================] - 3s 27ms/step
Accuracy val: 84.38
```

In [65]:
```
#plot the train and val curve
#get the details from the history object
acc = history.history['acc']
val_acc=history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

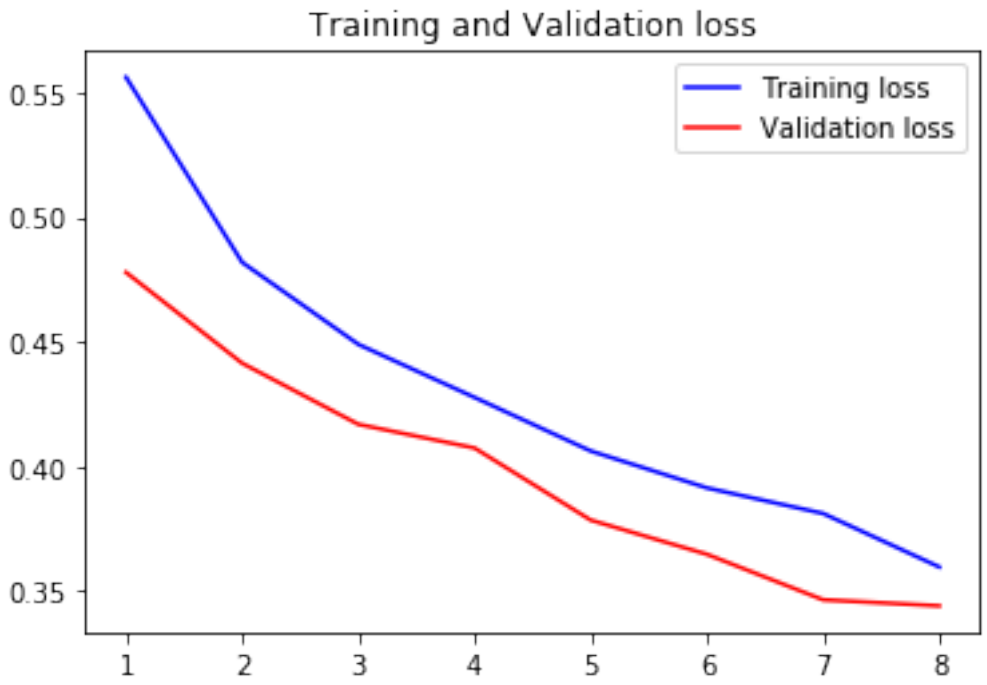
```
epochs = range(1,len(acc)+1)

#train and validation accuracy
plt.plot(epochs,acc,'b',label='Training accuracy')
plt.plot(epochs,val_acc,'r',label='Validation accuracy')
plt.title('Training and Validation accuracy')
plt.legend()
```

Out[65]: <matplotlib.legend.Legend at 0x7f441bccbe10>



In [66]:
```
#train and validation loss
plt.plot(epochs, loss, 'b',label='Training loss')
plt.plot(epochs, val_loss, 'r',label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```



In [67]:
```
test_generator = test_datagen.flow_from_directory('/home/mlab/Documents/brats_hl_data/test',
                                                  class_mode='binary',
                                                  batch_size=batch_size,
                                                  target_size=(150,150))
scores = model.evaluate_generator(test_generator, steps=nb_validation_samples//batch_size)
```

Found 5619 images belonging to 2 classes.

In [68]:
```
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

acc: 84.28%

In [69]:
```
#Confution Matrix and Classification Report
# Y_pred = model.predict_generator(val_generator, 5616 // batch_size)
nb_test_samples=5619
Y_pred1 = model.predict_generator(test_generator,nb_test_samples//batch_size+1)
# y_pred = np.argmax(Y_pred,axis=1)
```

In [70]:
```
true_classes_2 = test_generator.classes
print(true_classes_2)
class_labels_2 = list(test_generator.class_indices.keys())
print(class_labels_2)
```

```
[0 0 0 ... 1 1 1]
['high', 'low']
```

```
In [71]: # y_pred1 = (Y_pred1<0.5).astype(np.int)
         y_pred1 = (Y_pred1 < 0.475).astype(np.int)
         # print(y_pred)
         # print('Confusion Matrix')
         # print(confusion_matrix(true_classes_2, y_pred1))
         # print('Classification Report')
         # print(classification_report(true_classes_2, y_pred1, target_names=class_labels_2))
```

```
In [72]: confusion_matrix1 = metrics.confusion_matrix(true_classes_2,y_pred1)
         print(confusion_matrix1)
```

```
[[1555 1221]
 [1635 1208]]
```

```
In [73]: report1= sklearn.metrics.classification_report(true_classes_2, y_pred1,
                                                       target_names = class_labels_2)
         print(report1)
```

	precision	recall	f1-score	support
high	0.49	0.56	0.52	2776
low	0.50	0.42	0.46	2843
micro avg	0.49	0.49	0.49	5619
macro avg	0.49	0.49	0.49	5619
weighted avg	0.49	0.49	0.49	5619

```
In [81]: print(" Loss: ", scores[0],"\n","Accuracy: ", scores[1])
```

```
 Loss:  0.34708811100139175
 Accuracy:  0.8428415697674418
```