Skin Cancer Detection

Mole Classifier Kernel

Le probleme est relativement simple on doit detecter deux classes est ce que le tumeur qu'on a est ce que appartient a la classe 1 ou 2

1. Bénigne
2. Mauvais

Dans ce noyau, je vais essayer de détecter 2 classes différentes de taupes à l'aide du réseau de neurones à convolution avec keras tensorflow en backendpropagation, puis d'analyser le résultat pour voir comment le modèle peut être utile dans un scénario pratique.

Dans ce noyau, j'ai suivi les 14 étapes suivantes pour la construction et l'évaluation de modèles :

Step 1 : Importing Essential Libraries
Step 2: Loading pictures and making Dictionary of images and labels
Step 3: Categorical Labels
Step 4: Normalization
Step 5: Train and Test Split
Step 6: Model Building
Step 7: Cross-validating model
Step 8: Testing model
Step 9: ResNet50

Step 1 : importing Essential Libraries

In [1]:

```
import os

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
from glob import glob
import seaborn as sns
from PIL import Image
np.random.seed(11) # It's my lucky number
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, KFold, cross_val_score, Gric
from sklearn.metrics import accuracy_score
import itertools

import keras
from keras.utils.np_utils import to_categorical # used for converting labels to or
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras import backend as K
from keras.layers.normalization import BatchNormalization
from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
from keras.optimizers import Adam, RMSprop
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
```

```
from keras.wrappers.scikit_learn import KerasClassifier
from keras.applications.resnet50 import ResNet50
from keras import backend as K
```

```
Using TensorFlow backend.
```

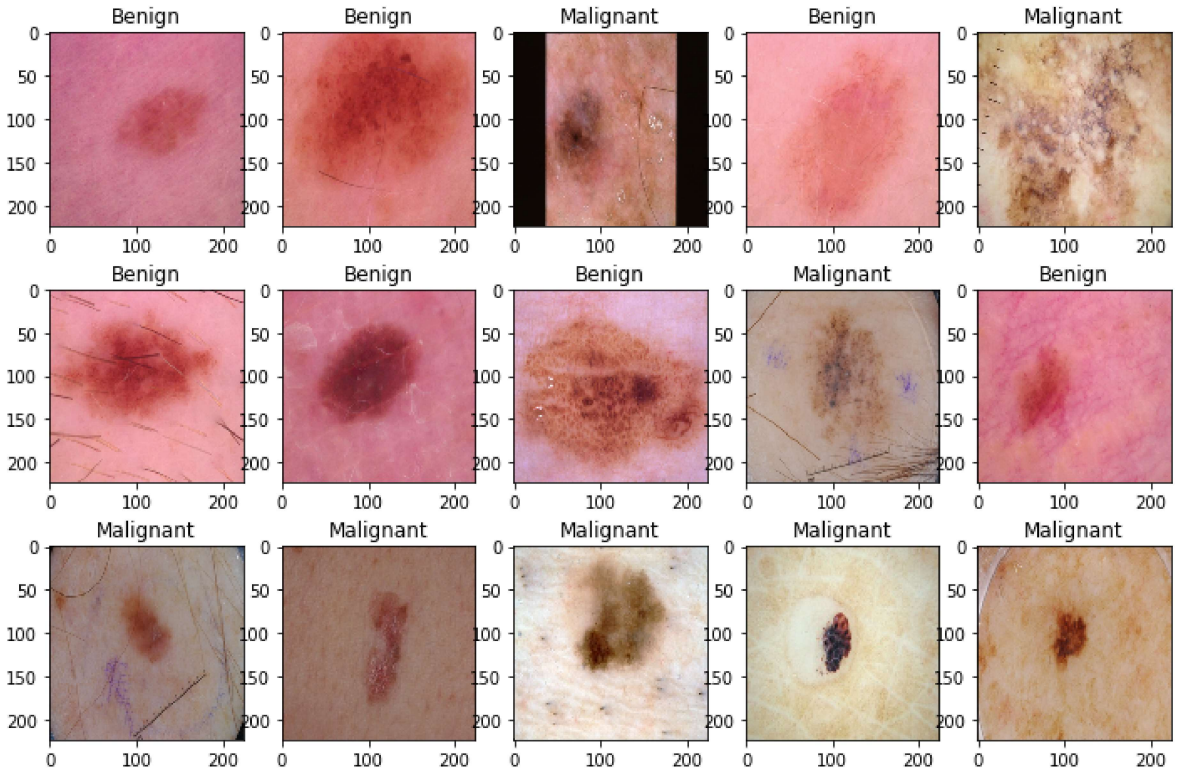
Step 2 : Loading pictures and making Dictionary of images and labels

In this step I load in the pictures and turn them into numpy arrays using their RGB values. As the pictures have already been resized to 224x224, there's no need to resize them. As the pictures do not have any labels, these need to be created. Finally, the pictures are added together to a big training set and shuffeled.

