Photo Triage (vgg siamese)

December 19, 2016

1 Photo Triage

In this project we will try to automate the preference between two similar photos based on human preference ground truths. A detailed explaination of the problem is explained in the report

1.0.1 1. Setup

- First, set up Python, `numpy`, and `matplotlib`.

```
In [1]: # set up Python environment: numpy for numerical routines, and matplotlib i
        import numpy as np
        import matplotlib.pyplot as plt
        import glob2
        import string
        # display plots in this notebook
        %matplotlib inline

        # set display defaults
        plt.rcParams['figure.figsize'] = (10, 10)        # large images
        plt.rcParams['image.interpolation'] = 'nearest'  # don't interpolate: show
        plt.rcParams['image.cmap'] = 'gray'  # use grayscale output rather than a
```

- Load `caffe`.

```
In [2]: # The caffe module needs to be on the Python path;
        #  we'll add it here explicitly.
        import sys
        caffe_root = '../'  # this file should be run from {caffe_root}/examples (o

        sys.path.insert(0, caffe_root + 'python')

        import caffe
        # If you get "No module named _caffe", either you have not built pycaffe or
```

- If needed, download the reference model ("CaffeNet", a variant of AlexNet).

```
In [3]: import os
        if os.path.isfile(caffe_root + 'models/vggsms/vgg_siamese.caffemodel'):
            print 'CaffeNet found.'
```

```
        else:
            print 'Downloading pre-trained CaffeNet model...'
            !../scripts/download_model_binary.py ../models/vggsms
```

```
CaffeNet found.
```

1.0.2 2. Load net and set up input preprocessing

- Set Caffe to CPU mode and load the net from disk.
- This part loads the pretrained vgg_siamese model, which is used as a feature extractor from the image

```
In [4]: caffe.set_mode_cpu()

        model_def = caffe_root + 'models/vggsms/deploy.prototxt'
        model_weights = caffe_root + 'models/vggsms/vgg_siamese.caffemodel'

        net = caffe.Net(model_def,      # defines the structure of the model
                        model_weights,  # contains the trained weights
                        caffe.TEST)     # use test mode (e.g., don't perform dropou
```

- Set up input preprocessing. (We'll use Caffe's `caffe.io.Transformer` to do this, but this step is independent of other parts of Caffe, so any custom preprocessing code may be used).

  Our default CaffeNet is configured to take images in BGR format. Values are expected to start in the range [0, 255] and then have the mean ImageNet pixel value subtracted from them. In addition, the channel dimension is expected as the first (*outermost*) dimension.

  As matplotlib will load images with values in the range [0, 1] in RGB format with the channel as the *innermost* dimension, we are arranging for the needed transformations here.

1.0.3 3. Feature extraction from pair of images

- In this part we will consider different combination of pair of similar images and extract a 4096 feature vector. We tap the final layer of the pretrained model to get this feature, it is known as 'fc7' feature since we tap the 7th layer of the deepNet

```
In [5]: def pairwiseLabelImages(image_folder, text_labels_path):

            text_labels = np.loadtxt(text_labels_path, delimiter=' ')
            labelmap = {}
            for i in range( len(text_labels) ):
                if(text_labels[i][4]<text_labels[i][5]):
                    val = [text_labels[i][3], 1, 0]
                else:
                    val = [ text_labels[i][3], 0, 1]

                group_A_B = str(int(text_labels[i][0])) +"_"+ str(int(text_labels[i
                labelmap[group_A_B] = val
```

```
        #for k,v in labelmap.items():
            #print k,v

        labelkeys = labelmap.keys()


        fullfilename = glob2.glob(image_folder + '*.JPG')
        filename = []
        for f in fullfilename:
            filename.append( string.replace(f, image_folder, ''))

        #print "\n".join(sorted(filename))
        groupmap = {}
        for f in filename:
            if(f.split("-")[0] in groupmap):
                val = groupmap[f.split("-")[0]]
                val.append(f.split("-")[1])
                groupmap[f.split("-")[0]] = val
            else:
                groupmap[f.split("-")[0]] = [ f.split("-")[1] ]

        groupkeys = sorted(groupmap.keys())
        #print "Different groups: ", groupkeys

        image_pairs_with_label = []
        for key in groupkeys:
            #print "Groups of photos : ", key,groupmap[key]
            items = sorted(groupmap[key])
            for i in range( len(items)-1 ):
                for j in range( i+1,len(items) ):
                    if(i != j):
                        #print key+'-'+items[i], key+'-'+items[j]
                        group_A_B = str(int(key)) +"_"+ str( int(items[i].split
                        if( group_A_B in labelkeys):
                            #image_pairs_with_label.append( [key+'-'+items[i],
                            image_pairs_with_label.append( [image_folder+key+'-

        #for i in image_pairs_with_label:
            #print i[0], i[1], [i[2][0], i[2][1]]

        return image_pairs_with_label
```

