histopathology-cancer-detection

April 1, 2019

```
In [1]: import pandas as pd
        import numpy as np
        import keras
        import os
        import shutil
        import skimage.io as skio
        from sklearn.model_selection import train_test_split
        from keras.preprocessing.image import ImageDataGenerator
        from keras.applications.densenet import DenseNet121
        from keras.applications import inception_v3,nasnet,mobilenet,vgg19,resnet50,xception
        import matplotlib.pyplot as plt
        import tensorflow as tf
        import keras.backend as K
        from sklearn.utils import shuffle
```

```
Using TensorFlow backend.
```

```
In [2]: file = pd.read_csv("../input/train_labels.csv")
```

```
In [3]: file =file[file['id'] != 'dd6dfed324f9fcb6f93f46f32fc800f2ec196be2']
        file =file[file['id'] != '9369c7278ec8bcc6c880d99194de09fc2bd4efbe']
```

```
In [4]: file.shape
```

```
Out[4]: (220023, 2)
```

```
In [5]: file['label'].value_counts()
```

```
Out[5]: 0    130907
        1     89116
        Name: label, dtype: int64
```

```
In [6]: f_0 = file[file['label'] == 0].sample(80000,random_state = 101)
        f_1 = file[file['label'] == 1].sample(80000,random_state = 101)
        file = pd.concat([f_0,f_1],axis=0).reset_index(drop = True)
        file = shuffle(file)

        file['label'].value_counts()
```

```
Out[6]: 1    80000
        0    80000
        Name: label, dtype: int64
```

```
In [7]: y = file['label']

        x_train,x_valid = train_test_split(file,test_size = 0.20,random_state= 101,stratify=y)

        print(x_train.shape)
        print(x_valid.shape)
```

```
(128000, 2)
(32000, 2)
```

```
In [8]: x_train['label'].value_counts()
```

```
Out[8]: 1    64000
        0    64000
        Name: label, dtype: int64
```

```
In [9]: x_valid['label'].value_counts()
```

```
Out[9]: 1    16000
        0    16000
        Name: label, dtype: int64
```

```
In [10]: def create_folder(folderName):
             if not os.path.exists(folderName):
                 try:
                     os.makedirs(folderName)
                 except OSError as exc:
                     if exc.errno != errno.EEXIST:
                         raise
```

```
In [11]: base_dir = 'data'
         create_folder(base_dir)
```

```
In [12]: train_dir = os.path.join(base_dir,'train_dataset')
         create_folder(train_dir)
         valid_dir = os.path.join(base_dir,'valid_dataset')
         create_folder(valid_dir)
```

```
In [13]: train_tum = os.path.join(train_dir,'0')
         create_folder(train_tum)
         train_notum = os.path.join(train_dir,'1')
         create_folder(train_notum)

         valid_tum = os.path.join(valid_dir,'0')
         create_folder(valid_tum)
         valid_notum = os.path.join(valid_dir,'1')
         create_folder(valid_notum)
```

```
In [14]: # check that the folders have been created
         os.listdir('data/train_dataset//')
```

```
Out[14]: ['1', '0']
```

```
In [15]: # Set the id as the index in df_data
         file.set_index('id', inplace=True)
```

```
In [16]: # Get a list of train and val images
         train_list = list(x_train['id'])
         val_list = list(x_valid['id'])
```

```
In [17]: # Transfer the train images

         for image in train_list:

             # the id in the csv file does not have the .tif extension therefore we add it her
             fname = image + '.tif'
             # get the label for a certain image
             target = file.loc[image,'label']

             # these must match the folder names
             if target == 0:
                 label = '0'
             if target == 1:
                 label = '1'

             # source path to image
             src = os.path.join('../input/train', fname)
             # destination path to image
             dst = os.path.join(train_dir, label, fname)
             # copy the image from the source to the destination
             shutil.copyfile(src, dst)
```

```
In [18]: # Transfer the val images

         for image in val_list:

             # the id in the csv file does not have the .tif extension therefore we add it her
             fname = image + '.tif'
             # get the label for a certain image
             target = file.loc[image,'label']

             # these must match the folder names
             if target == 0:
                 label = '0'
             if target == 1:
                 label = '1'
             # source path to image
             src = os.path.join('../input/train', fname)
             # destination path to image
             dst = os.path.join(valid_dir, label, fname)
             # copy the image from the source to the destination
             shutil.copyfile(src, dst)
```

```
In [19]: batch_size = 90
         epochs = 10
```

```
In [20]: datagen = ImageDataGenerator(rescale=1.0/255,
                         horizontal_flip=True,
                         vertical_flip=True)

         train_gen = datagen.flow_from_directory('data/train_dataset/' ,
                                                 target_size = (96,96) ,
                                                 batch_size = batch_size,
                                                class_mode ='categorical')
```

```
Found 128000 images belonging to 2 classes.
```

```
In [21]: def tr_x(tr_gen):
             for x,y in tr_gen:
                 print(x.shape)
                 yield x
         def tr_y(tr_gen):
             for x,y in tr_gen:
                 yield y
```

```
In [22]: valid_gen = datagen.flow_from_directory('data/valid_dataset/',
                                                 target_size = (96,96),
                                                 batch_size = batch_size,
                                                 class_mode='categorical')


         def va_x(val_gen):
             for x,y in val_gen:
                 yield x
         def va_y(val_gen):
             for x,y in val_gen:
                 yield y
```

```
Found 32000 images belonging to 2 classes.
```

```
In [23]: def patches(mode):

             if (mode == 'valid'):
```

```
        xy = valid_gen
    elif(mode == 'train'):
        xy = train_gen
    else:
        xy = test_gen

    batches = 0
    for x,y in xy:
        s = x.shape
        print(x)
        img = x[:,32:64,32:64,:]
        img = np.resize(img,s)
        batches += 1
#          yield ([img,y],[y,img])
        yield img,y
```