In [2]:

```
folder_benign_train = '../input/data/train/benign'
folder_malignant_train = '../input/data/train/malignant'

folder_benign_test = '../input/data/test/benign'
folder_malignant_test = '../input/data/test/malignant'

read = lambda imname: np.asarray(Image.open(imname).convert("RGB"))

# Load in training pictures
ims_benign = [read(os.path.join(folder_benign_train, filename)) for filename in os
X_benign = np.array(ims_benign, dtype='uint8')
ims_malignant = [read(os.path.join(folder_malignant_train, filename)) for filename
X_malignant = np.array(ims_malignant, dtype='uint8')

# Load in testing pictures
ims_benign = [read(os.path.join(folder_benign_test, filename)) for filename in os.
X_benign_test = np.array(ims_benign, dtype='uint8')
ims_malignant = [read(os.path.join(folder_malignant_test, filename)) for filename
X_malignant_test = np.array(ims_malignant, dtype='uint8')

# Create labels
y_benign = np.zeros(X_benign.shape[0])
y_malignant = np.ones(X_malignant.shape[0])

y_benign_test = np.zeros(X_benign_test.shape[0])
y_malignant_test = np.ones(X_malignant_test.shape[0])


# Merge data
X_train = np.concatenate((X_benign, X_malignant), axis = 0)
y_train = np.concatenate((y_benign, y_malignant), axis = 0)

X_test = np.concatenate((X_benign_test, X_malignant_test), axis = 0)
y_test = np.concatenate((y_benign_test, y_malignant_test), axis = 0)

# Shuffle data
s = np.arange(X_train.shape[0])
np.random.shuffle(s)
X_train = X_train[s]
y_train = y_train[s]

s = np.arange(X_test.shape[0])
np.random.shuffle(s)
X_test = X_test[s]
y_test = y_test[s]
```

In [3]:
```
# Display first 15 images of moles, and how they are classified
w=40
h=30
fig=plt.figure(figsize=(12, 8))
columns = 5
rows = 3

for i in range(1, columns*rows +1):
    ax = fig.add_subplot(rows, columns, i)
    if y_train[i] == 0:
        ax.title.set_text('Benign')
    else:
        ax.title.set_text('Malignant')
    plt.imshow(X_train[i], interpolation='nearest')
plt.show()
```



Step 3: Categorical Labels

Turn labels into one hot encoding

In [4]:
```
y_train = to_categorical(y_train, num_classes= 2)
y_test = to_categorical(y_test, num_classes= 2)
```

Step 4 : Normalization

Normalize all Values of the pictures by dividing all the RGB values by 255

In [5]:
```
# With data augmentation to prevent overfitting
X_train = X_train/255.
X_test = X_test/255.
```

Step 5: Model Building

CNN

I used the Keras Sequential API, where you have just to add one layer at a time, starting from the input.

The first is the convolutional (Conv2D) layer. It is like a set of learnable filters. I choosed to set 64 filters for the two firsts conv2D layers. Each filter transforms a part of the image (defined by the kernel size) using the kernel filter. The kernel filter matrix is applied on the whole image. Filters can be seen as a transformation of the image.

The CNN can isolate features that are useful everywhere from these transformed images (feature maps).

The second important layer in CNN is the pooling (MaxPool2D) layer. This layer simply acts as a downsampling filter. It looks at the 2 neighboring pixels and picks the maximal value. These are used to reduce computational cost, and to some extent also reduce overfitting. We have to choose the pooling size (i.e the area size pooled each time) more the pooling dimension is high, more the downsampling is important.

Combining convolutional and pooling layers, CNN are able to combine local features and learn more global features of the image.

