In [1]:

```
import os
import sys

%matplotlib inline
import matplotlib.pyplot as plt
import skimage.transform
import lasagne
from lasagne.utils import floatX

import theano
from theano import function, config, shared, sandbox
import theano.tensor as T
from theano.compile.nanguardmode import NanGuardMode
import numpy
import time
import os
import pickle
import random

import numpy as np
from scipy.ndimage import imread
from scipy.misc import imresize
from scipy.special import logit, expit
import pandas as pd

from sklearn import cross_validation
from sklearn.metrics import roc_curve, auc
from sklearn.cross_validation import train_test_split

import theano
import theano.tensor as T
import lasagne

#import azureml
#from azureml import services
#azureml.services._DEBUG = True

import pandas
```

```
Using gpu device 0: Tesla K80 (CNMeM is enabled with initial size: 91.0% of m
emory, cuDNN 5105)
```

In [2]:

```
# Seed for reproducibility
np.random.seed(42)
```

Functions for building the GoogLeNet model with Lasagne are defined in googlenet.py:

In [3]:

```
import googlenet
```

Build the model and select layers we need - the features are taken from the final network layer, before the softmax nonlinearity.

In [4]:

```
cnn_layers = googlenet.build_model()
cnn_input_var = cnn_layers['input'].input_var
cnn_feature_layer = cnn_layers['loss3/classifier']
#cnn_output_layer = cnn_layers['prob']
from lasagne.layers import DenseLayer
from lasagne.nonlinearities import softmax
cnn_output_layer = DenseLayer(cnn_layers['loss3/classifier'], num_units=2, non
linearity=softmax)

get_cnn_features = theano.function([cnn_input_var],
lasagne.layers.get_output(cnn_feature_layer))
```

Load the pretrained weights into the network

In [5]:

```
model_param_values = pickle.load(open('./modelzoo/working_blvc_googlenet_3.pk
l'))
lasagne.layers.set_all_param_values(cnn_output_layer, model_param_values)
```

The images need some preprocessing before they can be fed to the CNN

In [6]:

```
MEAN_VALUES = np.array([104, 117, 123]).reshape((3,1,1))

def prep_image(fn, ext='tiff'):
    im = plt.imread(fn, ext)
    if len(im.shape) == 2:
        im = im[:, :, np.newaxis]
        im = np.repeat(im, 3, axis=2)
    # Resize so smallest dim = 224, preserving aspect ratio
    h, w, _ = im.shape
    if h < w:
        im = skimage.transform.resize(im, (224, w*224/h), preserve_range=True)
    else:
        im = skimage.transform.resize(im, (h*224/w, 224), preserve_range=True)

    # Central crop to 224x224
    h, w, _ = im.shape
    im = im[h//2-112:h//2+112, w//2-112:w//2+112]

    rawim = np.copy(im).astype('uint8')

    # Shuffle axes to c01
    im = np.swapaxes(np.swapaxes(im, 1, 2), 0, 1)

    # Convert to BGR
    im = im[::-1, :, :]

    im = im - MEAN_VALUES
    return rawim, floatX(im[np.newaxis])
```

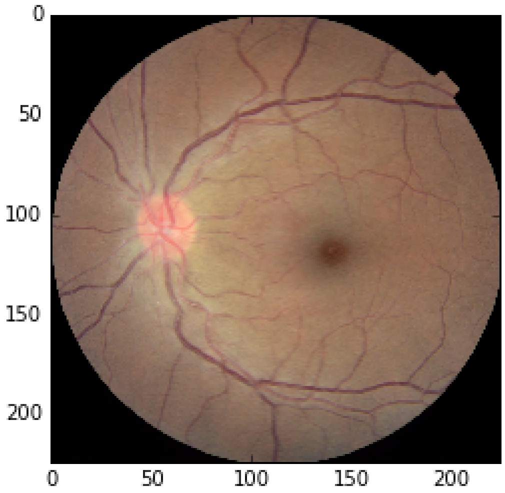
Let's verify that GoogLeNet and our preprocessing are functioning properly

In [7]:

```
# Test preprocesing and show the cropped input
rawim, im = prep_image('./images/NDR/33_left.tiff')
plt.imshow(rawim)
```

Out[7]:

```
<matplotlib.image.AxesImage at 0x7fb8971073d0>
```



In [8]:

```
CLASSES = ['NDR', 'DR']
LABELS = {cls: i for i, cls in enumerate(CLASSES)}
```

In [9]:

```
# Load and preprocess the entire dataset into numpy arrays
X = []
y = []

for cls in CLASSES:
    for fn in os.listdir('./images/test_images/{}'.format(cls)):
        _, im = prep_image('./images/test_images/{}/{}'.format(cls, fn))
        X.append(im)
        y.append(LABELS[cls])

X = np.concatenate(X)
y = np.array(y).astype('int32')
```

In [10]:

```
import skimage.transform
import sklearn.cross_validation

# Split into train, validation and test sets
#train_ix, test_ix = sklearn.cross_validation.train_test_split(range(len(y)))
#train_ix, val_ix = sklearn.cross_validation.train_test_split(range(len(train_
ix)))

#X_tr = X[train_ix]
#y_tr = y[train_ix]

X_val = X
y_val = y
```

In [11]:

```
# Define loss function and metrics, and get an updates dictionary
X_sym = T.tensor4()
y_sym = T.ivector()

prediction = lasagne.layers.get_output(cnn_output_layer, X_sym)
```