In [24]:
```
patches('valid')
```

Out[24]:
```
<generator object patches at 0x7f7620234d00>
```

In [25]:
```
from keras.models import Sequential
from keras.layers import Conv2D,MaxPool2D,SeparableConv2D,Dropout,Flatten,Dense,Batch
from keras import layers,models
from keras import initializers
```

DENSE NET

In [27]:
```
def pretrained_model(model):
    if model == 'densenet':
        base_model = DenseNet121(include_top=False,weights='imagenet',input_shape = (
    elif model == 'inception':
        base_model = inception_v3.InceptionV3(include_top=False,weights='imagenet',inp
    elif model == 'mobilenet':
        base_model = mobilenet.MobileNet(include_top=False,weights='imagenet',input_s
    elif model == 'vgg':
        base_model = vgg19.VGG19(include_top=False,weights='imagenet',input_shape = (
    elif model == 'resnet':
        base_model = resnet50.ResNet50(include_top=False,weights='imagenet',input_sha
    elif model == 'xception':
        base_model = xception.Xception(include_top=False,weights='imagenet',input_sha

    for layer in base_model.layers:
        layer.trainable = False

    x = base_model.output
    x = Flatten()(x)
    x = Dense(150,activation='relu')(x)
    x = Dropout(0.2)(x)
    predictions = Dense(2,activation='softmax')(x)
```

```
            return models.Model(base_model.input,predictions)
```

```
In [28]: main_model = pretrained_model('vgg')
         main_model.summary()
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.1/v
80142336/80134624 [==============================] - 6s 0us/step
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 96, 96, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 96, 96, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 96, 96, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 48, 48, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 48, 48, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 48, 48, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 24, 24, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 24, 24, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 24, 24, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 24, 24, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 24, 24, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 12, 12, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 12, 12, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 12, 12, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 12, 12, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 12, 12, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 6, 6, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 6, 6, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 6, 6, 512)         2359808
```

```
-----------------------------------------------------------------
block5_conv3 (Conv2D)        (None, 6, 6, 512)         2359808
-----------------------------------------------------------------
block5_conv4 (Conv2D)        (None, 6, 6, 512)         2359808
-----------------------------------------------------------------
block5_pool (MaxPooling2D)   (None, 3, 3, 512)         0
-----------------------------------------------------------------
flatten_1 (Flatten)          (None, 4608)              0
-----------------------------------------------------------------
dense_1 (Dense)              (None, 150)               691350
-----------------------------------------------------------------
dropout_1 (Dropout)          (None, 150)               0
-----------------------------------------------------------------
dense_2 (Dense)              (None, 2)                 302
=================================================================
Total params: 20,716,036
Trainable params: 691,652
Non-trainable params: 20,024,384
-----------------------------------------------------------------
```

```
In [29]: from keras.callbacks import ModelCheckpoint,ReduceLROnPlateau,CSVLogger
         from keras.optimizers import Adam,RMSprop
```

```
In [30]: csv_logger = CSVLogger("result.csv",separator = ",",append=True)

         checkpoint_fp = "vgg_model.h5"
         checkpoint = ModelCheckpoint(checkpoint_fp,monitor='val_acc',
                                       verbose=1,
                                     save_best_only= True,mode='max')

         learning_rate = ReduceLROnPlateau(monitor='val_acc',
                                           factor = 0.1,
                                           patience = 2,