Dropout is a regularization method, where a proportion of nodes in the layer are randomly ignored (setting their wieghts to zero) for each training sample. This drops randomly a propotion of the network and forces the network to learn features in a distributed way. This technique also improves generalization and reduces the overfitting.

'relu' is the rectifier (activation function max(0,x). The rectifier activation function is used to add non linearity to the network.

The Flatten layer is use to convert the final feature maps into a one single 1D vector. This flattening step is needed so that you can make use of fully connected layers after some convolutional/maxpool layers. It combines all the found local features of the previous convolutional layers.

In the end i used the features in one fully-connected (Dense) layer which is just artificial an neural networks (ANN) classifier.

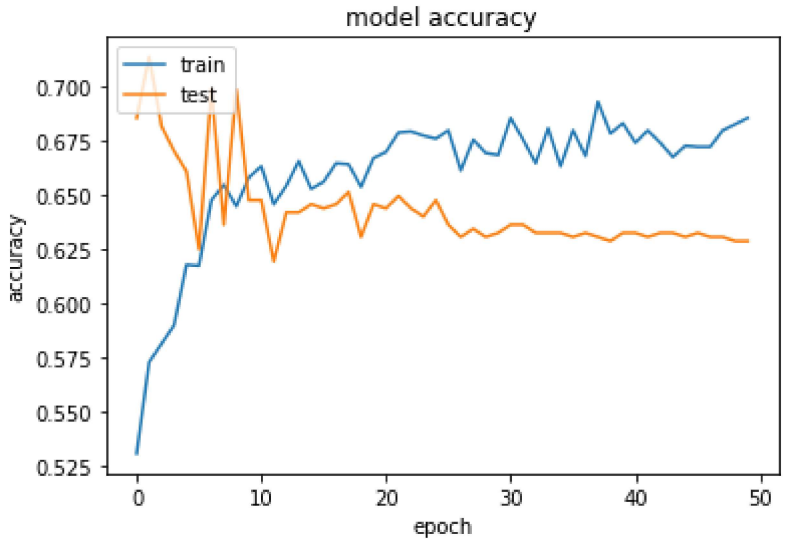
In [6]:
```
# See learning curve and validation curve

def build(input_shape= (224,224,3), lr = 1e-3, num_classes= 2,
          init= 'normal', activ= 'relu', optim= 'adam'):
    model = Sequential()
    model.add(Conv2D(64, kernel_size=(3, 3),padding = 'Same',input_shape=input_sha
                     activation= activ, kernel_initializer='glorot_uniform'))
    model.add(MaxPool2D(pool_size = (2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, kernel_size=(3, 3),padding = 'Same',
```

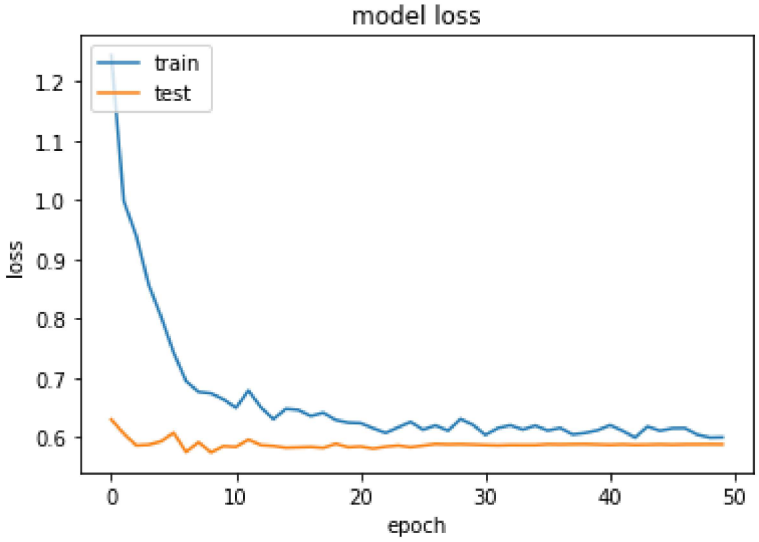
```
                     activation =activ, kernel_initializer = 'glorot_uniform'))
    model.add(MaxPool2D(pool_size = (2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer=init))
    model.add(Dense(num_classes, activation='softmax'))
    model.summary()

    if optim == 'rmsprop':
        optimizer = RMSprop(lr=lr)

    else:
        optimizer = Adam(lr=lr)

    model.compile(optimizer = optimizer ,loss = "binary_crossentropy", metrics=["a
    return model

# Set a learning rate annealer
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=5,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=1e-7)
```

