Classify histopathology slides of Invasive Ductal Carcinoma (IDC) as either malignant or benign

Note: This is the initial version, accuracy obtained: 84.43%

Breast Cancer Detection using Convolutional Neural Network (CNN)

Dataset Description

Dataset obtained from: http://www.andrewjanowczyk.com/use-case-6-invasive-ductal-carcinoma-idc-segmentation/ (http://www.andrewjanowczyk.com/use-case-6-invasive-ductal-carcinoma-idc-segmentation/) The original dataset consisted of 162 whole mount slide images of Breast Cancer (BCa) specimens scanned at 40x. From that, 277,524 patches of size 50 x 50 were extracted (198,738 IDC negative and 78,786 IDC positive).

Each patch's file name is of the format:

u_xX_yY_classC.png — > example 10253_idx5_x1351_y1101_class0.png

Where u is the patient ID (10253_idx5), X is the x-coordinate of where this patch was cropped from, Y is the y-coordinate of where this patch was cropped from, and C indicates the class where 0 is non-IDC and 1 is IDC.

Tags: Image Classification, Breast Cancer Detection, CNN, Loading Data, Normalization, Data Augmentation, Random Undersampling, Train-test Split, Model Training, Model Evaluation, Prediction, ROC Curve, Precision, Recall, Accuracy

Tools: OpenCV, Python, Fnmatch, Glob, NumPy, Matplotlib, Keras, Scikit-Learn, Imblearn, Jupyter Notebook

Import Dependencies

In [1]:
```
#! pip install opencv-python
```

In [4]:
```
from glob import glob #Finds all the pathnames matching a specified pattern
import fnmatch #Test whether the filename string matches the pattern string
import cv2 #Reading images

import numpy as np #Math

import matplotlib.pyplot as plt
%matplotlib inline

import keras
from keras.utils import to_categorical

#Train-test split
from sklearn.model_selection import train_test_split

#Resampling data
from imblearn.under_sampling import RandomUnderSampler

#Building the CNN model
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.models import load_model

#Evaluation metrics
from sklearn import metrics
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score
```

Extract zip file

In [1]:
```
#Importing required modules
#from zipfile import ZipFile

#Specifying the zip file name
#file_name = "IDC_regular_ps50_idx5.zip" #This file is in the current worki

#Opening the zip file in READ mode
#with ZipFile(file_name, 'r') as zip:
    #Printing all the contents of the zip file
    #zip.printdir()

    #Extracting all the files
    #print('Extracting all the files now...')
    #zip.extractall()
    #print('Done!')
```

The dataset consists of 162 whole mount slide images of breast cancer specimens scanned at 40x. Out of which, 277524 patches of size 50 x 50 were extracted, out of which 198738 are IDC negative (benign) and 78786 are IDC positive (malignant).

Find pathname matching the pattern as follows

In [3]:
```
images = glob('**/*.png', recursive=True)
```

In [5]:
```
patternZero = '*class0.png'
patternOne = '*class1.png'
classZero = fnmatch.filter(images, patternZero) #Saves the file location of
classOne = fnmatch.filter(images, patternOne) #Saves the file location of a]
```

Process the images

In [6]:
```
def process_images(lowerIndex,upperIndex):
    """
    This function returns two arrays:
        x is an array of resized images
        y is an array of labels
    """
    height = 50
    width = 50
    channels = 3
    x = [] #Store image data
    y = [] #Store corresponding class labels
    for img in images[lowerIndex:upperIndex]:
        full_size_image = cv2.imread(img)
        image = (cv2.resize(full_size_image, (width,height), interpolation=c
        x.append(image)
        if img in classZero:
            y.append(0)
        elif img in classOne:
            y.append(1)
        else:
            return
    return x,y
#------------------------------------------------------------------------Pa]
```

In [7]:
```
X, Y = process_images(0,60000) #Analyze first 60000 images
```

In [8]:
```
X = np.array(X) #Convert to a numpy array
```

In [9]:
```
X = X.astype(np.float32) #Casting the array to single precision takes half a
```

Normalize pixels

In [10]:
```
X /= 255. #Normalizing the pixels
```

Train-test split

In [11]:
```
#Split the dataset in 80%-20%
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.2)
```

In [12]:
```
Y.count(0) #Checking the number of 0's in the array Y (this denotes number c
```

Out[12]:
```
44506
```

In [13]:
```
Y.count(1) #Checking the number of 1's in the array Y (this denotes number c
```

Out[13]:
```
15494
```

In [14]:
```
y_train.count(1) #Checking the number of 1's in the array y_train
```