In [12]:

```
pred_fn = theano.function([X_sym], prediction)
```

In [13]:

```
def deprocess(im):
    im = im[::-1, :, :]
    im = np.swapaxes(np.swapaxes(im, 0, 1), 1, 2)
    im = (im - im.min())
    im = im / im.max()
    return im
```

In [14]:
```
# Plot some results from the validation set
p_y = pred_fn(X_val).argmax(-1)
print p_y
```

```
[1 1 1 1 1 1 1 0 0 0 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0
 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0
 1 1 0 0 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 0 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

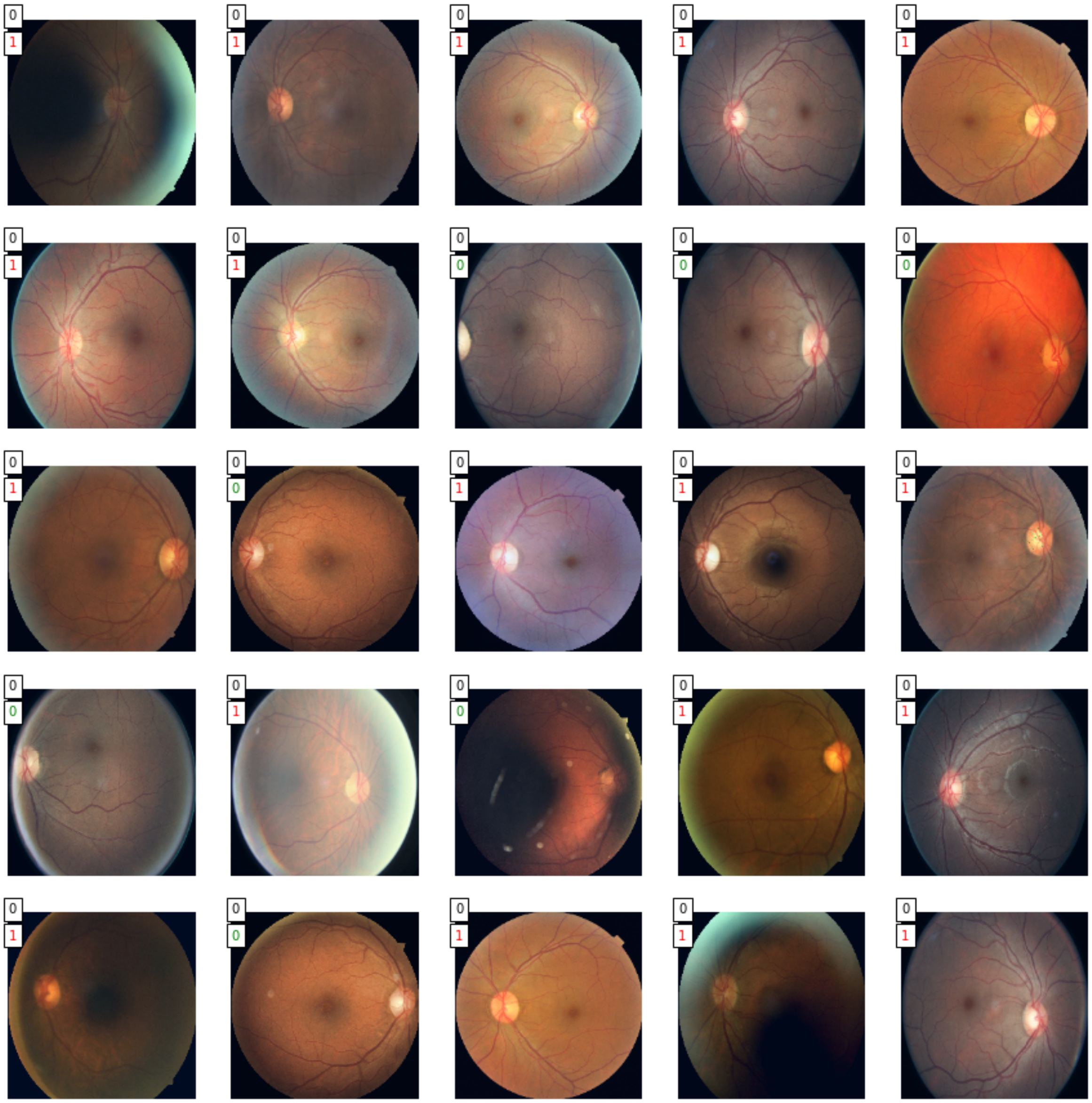
In [15]:
```
import numpy as np
from sklearn.metrics import accuracy_score
y_pred = pd.DataFrame(p_y)
#y_true = [1,0,0,1,0,0,0,1,0,0,1,1,0,1,0,1,0,0,0,1,1,1,0,0,1]
y_true = y_val
print accuracy_score(y_true, y_pred)
```

```
0.580952380952
```

In [16]:

```
plt.figure(figsize=(15, 15))
for i in range(0, 25):
    plt.subplot(5, 5, i+1)
    plt.imshow(deprocess(X_val[i]))
    true = y_val[i]
    pred = p_y[i]
    color = 'green' if true == pred else 'red'
    plt.text(0, 0, true, color='black', bbox=dict(facecolor='white', alpha=1))
    plt.text(0, 32, pred, color=color, bbox=dict(facecolor='white', alpha=1))

    plt.axis('off')
```



In [ ]: