Cancer detection using deep Convolutional Neural Networks

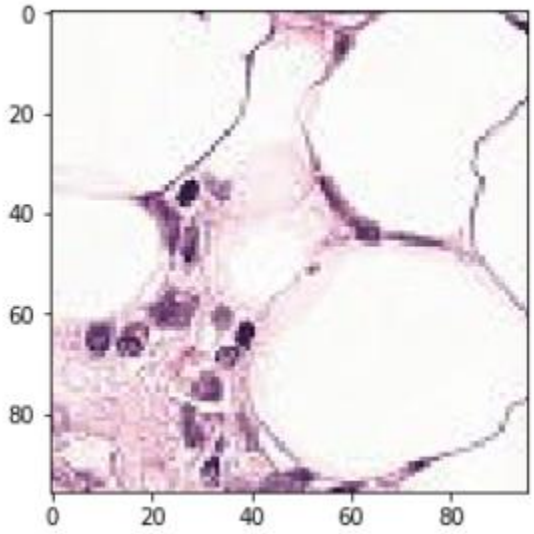
Prithvi Shetty
prithvi1@uw.edu
ASTRO 598

Index:

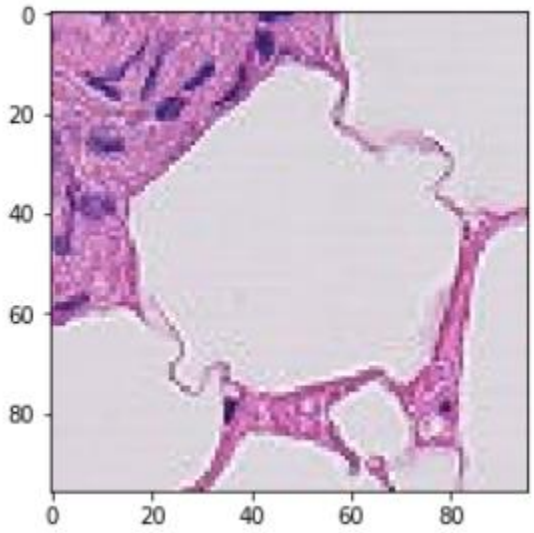
I. Abstract : To detect metastatic cancer in small image patches taken from larger digital pathology scans using deep learning.

II. Data : The data for this competition is a slightly modified version of the PatchCamelyon (PCam) (https://github.com/basveeling/pcam). There are total of 220,000 images in .tif format which are labeled as '1'(tumor) or '0'(non-tumor).

Tumor image example:



Non-tumor image example:



III. Code run through :

1. **Importing libraries.**

   The libraries used in the code are : **Keras, TensorFlow, sci-kit learn, shutil, os, itertools, matplotlib, pandas, numpy, re**

2. **Reading the data.**

```
df=pd.read_csv("C:\\Users\\shett\\Cancer_data\\train_labels\\train_labels.csv")
```

```
df.head(5)
```

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

   Id : Indicates the name of the image (in .tif format)

   Label '1' : Indicates 'tumor'

   Label '0' : Indicates 'not tumor'

3. **Sampling the data for balanced classes and splitting into train and validation.**

   Sampling the data so as to get 50% of tumorous and 50% of non-tumorous images for unbiased classification results.

```
not_tumor=df[df.label==0].sample(85000)
tumor=df[df.label==1].sample(85000)
```

```
not_tumor.shape
tumor.shape
```

```
(85000, 2)
(85000, 2)
```

```
new_df=pd.concat([not_tumor,tumor],axis=0).reset_index(drop=True)
```

Splitting the data with stratified sampling so that the same proportion of labels is maintained in test as well as validation.

```
In [15]: train_data,val_data=train_test_split(new_df, test_size=0.15, random_state=31, stratify=new_df.label) #Stra
         tify makes sure that the same proportion of labels is maintained in test as well as validation

In [16]: train_data.shape
         val_data.shape

Out[16]: (144500, 2)
Out[16]: (25500, 2)
```

4. Creating directories for training and validation

This step is to make the image data augmentation and the train/test process very convenient.

```
train_directory = os.path.join(root, 'train_images')
os.mkdir(train_directory)

val_directory = os.path.join(root, 'val_images')
os.mkdir(val_directory)


#For training images
not_tumor = os.path.join(train_directory, 'not_tumor')
os.mkdir(not_tumor)
tumor = os.path.join(train_directory, 'tumor')
os.mkdir(tumor)


#For validation images
not_tumor2 = os.path.join(val_directory, 'not_tumor')
os.mkdir(not_tumor2)
tumor2 = os.path.join(val_directory, 'tumor')
os.mkdir(tumor2)
```