Out[14]:
```
12337
```

In [15]:
```
y_train.count(0) #Checking the number of 1's in the array y_train
```

Out[15]:
```
35663
```

In [17]:
```
#One-Hot-Encode y_train and y_test
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

In [18]:
```
X_trainShape = X_train.shape[1]*X_train.shape[2]*X_train.shape[3]
X_testShape = X_test.shape[1]*X_test.shape[2]*X_test.shape[3]
X_trainFlat = X_train.reshape(X_train.shape[0], X_trainShape)
X_testFlat = X_test.reshape(X_test.shape[0], X_testShape)
```

Randomly undersample the majority class to battle class imbalance

In [19]:
```
#Random undersampling to battle class imbalance
random_under_sampler = RandomUnderSampler(ratio='majority')
X_trainRus, Y_trainRus = random_under_sampler.fit_sample(X_trainFlat, y_trai
X_testRus, Y_testRus = random_under_sampler.fit_sample(X_testFlat, y_test)
```

In [20]:
```
#One-hot-encoding
Y_trainRusHot = to_categorical(Y_trainRus, num_classes = 2)
Y_testRusHot = to_categorical(Y_testRus, num_classes = 2)
```

In [21]:
```
np.unique(Y_trainRus, return_counts=True)
```

Out[21]:
```
(array([0, 1]), array([12337, 12337]))
```

In [22]:
```
for i in range(len(X_trainRus)):
    height, width, channels = 50,50,3
    X_trainRusReshaped = X_trainRus.reshape(len(X_trainRus),height,width,cha
```

In [23]:
```
for i in range(len(X_testRus)):
    height, width, channels = 50,50,3
    X_testRusReshaped = X_testRus.reshape(len(X_testRus),height,width,channe
```

In [25]:
```
#Define hyperparameters
batch_size = 32
num_classes = 2
epochs = 15
```

Define the CNN model

In [26]:
```
#Building the model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),
                 activation='relu',
                 input_shape=(50,50,3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(Flatten()) #3D feature maps to 1D feature vectors for the dense la
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='sigmoid'))
```

Compile the CNN model

In [27]:
```
model.compile(loss=keras.losses.binary_crossentropy,
              optimizer=keras.optimizers.Adam(lr=0.0001),
              metrics=['accuracy'])
```

Data Augmentation

In [28]:
```
#Data Augmentation
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=180,
    horizontal_flip=True,vertical_flip = True)
```

In [29]:
```
early_stopping_monitor = EarlyStopping(monitor='val_loss', patience=3, mode=
model_checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', mode
```

Model Training

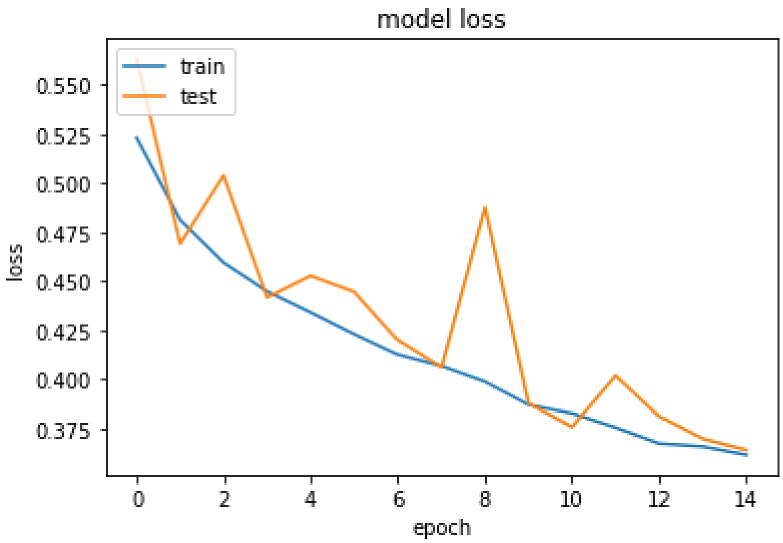
In [30]:
```
#Training the model
training = model.fit_generator(datagen.flow(X_trainRusReshaped,Y_trainRusHot
                    steps_per_epoch=len(X_trainRusReshaped) / batch_size, ep
                    validation_data=(X_testRusReshaped, Y_testRusHot), verbo
                    callbacks=[early_stopping_monitor, model_checkpoint])
```