In [7]:

```
input_shape = (224,224,3)
lr = 1e-5
init = 'normal'
activ = 'relu'
optim = 'adam'
epochs = 50
batch_size = 64

model = build(lr=lr, init= init, activ= activ, optim=optim, input_shape= input_sha

history = model.fit(X_train, y_train, validation_split=0.2,
                    epochs= epochs, batch_size= batch_size, verbose=0,
                    callbacks=[learning_rate_reduction]
                   )

# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/keras/backend/tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 64)      1792      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 64)      0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 112, 112, 64)      0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 64)      36928     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 64)        0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 56, 56, 64)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 200704)            0         
_________________________________________________________________
dense_1 (Dense)              (None, 128)               25690240  
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 258       
=================================================================
Total params: 25,729,218
Trainable params: 25,729,218
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.

Epoch 00007: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-06.

Epoch 00012: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-06.

Epoch 00017: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-06.

Epoch 00022: ReduceLROnPlateau reducing learning rate to 6.24999984211172e-07.

Epoch 00027: ReduceLROnPlateau reducing learning rate to 3.12499992105586e-07.

Epoch 00032: ReduceLROnPlateau reducing learning rate to 1.56249996052793e-07.

Epoch 00037: ReduceLROnPlateau reducing learning rate to 1e-07.

Epoch 00042: ReduceLROnPlateau reducing learning rate to 1e-07.

Epoch 00047: ReduceLROnPlateau reducing learning rate to 1e-07.
dict_keys(['val_loss', 'val_acc', 'loss', 'acc', 'lr'])
```





In [8]:

```
K.clear_session()
del model
del history
```

Step 6: Cross-Validating Model

In [9]:

```
# define 3-fold cross validation test harness
kfold = KFold(n_splits=3, shuffle=True, random_state=11)

cvscores = []
for train, test in kfold.split(X_train, y_train):
  # create model
    model = build(lr=lr,
                  init= init,
                  activ= activ,
                  optim=optim,
                  input_shape= input_shape)

    # Fit the model
    model.fit(X_train[train], y_train[train], epochs=epochs, batch_size=batch_size
    # evaluate the model
    scores = model.evaluate(X_train[test], y_train[test], verbose=0)
    print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)
    K.clear_session()
```

```
    del model

print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 64)      1792
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 64)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 64)      36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 56, 56, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 200704)            0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               25690240
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 258
=================================================================
Total params: 25,729,218
Trainable params: 25,729,218
Non-trainable params: 0
_________________________________________________________________
acc: 63.94%
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 64)      1792
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 64)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 64)      36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 56, 56, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 200704)            0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               25690240
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 258
=================================================================
Total params: 25,729,218
Trainable params: 25,729,218
Non-trainable params: 0
_________________________________________________________________
acc: 73.38%
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 64)      1792
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 64)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 64)      36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 64)        0
```

```
dropout_2 (Dropout)          (None, 56, 56, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 200704)            0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               25690240
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 258
=================================================================
Total params: 25,729,218
Trainable params: 25,729,218
Non-trainable params: 0
_________________________________________________________________
acc: 69.85%
69.06% (+/- 3.90%)
```

Step 7: Testing the model

First the model has to be fitted with all the data, such that no data is left out.

In [10]:

```
# Fitting model to all data
model = build(lr=lr,
              init= init,
              activ= activ,
              optim=optim,
              input_shape= input_shape)

model.fit(X_train, y_train,
          epochs=epochs, batch_size= batch_size, verbose=0,
          callbacks=[learning_rate_reduction]
         )

# Testing model on test data to evaluate
y_pred = model.predict_classes(X_test)

print(accuracy_score(np.argmax(y_test, axis=1),y_pred))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 64)      1792
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 64)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 64)      36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 56, 56, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 200704)            0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               25690240
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 258
=================================================================
Total params: 25,729,218
Trainable params: 25,729,218
Non-trainable params: 0
_________________________________________________________________
/opt/conda/lib/python3.6/site-packages/keras/callbacks.py:1109: RuntimeWarning: Re
duce LR on plateau conditioned on metric `val_acc` which is not available. Availab
le metrics are: loss,acc,lr
  (self.monitor, ','.join(list(logs.keys()))), RuntimeWarning
0.6818181818181818
```