5. **Move the images in the appropriate folders**

   After creating the directories for the train, test and validation, the images are moved in the appropriate folders.

```
for index,row in val_data.iterrows():
    old_destination2=os.path.join(extract,row.id)

    new_destination2=os.path.join(val_directory,row.label_text, row.id)

    shutil.copyfile(old_destination2, new_destination2)
```

```
print(len(os.listdir("C:\\Users\\shett\\Cancer_data\\train_images\\not_tumor")))
```

```
72250
```

6. **Data augmentation of images and conversion to tensors:**

   The data was augmented by the following features:

   a) Rescaling by a factor of 255.
   b) Rotating it by 40 degrees.
   c) Scaling the width by 20%
   d) Scaling the height by 20%
   e) Shear angle in counter-clockwise direction in degrees.
   f) Zoomed by 20%

g) Flipping the image horizontally.

```
train_datagen = ImageDataGenerator(rescale=1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')


val_datagen = ImageDataGenerator(rescale=1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')


test_datagen = ImageDataGenerator(rescale=1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

7. Model compilation and training:

Model summary: I adapted the rough structure of my model from: https://www.kaggle.com/fmarazzi/baseline-keras-cnn-roc-fast-5min-0-8253-lb

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 94, 94, 32)        896
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 92, 92, 32)        9248
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 90, 90, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 45, 45, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 45, 45, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 43, 43, 64)        18496
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 41, 41, 64)        36928
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 39, 39, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 19, 19, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 19, 19, 64)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 17, 17, 128)       73856
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 15, 15, 128)       147584
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 13, 13, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               1179904
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 514
=================================================================
Total params: 1,661,186
Trainable params: 1,661,186
Non-trainable params: 0
_________________________________________________________________
```

Dropout: 25% used after every Convolutional layer to avoid overfitting.

Activation : Rectified Linear Unit (for every Convolutional layer)

Pooling used: Maxpooling2D to select the maximum of the image feature maps.

Output layer: The output layer used is softmax for this problem to get the probabilistic values of each class.

Convolution layer size: 32, 64 and 128

The model is compiled with Adam's optimizer for weight optimization (Learning rate of 0.0001 and decay rate of 10^-6. The decay rate ensures that the learning rate decreases over time as the training progresses.)

The loss used is : Binary_crossentropy as there are two classes to be predicted.

Epochs: The training goes for 5 epochs and checkpoint ensures that the model is saved which has the best accuracy overall.

```
model.compile(optimizer=Adam(lr=0.0001, decay=1e-6), loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
checkpoint = ModelCheckpoint('cancer_new.h5', monitor='val_acc', verbose=1,
                             save_best_only=True, mode='max')
```

```
history = model.fit_generator(train_generator, steps_per_epoch=train_steps,
                    validation_data=val_generator,
                    validation_steps=val_steps,
                    epochs=5, verbose=1,
                   callbacks=[checkpoint])
```

8. Prediction on test set images:

Final test accuracy : 87.46%
ROC score: 0.958
Final F1-score : 87%

```
val_loss, val_acc = model.evaluate_generator(test_generator,
                        steps=len(val_data))

print('val_loss:', val_loss)
print('val_acc:', val_acc)
```

```
val_loss: 0.30025689363840674
val_acc: 0.8746666666666667
```

```
predictions =model.predict_generator(test_generator, steps=len(val_data), verbose=1)
```

```
25500/25500 [==============================] - 507s 20ms/step
```

```
pred_data = pd.DataFrame(predictions, columns=['not_tumor', 'tumor'])

pred_data.head()
```

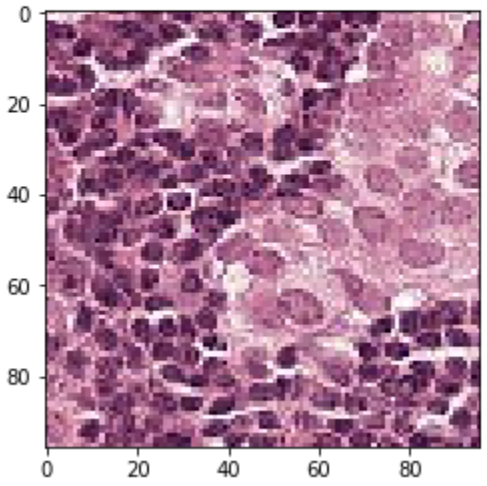
	not_tumor	tumor
0	0.963075	0.036925
1	0.918253	0.081747
2	0.970330	0.029670
3	0.803935	0.196065
4	0.752873	0.247128

Examples of predicted images :

A) Images classified as tumor

```
plt.imshow(plt.imread("{}\\test\\0
```

```
<matplotlib.image.AxesImage at 0x1
```



B) Images classified as non-tumor

```
plt.imshow(plt.imread("{}\\test\\fft
```

```
<matplotlib.image.AxesImage at 0x188
```