```
Epoch 1/15

/Users/mrinmayi/anaconda3/lib/python3.6/site-packages/keras_preprocessin
g/image.py:1131: UserWarning: This ImageDataGenerator specifies `featurew
ise_center`, but it hasn't been fit on any training data. Fit it first by
calling `.fit(numpy_data)`.
  warnings.warn('This ImageDataGenerator specifies '
/Users/mrinmayi/anaconda3/lib/python3.6/site-packages/keras_preprocessin
g/image.py:1139: UserWarning: This ImageDataGenerator specifies `featurew
ise_std_normalization`, but it hasn't been fit on any training data. Fit
it first by calling `.fit(numpy_data)`.
  warnings.warn('This ImageDataGenerator specifies '

772/771 [==============================] - 2210s 3s/step - loss: 0.5233 -
acc: 0.7501 - val_loss: 0.5631 - val_acc: 0.7286

Epoch 00001: val_loss improved from inf to 0.56315, saving model to best_
model.h5
Epoch 2/15
772/771 [==============================] - 2197s 3s/step - loss: 0.4815 -
acc: 0.7852 - val_loss: 0.4691 - val_acc: 0.7862

Epoch 00002: val_loss improved from 0.56315 to 0.46910, saving model to b
est_model.h5
Epoch 3/15
772/771 [==============================] - 2220s 3s/step - loss: 0.4597 -
acc: 0.7952 - val_loss: 0.5039 - val_acc: 0.7550

Epoch 00003: val_loss did not improve from 0.46910
Epoch 4/15
772/771 [==============================] - 2242s 3s/step - loss: 0.4447 -
acc: 0.8037 - val_loss: 0.4418 - val_acc: 0.8031

Epoch 00004: val_loss improved from 0.46910 to 0.44180, saving model to b
est_model.h5
Epoch 5/15
772/771 [==============================] - 2242s 3s/step - loss: 0.4340 -
acc: 0.8087 - val_loss: 0.4528 - val_acc: 0.7901

Epoch 00005: val_loss did not improve from 0.44180
Epoch 6/15
772/771 [==============================] - 2245s 3s/step - loss: 0.4227 -
acc: 0.8155 - val_loss: 0.4447 - val_acc: 0.7922

Epoch 00006: val_loss did not improve from 0.44180
Epoch 7/15
772/771 [==============================] - 2365s 3s/step - loss: 0.4123 -
acc: 0.8198 - val_loss: 0.4200 - val_acc: 0.8078

Epoch 00007: val_loss improved from 0.44180 to 0.41999, saving model to b
est_model.h5
```

```
Epoch 8/15
772/771 [==============================] - 2366s 3s/step - loss: 0.4075 -
acc: 0.8216 - val_loss: 0.4062 - val_acc: 0.8261

Epoch 00008: val_loss improved from 0.41999 to 0.40618, saving model to b
est_model.h5
Epoch 9/15
772/771 [==============================] - 2245s 3s/step - loss: 0.3987 -
acc: 0.8268 - val_loss: 0.4876 - val_acc: 0.7663

Epoch 00009: val_loss did not improve from 0.40618
Epoch 10/15
772/771 [==============================] - 2178s 3s/step - loss: 0.3869 -
acc: 0.8330 - val_loss: 0.3880 - val_acc: 0.8310

Epoch 00010: val_loss improved from 0.40618 to 0.38796, saving model to b
est_model.h5
Epoch 11/15
772/771 [==============================] - 2179s 3s/step - loss: 0.3825 -
acc: 0.8358 - val_loss: 0.3760 - val_acc: 0.8391

Epoch 00011: val_loss improved from 0.38796 to 0.37597, saving model to b
est_model.h5
Epoch 12/15
772/771 [==============================] - 2175s 3s/step - loss: 0.3754 -
acc: 0.8411 - val_loss: 0.4020 - val_acc: 0.8284

Epoch 00012: val_loss did not improve from 0.37597
Epoch 13/15
772/771 [==============================] - 2176s 3s/step - loss: 0.3672 -
acc: 0.8434 - val_loss: 0.3813 - val_acc: 0.8407

Epoch 00013: val_loss did not improve from 0.37597
Epoch 14/15
772/771 [==============================] - 9953s 13s/step - loss: 0.3659
- acc: 0.8454 - val_loss: 0.3700 - val_acc: 0.8427

Epoch 00014: val_loss improved from 0.37597 to 0.36998, saving model to b
est_model.h5
Epoch 15/15
772/771 [==============================] - 4885s 6s/step - loss: 0.3615 -
acc: 0.8465 - val_loss: 0.3643 - val_acc: 0.8441

Epoch 00015: val_loss improved from 0.36998 to 0.36434, saving model to b
est_model.h5
```

Plotting train-test loss

In [31]:
```
#Plot the losses
plt.plot(training.history['loss'])
plt.plot(training.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [32]:
```
model = load_model('best_model.h5')

y_pred_one_hot = model.predict(X_testRusReshaped)
y_pred_labels = np.argmax(y_pred_one_hot, axis = 1)

y_true_labels = np.argmax(Y_testRusHot,axis=1)

confusion_matrix = metrics.confusion_matrix(y_true=y_true_labels, y_pred=y_p
print(confusion_matrix)
```

```
[[2533  624]
 [ 359 2798]]
```

Evaluating performance

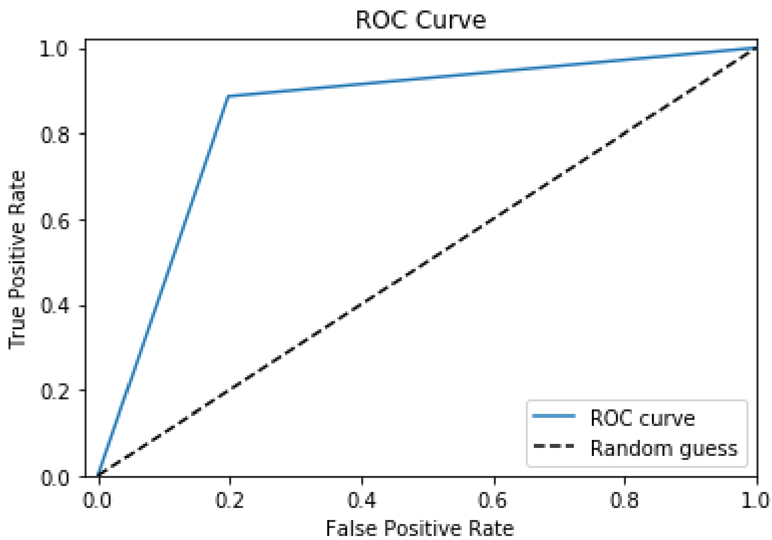
In [34]:
```
#Check Precision, Recall and F1-Score of both classes (0 and 1)
print(classification_report(y_true_labels, y_pred_labels))
```

```
              precision    recall  f1-score   support

           0       0.88      0.80      0.84      3157
           1       0.82      0.89      0.85      3157

   micro avg       0.84      0.84      0.84      6314
   macro avg       0.85      0.84      0.84      6314
weighted avg       0.85      0.84      0.84      6314
```

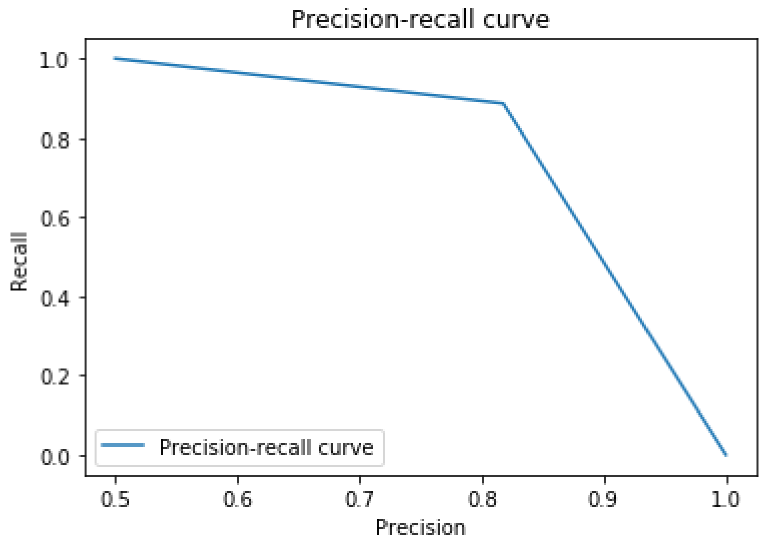
In [35]:
```
#Check the roc_auc_score
roc_auc_score(y_true_labels, y_pred_labels)
```

Out[35]: 0.8443142223630028

In [36]:
```
#ROC Curve
fpr, tpr, thresholds = roc_curve(y_true_labels, y_pred_labels)
#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 [37]:
```
#Precision vs Recall Curve
precision, recall, thresholds = precision_recall_curve(y_true_labels, y_pred
#Create plot
plt.plot(precision, recall, label='Precision-recall curve')
_ = plt.xlabel('Precision')
_ = plt.ylabel('Recall')
_ = plt.title('Precision-recall curve')
_ = plt.legend(loc="lower left")
```



In [39]:
```
#Check accuracy score
accuracy_score(y_true_labels, y_pred_labels)
```

Out[39]: 0.8443142223630028