In [6]:
```
import PIL
from PIL import Image

def getFeatures(single_image_pair_with_label):
    img1 = Image.open(single_image_pair_with_label[0])
```

```
img1=img1.resize((224,224),PIL.Image.ANTIALIAS)
img1=np.uint8(img1)
img1= img1[:, :, (2, 1, 0)]
img1 = img1.transpose((2, 0, 1))

img2 = np.uint8(Image.open(single_image_pair_with_label[1]).resize((224
img2= img2[:, :, (2, 1, 0)]
img2 = img2.transpose((2, 0, 1))

img3 = np.concatenate((img1,img2))
img3 = img3;

datum = caffe.io.array_to_datum(img3)

#print img1.shape
#print img2.shape
#print img3.shape
#print {'data': net.blobs['data'].data.shape}

# create transformer for the input called 'data'
transformer = caffe.io.Transformer({'data': net.blobs['data'].data.shap

transformer.set_transpose('data', (2,0,1))  # move image channels to ou
#transformer.set_mean('data', mu)            # subtract the dataset-mea
transformer.set_raw_scale('data', 255)      # rescale from [0, 1] to [0
transformer.set_channel_swap('data', (5,4,3,2,1,0))  # swap channels fr

# set the size of the input (we can skip this if we're happy
#  with the default; we can also change it later, e.g., for different b
net.blobs['data'].reshape(1,          # batch size
                          6,         # 3-channel (BGR) images
                          224, 224)  # image size is 227x227

# Load an image (that comes with Caffe) and perform the preprocessing w

#image = caffe.io.load_image(caffe_root + 'examples/images/cat.jpg')
#transformer = caffe.io.Transformer({'data': net.blobs['data'].data.sha
transformed_image = transformer.preprocess('data', img3)
#plt.imshow(img3)

# Adorable! Let's classify it!

# copy the image data into the memory allocated for the net
net.blobs['data'].data[...] = transformed_image

### perform classification
output = net.forward()
```

```
    output_prob = output['diff'][0]  # the output probability vector for th

    #print output_prob
    #print 'predicted class is:', output_prob.argmax()
    #print len(output_prob)

    return list(output_prob)
```

- In this part we will extract the features and their labels and store it in the .numpy database. These features are loaded later for the trining and validation purposes.

```
In [14]: from sklearn.neural_network import MLPClassifier
         import time

         def writeMatrixToFile(image_pairs_with_labels_list, filename, max_iteratio
             start = time.time()
             loop_c = 0
             matrix = []
             for image in image_pairs_with_labels_list:

                 t0 = time.time()
                 features = getFeatures(image)
                 features.append(image[2][0])
                 matrix.append(features)

                 t = time.time()-t0
                 print "loop", loop_c+1, "of", min(len(image_pairs_with_labels_list
                 loop_c = loop_c + 1

                 if(loop_c == max_iterations):
                     break

             end = time.time()
             print "Total time = ", str(end-start)

             mat = np.array(matrix)
             # type(mat)
             np.save(filename,mat)
             print "Total rows: ", len(matrix), "and columns", len(matrix[0]), "are

In [15]: image_folder = '/home/darshan/ML/pics/'
         text_labels = '/home/darshan/ML/train_pairlist.txt'

         image_pairs_with_label = pairwiseLabelImages(image_folder, text_labels)

In [16]: two_way_image_pairs_with_label = []
         for i in image_pairs_with_label:
             #print i[0], i[1], [i[2][0], i[2][1]]
```

```
            two_way_image_pairs_with_label.append( [i[0], i[1], [i[2][0], i[2][1]]
            two_way_image_pairs_with_label.append( [i[1], i[0], [i[2][1], i[2][0]]

        print "Number of pairs : ", len(image_pairs_with_label)
        print "Number of two way pairs : ", len(two_way_image_pairs_with_label)

        #for i in two_way_image_pairs_with_label:
        #    print i[0], i[1], [i[2][0], i[2][1]]
```

```
Number of pairs :  3023
Number of two way pairs :  6046
```