In [11]:

```
# save model
# serialize model to JSON
model_json = model.to_json()

with open("model.json", "w") as json_file:
    json_file.write(model_json)

# serialize weights to HDF5
model.save_weights("model.h5")
print("Saved model to disk")

# Clear memory, because of memory overload
del model
K.clear_session()
```

```
Saved model to disk
```

Step 8: ResNet50

The CNN above is not a very sophisticated model, thus the resnet50, is also tried

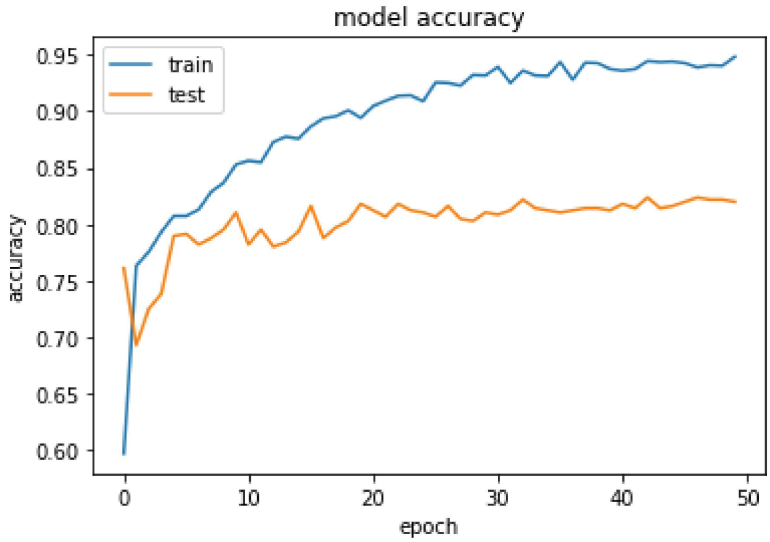
In [12]:

