Supplement to Alcohol Outlets and Firearm Violence: A place-based case-control study using satellite imagery and machine learning

This code extracts generic features from aerial imagery, then converts these features to 2 latent dimensions for place-based matching. Here we use VGG16, a pretrained convolutional neural network (CNN), and extract features from the 3rd convolutional block. Next we use t-stochastic neighbor embedding to convert 256 features to 2.

While alternative methods, e.g. convolutional autoencoders, could convert imagery into 2D latent space more directly, the approach below takes advantage of VGG16's pretraining, is readily implemented in Python, and builds upon methods (e.g. CNN feature extraction) that may be familiar to novice deep learning users.

Note: the code below was originally written and executed in Python 2. Some code may require updating to run in Python 3. These tasks required < 2hrs to run on an AWS p2.xlarge instance. Running this code without GPU is not recommended for large datasets (e.g. over 5000 locations).

In [ ]:

```
# load packages
import pandas as pd
import numpy as np
import pickle
from sklearn.manifold import TSNE

# requires the Keras deep learning API: https://keras.io/
import keras.applications
from keras.applications import vgg16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.models import Model
from keras import backend as K
```

1. Extract features

In [11]:

```
# load the VGG16 model
model = vgg16.VGG16(weights='imagenet', include_top=True, input_shape=(224,224,3))
model.layers.pop()
model.outputs = [model.layers[-12].output] #extract the output of the pooling layer of th
e 3rd convolutional block
model.layers[-12].outbound_nodes = []
model.summary() #summarize full model for inspection
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
```

```
_________________________________________________________________
block2_conv2 (Conv2D)         (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)    (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)         (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)         (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)         (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)    (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)         (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)         (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)         (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)    (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)         (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)         (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)         (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)    (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)             (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                   (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                   (None, 4096)              16781312
=================================================================
Total params: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0
_________________________________________________________________
```

In [10]:

```
[model.layers[-12].output] #confirm output of the layer from which we'll extract
```

Out[10]:

```
[<tf.Tensor 'block3_pool_2/MaxPool:0' shape=(?, 28, 28, 256) dtype=float32>]
```

In [5]:

```
# load your dataset with one row per location, if easy--otherwise just enter the length m
anually below
dataset_name = <YourDatasetName>
dataset = pd.read_csv(dataset_name)
len(dataset)
```

Out[5]:

```
24408
```

In [ ]:

```
### NOTE: this code is adapted from: http://www.cs.virginia.edu/~vicente/vislang/notebook
s/visual_recognition_lab.html
### Any errors are my own

max = len(dataset)

features = np.zeros((max, 256), dtype=np.float32) #a placeholder for feature values

# process images in batches
batch_size = 339 #a batch size that divides without remainder
```

```
n_batches = max / batch_size

index = 0
for b in range(0, n_batches):
    batch = np.zeros((batch_size, 224, 224, 3))
    print(('Computing features for batch %d of %d') % (b + 1, n_batches))
    for i in range(0, batch_size):
        img_path = #"./Images/im" + format((index+1), '05') + ".png" ### replace with you
r image locations
        img = image.load_img(img_path, target_size=(224, 224))
        img = image.img_to_array(img)
        batch[i, :, :, :] = img
        index = index + 1

    print(('Batch loaded for batch %d of %d') % (b + 1, n_batches))
    batch = vgg16.preprocess_input(batch)
    y = np.mean(model.predict(batch), axis=(1, 2)) #collapse layers to 1D by averaging (
for one 256-long vector per image)
    features[b * batch_size : (b + 1) * batch_size, :] = y

np.save(open('vgg-block3-features.npy', 'w'),
        features)
```

2. Reduce dimensions to 2

In [ ]:

```
#Load image features
features = np.load(open('vgg-block3-features.npy'))

#Implement TSNE over 2 dimensions. Runtime depends on number of images and computing powe
r.
X_embedded = TSNE(n_components=2, perplexity=50, verbose=2).fit_transform(features)

#Save results
np.savetxt("TSNE_mapping.csv", X_embedded)
```

NOTE: matching was conducted in R using a simple loop, with parameters described in the article. Code available upon request to author.