- We used MLP as out training model since it is found to give the best results among all. Input layer:4096 nurons, First hidden layer:128 nurons, second hidden layer:128 nuron, output layer: single nuron for binary classification.We train the model on 2885 feature vector.

```
In [18]: writeMatrixToFile(two_way_image_pairs_with_label, "train_matrix_2885.npy",
```

```
loop 1 of 2885 with time 6.39626288414
loop 2 of 2885 with time 6.41474080086
loop 3 of 2885 with time 6.4109377861
loop 4 of 2885 with time 6.35202598572
loop 5 of 2885 with time 6.40195798874
loop 6 of 2885 with time 6.42132282257
loop 7 of 2885 with time 6.38849687576
loop 8 of 2885 with time 6.40144395828
loop 9 of 2885 with time 6.41457605362
loop 10 of 2885 with time 6.47701096535
loop 11 of 2885 with time 6.4445078373
loop 12 of 2885 with time 6.48666214943
loop 13 of 2885 with time 6.4600288868
loop 14 of 2885 with time 6.40508008003
loop 15 of 2885 with time 6.43091201782
loop 16 of 2885 with time 6.37551999092
loop 17 of 2885 with time 6.48843693733
loop 18 of 2885 with time 6.47082686424
loop 19 of 2885 with time 6.50606393814
loop 20 of 2885 with time 6.64659690857
loop 21 of 2885 with time 6.67915391922
loop 22 of 2885 with time 6.37243914604
loop 23 of 2885 with time 6.46111679077
loop 24 of 2885 with time 6.48526906967
loop 25 of 2885 with time 6.39492511749
loop 26 of 2885 with time 6.40022492409
loop 27 of 2885 with time 6.34130001068
loop 28 of 2885 with time 6.34775805473
loop 29 of 2885 with time 6.45306301117
loop 30 of 2885 with time 6.40485692024
```

```
loop 2863 of 2885 with time 6.5157418251
loop 2864 of 2885 with time 6.40111017227
loop 2865 of 2885 with time 6.46555781364
loop 2866 of 2885 with time 6.40170311928
loop 2867 of 2885 with time 6.48218488693
loop 2868 of 2885 with time 6.4380979538
loop 2869 of 2885 with time 6.40402698517
loop 2870 of 2885 with time 6.4796090126
loop 2871 of 2885 with time 6.40990710258
loop 2872 of 2885 with time 6.44064903259
loop 2873 of 2885 with time 6.41000413895
loop 2874 of 2885 with time 6.40587997437
loop 2875 of 2885 with time 6.40161299706
loop 2876 of 2885 with time 6.4666659832
loop 2877 of 2885 with time 6.40219187737
loop 2878 of 2885 with time 6.44413495064
loop 2879 of 2885 with time 6.41300797462
loop 2880 of 2885 with time 6.40364003181
loop 2881 of 2885 with time 6.39623403549
loop 2882 of 2885 with time 6.46812009811
loop 2883 of 2885 with time 6.39488101006
loop 2884 of 2885 with time 6.45657610893
loop 2885 of 2885 with time 6.38831186295
Total time =  18438.9980869
Total rows:  2885 and columns 4097 are written into train_matrix_2885.npy
```

```
In [217]: image_folder = '/home/darshan/ML/pics/'
          text_labels = '/home/darshan/ML/val_pairlist.txt'

          image_pairs_with_label_val_1 = pairwiseLabelImages(image_folder, text_lab

          print "Number of pairs : ", len(image_pairs_with_label_val_1)
```

```
Number of pairs :  100
```

```
In [178]: writeMatrixToFile(image_pairs_with_label_val_1, "validation_matrix_100.np
```

```
loop 1 of 100 with time 32.2952570915
loop 2 of 100 with time 8.38135719299
loop 3 of 100 with time 6.92146492004
loop 4 of 100 with time 6.76979112625
loop 5 of 100 with time 6.58378696442
loop 6 of 100 with time 6.46458482742
loop 7 of 100 with time 6.56590294838
loop 8 of 100 with time 6.73145008087
loop 9 of 100 with time 6.57111501694
loop 10 of 100 with time 6.63628411293
```

```
loop 59 of 100 with time 6.52208590508
loop 60 of 100 with time 6.56782889366
loop 61 of 100 with time 6.66256308556
loop 62 of 100 with time 6.60881495476
loop 63 of 100 with time 6.66243696213
loop 64 of 100 with time 6.82626891136
loop 65 of 100 with time 6.60194587708
loop 66 of 100 with time 7.03503203392
loop 67 of 100 with time 6.58157300949
loop 68 of 100 with time 6.64673781395
loop 69 of 100 with time 6.79000806808
loop 70 of 100 with time 6.52996110916
loop 71 of 100 with time 6.4844019413
loop 72 of 100 with time 6.47190594673
loop 73 of 100 with time 6.53633999825
loop 74 of 100 with time 6.71099591255
loop 75 of 100 with time 6.68150687218
loop 76 of 100 with time 6.56575202942
loop 77 of 100 with time 6.52584195137
loop 78 of 100 with time 6.68613409996
loop 79 of 100 with time 6.52901101112
loop 80 of 100 with time 6.5812959671
loop 81 of 100 with time 6.67485308647
loop 82 of 100 with time 6.54997897148
loop 83 of 100 with time 6.61866998672
loop 84 of 100 with time 6.69286394119
loop 85 of 100 with time 6.61522603035
loop 86 of 100 with time 7.24865198135
loop 87 of 100 with time 6.62988901138
loop 88 of 100 with time 6.48882079124
loop 89 of 100 with time 6.40993380547
loop 90 of 100 with time 6.41969013214
loop 91 of 100 with time 6.90267300606
loop 92 of 100 with time 6.62194299698
loop 93 of 100 with time 6.54334688187
loop 94 of 100 with time 6.67910790443
loop 95 of 100 with time 6.6764550209
loop 96 of 100 with time 6.60475587845
loop 97 of 100 with time 6.6842238903
loop 98 of 100 with time 7.02549481392
loop 99 of 100 with time 6.71156811714
loop 100 of 100 with time 6.58508706093
Total time =  692.370344162
Total rows:  100 and columns 4097 are written into validation_matrix_100.npy


In [196]: mat=np.load('train_matrix_2885.npy')
          print "Training Matrix: ", len(mat), "*", len(mat[0])
```

```
          X= mat[:,0:4096]
          print "X: ",len(X), "*", len(X[0])
          y = mat[:,4096]
          print "Y: ",len(y),"* 1"
```

```
Training Matrix:  2885 * 4097
X:  2885 * 4096
Y:  2885 * 1
```

```
In [218]: mat_validation=np.load('validation_matrix_100.npy')

          print "Matrix: ", len(mat_validation), "*", len(mat_validation[0])
          X_validation = mat_validation[:,0:4096]
          print "X: ",len(X_validation), "*", len(X_validation[0])
          y_validation = mat_validation[:,4096]
          print "Y: ",len(y_validation),"* 1"
```

```
Matrix:  100 * 4097
X:  100 * 4096
Y:  100 * 1
```