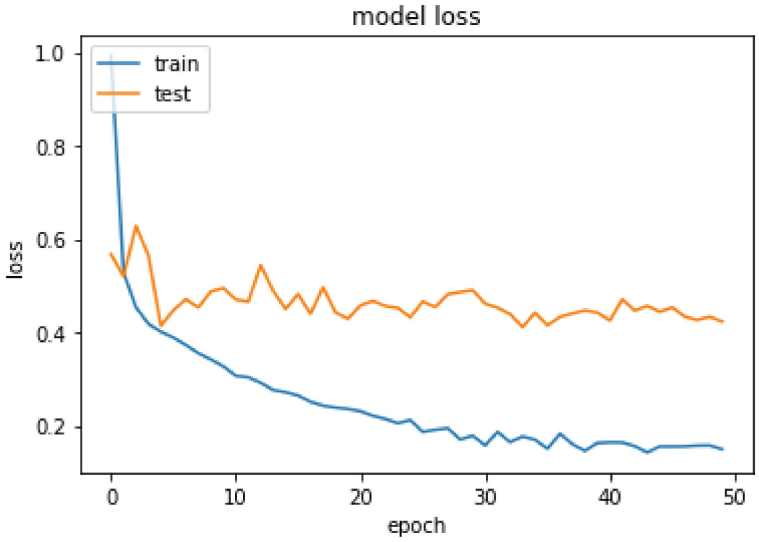
```
input_shape = (224,224,3)
lr = 1e-5
epochs = 50
batch_size = 64

model = ResNet50(include_top=True,
                 weights= None,
                 input_tensor=None,
```

```
                    input_shape=input_shape,
                    pooling='avg',
                    classes=2)

model.compile(optimizer = Adam(lr) ,
              loss = "binary_crossentropy",
              metrics=["accuracy"])

history = model.fit(X_train, y_train, validation_split=0.2,
                    epochs= epochs, batch_size= batch_size, verbose=2,
                    callbacks=[learning_rate_reduction]
                   )

# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
Train on 2109 samples, validate on 528 samples
Epoch 1/50
 - 33s - loss: 0.9927 - acc: 0.5974 - val_loss: 0.5673 - val_acc: 0.7614
Epoch 2/50
 - 18s - loss: 0.5286 - acc: 0.7634 - val_loss: 0.5200 - val_acc: 0.6932
Epoch 3/50
 - 18s - loss: 0.4536 - acc: 0.7757 - val_loss: 0.6281 - val_acc: 0.7254
Epoch 4/50
 - 18s - loss: 0.4181 - acc: 0.7933 - val_loss: 0.5645 - val_acc: 0.7386
Epoch 5/50
 - 18s - loss: 0.4011 - acc: 0.8075 - val_loss: 0.4142 - val_acc: 0.7898
Epoch 6/50
 - 18s - loss: 0.3887 - acc: 0.8075 - val_loss: 0.4471 - val_acc: 0.7917
Epoch 7/50
 - 18s - loss: 0.3729 - acc: 0.8132 - val_loss: 0.4708 - val_acc: 0.7822
Epoch 8/50
 - 18s - loss: 0.3552 - acc: 0.8288 - val_loss: 0.4532 - val_acc: 0.7879
Epoch 9/50
 - 18s - loss: 0.3419 - acc: 0.8369 - val_loss: 0.4875 - val_acc: 0.7955
Epoch 10/50
 - 18s - loss: 0.3273 - acc: 0.8530 - val_loss: 0.4950 - val_acc: 0.8106
Epoch 11/50
 - 18s - loss: 0.3069 - acc: 0.8563 - val_loss: 0.4702 - val_acc: 0.7822
Epoch 12/50
 - 18s - loss: 0.3037 - acc: 0.8549 - val_loss: 0.4657 - val_acc: 0.7955
Epoch 13/50
 - 18s - loss: 0.2920 - acc: 0.8729 - val_loss: 0.5437 - val_acc: 0.7803
Epoch 14/50
 - 18s - loss: 0.2764 - acc: 0.8777 - val_loss: 0.4889 - val_acc: 0.7841
Epoch 15/50
 - 18s - loss: 0.2717 - acc: 0.8758 - val_loss: 0.4500 - val_acc: 0.7936

Epoch 00015: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-06.
Epoch 16/50
 - 18s - loss: 0.2646 - acc: 0.8867 - val_loss: 0.4820 - val_acc: 0.8163
Epoch 17/50
 - 18s - loss: 0.2512 - acc: 0.8938 - val_loss: 0.4397 - val_acc: 0.7879
Epoch 18/50
 - 18s - loss: 0.2429 - acc: 0.8957 - val_loss: 0.4963 - val_acc: 0.7973
Epoch 19/50
 - 18s - loss: 0.2387 - acc: 0.9009 - val_loss: 0.4425 - val_acc: 0.8030
Epoch 20/50
 - 18s - loss: 0.2358 - acc: 0.8943 - val_loss: 0.4292 - val_acc: 0.8182
Epoch 21/50
 - 18s - loss: 0.2309 - acc: 0.9047 - val_loss: 0.4578 - val_acc: 0.8125
Epoch 22/50
 - 18s - loss: 0.2211 - acc: 0.9094 - val_loss: 0.4671 - val_acc: 0.8068
Epoch 23/50
 - 18s - loss: 0.2144 - acc: 0.9137 - val_loss: 0.4563 - val_acc: 0.8182
Epoch 24/50
 - 18s - loss: 0.2052 - acc: 0.9142 - val_loss: 0.4523 - val_acc: 0.8125
Epoch 25/50
 - 18s - loss: 0.2120 - acc: 0.9090 - val_loss: 0.4321 - val_acc: 0.8106

Epoch 00025: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-06.
Epoch 26/50
 - 18s - loss: 0.1866 - acc: 0.9256 - val_loss: 0.4659 - val_acc: 0.8068
Epoch 27/50
 - 18s - loss: 0.1911 - acc: 0.9251 - val_loss: 0.4541 - val_acc: 0.8163
Epoch 28/50
 - 18s - loss: 0.1944 - acc: 0.9227 - val_loss: 0.4816 - val_acc: 0.8049