                                           verbose = 1,
                                           mode = 'max',
                                           min_lr = 0.00001)

         callback = [checkpoint,learning_rate,csv_logger]
```

```
In [31]: steps_p_ep_tr =np.ceil(len(x_train)/batch_size)
         steps_p_ep_va =np.ceil(len(x_valid)/batch_size)
```

```
In [32]: main_model.compile(optimizer = Adam(lr=0.0001),
                       loss = 'binary_crossentropy', metrics=['accuracy'])
```

```
In [36]: %time
         my_model = main_model.fit_generator(train_gen,
                                             steps_per_epoch = steps_p_ep_tr,
                                             validation_data = valid_gen,
                                             validation_steps = steps_p_ep_va,
                                             verbose = 1,
                                             epochs = epochs,
                                             callbacks = callback)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.68 ţs
Epoch 1/10
1423/1423 [==============================] - 572s 402ms/step - loss: 0.3979 - acc: 0.8179 - val

Epoch 00001: val_acc improved from -inf to 0.83272, saving model to vgg_model.h5
Epoch 2/10
1423/1423 [==============================] - 561s 394ms/step - loss: 0.3604 - acc: 0.8382 - val

Epoch 00002: val_acc improved from 0.83272 to 0.84309, saving model to vgg_model.h5
Epoch 3/10
1423/1423 [==============================] - 557s 391ms/step - loss: 0.3478 - acc: 0.8464 - val

Epoch 00003: val_acc improved from 0.84309 to 0.84797, saving model to vgg_model.h5
Epoch 4/10
1423/1423 [==============================] - 554s 389ms/step - loss: 0.3376 - acc: 0.8516 - val

Epoch 00004: val_acc did not improve from 0.84797
Epoch 5/10
 610/1423 [===========>...] - ETA: 4:31 - loss: 0.3315 - acc: 0.8552
```

```
In [37]: !ls
```

```
__notebook__.ipynb  __output__.json  data  result.csv  vgg_model.h5
```

```
In [38]: shutil.rmtree('data')
```

```
In [39]: # create test_dir
         test_dir = 'test_dir'
         os.mkdir(test_dir)

         # create test_images inside test_dir
         test_images = os.path.join(test_dir, 'test_images')
         os.mkdir(test_images)
```

```
In [40]: os.listdir('test_dir/')
```

```
Out[40]: ['test_images']
```

```
In [41]: test_list = os.listdir('../input/test')
```

```
        for image in test_list:

            fname = image

            # source path to image
            src = os.path.join('../input/test', fname)
            # destination path to image
            dst = os.path.join(test_images, fname)
            # copy the image from the source to the destination
            shutil.copyfile(src, dst)
```

```
In [42]: test_gen = datagen.flow_from_directory('test_dir/',target_size = (96,96),
                                batch_size = batch_size,
                                class_mode='categorical',
                                shuffle= False)

         def te(te_gen):
             for x,y in te_gen:
                 yield ([x,y],[y,x])
```

```
Found 57458 images belonging to 1 classes.
```

```
In [43]: # make sure we are using the best epoch
         main_model.load_weights('vgg_model.h5')

         predictions = main_model.predict_generator(test_gen, steps=57458, verbose=1)
```

```
11461/57458 [====>...] - ETA: 2:49:08
```

```
In [44]: predictions.shape
```

```
Out[44]: (5166592, 2)
```

```
In [45]: test_preds = np.argmax(predictions,axis = 1)
         test_preds.shape
```

```
Out[45]: (5166592,)
```

```
In [46]: f_preds = pd.DataFrame(test_preds, columns=['label'])

         f_preds.head()
```

```
Out[46]:    label
         0      1
         1      0
         2      1
         3      1
         4      0
```

```
In [ ]: test_filenames = test_gen.filenames

        # add the filenames to the dataframe
        f_preds['file_names'] = test_filenames

        f_preds.head()
```

```
In [ ]: def extract_id(x):

            # split into a list
            a = x.split('/')
            # split into a list
            b = a[1].split('.')
            extracted_id = b[0]

            return extracted_id

        f_preds['id'] = f_preds['file_names'].apply(extract_id)
        f_preds.head()
```

```
In [ ]: submission = pd.DataFrame({'id':f_preds['id'],
                                   'label':f_preds['label'],
                                  }).set_index('id')

        submission.to_csv('submission_dense.csv', columns=['label'])
```