- Here we are training the model with 2885 pairs of images. MLP classifier with the below configuration gave us the best fit.

```
In [222]: from sklearn.neural_network import MLPClassifier
          import time


          clf = MLPClassifier(solver='adam', alpha=1e-5, hidden_layer_sizes=(128,12
              #clf.Layer('Softmax', warning=None, name='output', units=None, weight

          start = time.time()

          clf.fit(X,y)

          end = time.time()
          print "Total time to fit = ", str(end-start)

          pred_train = list(clf.predict(X))
```

```
Total time to fit =  16.0360620022
```

- In this part we will test out model on the 100 different pairs of images. The test accuracy was found to be 66%. The state of the art model is giving accuracy around 78%.

```
In [228]: pred_validation = list(clf.predict(X_validation))
          #print pred_validation
```

```
        #print y_validation
        print "Training Accuracy =", float(sum(y==pred_train))/len(pred_train)
        print "Validation Accuracy =", float(sum(y_validation==pred_validation))/
        #print (y_validation==pred_validation)

Training Accuracy = 0.622183708839
Validation Accuracy = 0.66
```

- Configuration used for the best MFC classifier model is listed below

```
In [208]: clf.get_params(deep=True)

Out[208]: {'activation': 'tanh',
           'alpha': 1e-05,
           'batch_size': 'auto',
           'beta_1': 0.9,
           'beta_2': 0.999,
           'early_stopping': False,
           'epsilon': 1e-08,
           'hidden_layer_sizes': (128, 128),
           'learning_rate': 'constant',
           'learning_rate_init': 0.001,
           'max_iter': 200,
           'momentum': 0.9,
           'nesterovs_momentum': True,
           'power_t': 0.5,
           'random_state': 1,
           'shuffle': True,
           'solver': 'adam',
           'tol': 0.0001,
           'validation_fraction': 0.1,
           'verbose': False,
           'warm_start': False}
```