Epoch 29/50
 - 18s - loss: 0.1703 - acc: 0.9322 - val_loss: 0.4862 - val_acc: 0.8030
Epoch 30/50
```

```
 - 18s - loss: 0.1785 - acc: 0.9317 - val_loss: 0.4903 - val_acc: 0.8106

Epoch 00030: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-06.
Epoch 31/50
 - 18s - loss: 0.1576 - acc: 0.9393 - val_loss: 0.4615 - val_acc: 0.8087
Epoch 32/50
 - 18s - loss: 0.1865 - acc: 0.9251 - val_loss: 0.4525 - val_acc: 0.8125
Epoch 33/50
 - 18s - loss: 0.1647 - acc: 0.9360 - val_loss: 0.4391 - val_acc: 0.8220
Epoch 34/50
 - 18s - loss: 0.1766 - acc: 0.9317 - val_loss: 0.4113 - val_acc: 0.8144
Epoch 35/50
 - 18s - loss: 0.1698 - acc: 0.9312 - val_loss: 0.4421 - val_acc: 0.8125
Epoch 36/50
 - 18s - loss: 0.1507 - acc: 0.9436 - val_loss: 0.4149 - val_acc: 0.8106
Epoch 37/50
 - 18s - loss: 0.1828 - acc: 0.9279 - val_loss: 0.4332 - val_acc: 0.8125
Epoch 38/50
 - 18s - loss: 0.1601 - acc: 0.9431 - val_loss: 0.4404 - val_acc: 0.8144

Epoch 00038: ReduceLROnPlateau reducing learning rate to 6.249999842111172e-07.
Epoch 39/50
 - 18s - loss: 0.1459 - acc: 0.9426 - val_loss: 0.4470 - val_acc: 0.8144
Epoch 40/50
 - 18s - loss: 0.1629 - acc: 0.9374 - val_loss: 0.4423 - val_acc: 0.8125
Epoch 41/50
 - 18s - loss: 0.1641 - acc: 0.9360 - val_loss: 0.4253 - val_acc: 0.8182
Epoch 42/50
 - 18s - loss: 0.1639 - acc: 0.9374 - val_loss: 0.4707 - val_acc: 0.8144
Epoch 43/50
 - 18s - loss: 0.1559 - acc: 0.9445 - val_loss: 0.4466 - val_acc: 0.8239
Epoch 44/50
 - 18s - loss: 0.1425 - acc: 0.9436 - val_loss: 0.4566 - val_acc: 0.8144
Epoch 45/50
 - 18s - loss: 0.1553 - acc: 0.9440 - val_loss: 0.4442 - val_acc: 0.8163
Epoch 46/50
 - 18s - loss: 0.1551 - acc: 0.9426 - val_loss: 0.4533 - val_acc: 0.8201
Epoch 47/50
 - 18s - loss: 0.1555 - acc: 0.9388 - val_loss: 0.4338 - val_acc: 0.8239
Epoch 48/50
 - 18s - loss: 0.1574 - acc: 0.9407 - val_loss: 0.4263 - val_acc: 0.8220

Epoch 00048: ReduceLROnPlateau reducing learning rate to 3.124999921055860e-07.
Epoch 49/50
 - 18s - loss: 0.1576 - acc: 0.9403 - val_loss: 0.4332 - val_acc: 0.8220
Epoch 50/50
 - 18s - loss: 0.1494 - acc: 0.9483 - val_loss: 0.4231 - val_acc: 0.8201
dict_keys(['val_loss', 'val_acc', 'loss', 'acc', 'lr'])
```





In [13]:
```
# Train ResNet50 on all the data
model.fit(X_train, y_train,
          epochs=epochs, batch_size= epochs, verbose=0,
          callbacks=[learning_rate_reduction]
         )
```

```
/opt/conda/lib/python3.6/site-packages/keras/callbacks.py:1109: RuntimeWarning: Re
duce LR on plateau conditioned on metric `val_acc` which is not available. Availab
le metrics are: loss,acc,lr
  (self.monitor, ','.join(list(logs.keys()))), RuntimeWarning
```

Out[13]:
```
<keras.callbacks.History at 0x7fef804cb160>
```

In [14]:
```
# Testing model on test data to evaluate
y_pred = model.predict(X_test)
print(accuracy_score(np.argmax(y_test, axis=1), np.argmax(y_pred, axis=1)))

# save model
# serialize model to JSON
resnet50_json = model.to_json()

with open("resnet50.json", "w") as json_file:
    json_file.write(resnet50_json)

# serialize weights to HDF5
model.save_weights("resnet50.h5")
print("Saved model to disk")
```

```
0.8287878787878787
Saved model to disk
```

codes ends here

author : Ahbar Abdellah

contact : +212 640 6629 54