Python For Applied Machine Learning - Practical 5

Michael Halstead

May 6, 2024

1 Introduction

Today we will be covering:

- Using HOG as an input to Bag of Visual Words; and
- Coding up our own K-Nearest Neighbours class.

2 Bag of Visual Words

There are a lot of resources online that can help to understand what bag of visual words is. But, essentially it is a way of clustering similar visual features (like texture information) into groups. The idea is that images of similar classes would contain similar cluster representations. So an image with all vertical stripes would contain cluster groups representing this information. However, there are caveats here, sometimes if your number of clusters is too small you can have too coarse of a representation. The means that your results won't necessarily represent the image well.

2.1 HOG for BoVW

Earlier in the semester we looked at histogram of oriented gradients, remember we needed three inputs to this function, orientations, pixels per cell, and cells per block. We will once again disregard the cells per block and you can set this to [1,1]. For the other two parameters you should create arguments that you can parse through the command line and play around with them seeing what they do.

The first thing you will need to do is create lists of filenames and labels. You will need to separate the filenames into training and evaluation and have the corresponding labels for each. You can do this in any method you can, but below is the pseudo-code for how I did it.

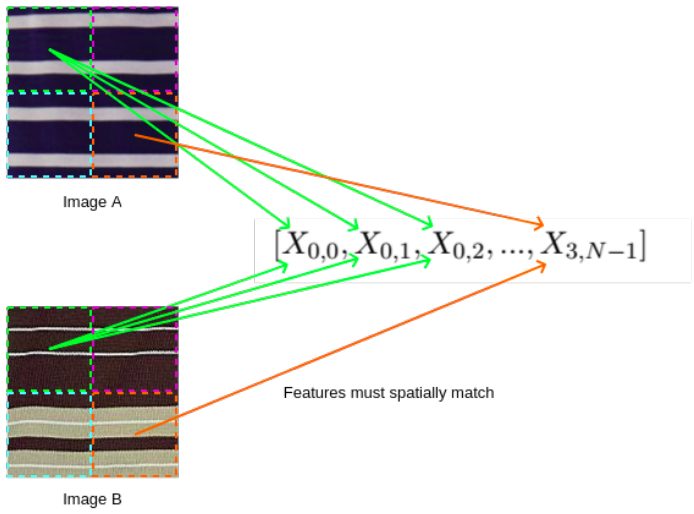

Figure 1: Extracting a single feature vector for an image using the HOG descriptor where N depends on the orientations, pixels per cell and cells per block.

```
def extract_train_eval_files( dataset location, testing split,
    colours to use ):
  create four empty lists for the training files, eval files, train
     labels, and eval labels.
  for all the textures in the dataset
    get the file paths for that texture to all images
    split the files into train and eval (train_test_split)
    store the training files, testing files, training labels, and
    testing labels in your lists.
  return training files, testing files, training labels, and
    testing labels.
```

When we originally used the HOG function from sklearn we disregarded the feature vector and only used the map. For this exercise, we will need the feature vector to cluster our information. The first step in this is to create a function that extracts the HOG information from an image. Your HOG function should have the feature_vector parameter set to *True*, and then you will reshape the output from this to have dimensionality (-1,orientations).

Why do we want to have an (-1,orientations) feature vector? Well let's look at Figure 1. In this case we have split the image into four non-overlapping blocks (let's ignore the cells) with N orientations. Such that the output feature vector would be $FV = [X_{0,0}, X_{0,1}, X_{0,2}, ..., X_{3,N-1}]$. Where the top left box (green) would be represented by $X_{0,0} \rightarrow X_{0,N-1}$ and the bottom right would be $X_{3,0} \rightarrow X_{3,N-1}$.

In this instance we are retaining the spatial representation of the image, so the second image would only match the first if and only if the values in $X_{i,j}$

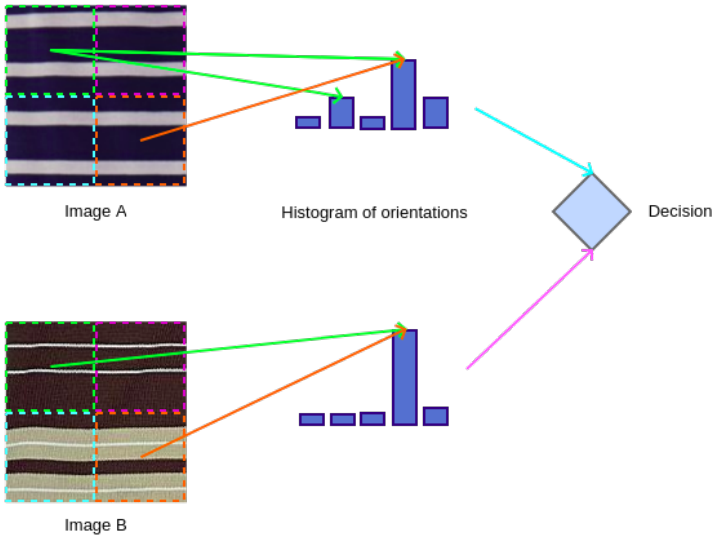

Figure 2: Extracting histograms from HOG features, the orientations are binned such that the spatial information is removed, leaving a global description of the image.

were congruent. So, we need a descriptor that represents the global information within the image without retaining spatial information. Now spatial information can be incredibly important so we don't always do this but in this case we can disregard it. It depends on the task you are trying to achieve, but histograms are a great way of creating a global descriptor.

Figure 2 illustrates this, in this case the spatial representation is removed as any candidate orientation can fill any bin in the histogram. Overall, today, this global representation will be what interests us. Based on the HOG information we will extract orientation information only and categorise it (cluster) using the kmeans algorithm to create a bag of visual words.

So, using our feature extraction function we will extract the HOG information for training our BoVW class. Below is the pseudo-code I used, but you can use whatever you want, ass long as at the end you have a matrix with HOG features extracted from the images (D,orientations) and a label vector that corresponds to the feature matrix (D,)

```
def extract_train_hog_feature_vector(files, labels, orientations,
    pixels per cell, cells per block)
  for f, l in zip(files, labels)
    extract hog features
    store hog features
    store labels for each hog feature
  return the hog feature matrix and the labels vector
```

Next, we need to code up the BoVW class, once again you can do this

however you like but below is my pseudo-code:

```
class BoVW
  def __init__(nclusters)
    store nclusters as a member
  def fit(X)
    use X and KMeans to fit your data to the nclusters member
  def predict(X)
    create the feature vector from KMeans predict
    create a histogram with your feature vector and nclusters as
    the bins
```

Now you will need to load all your data for training your BoVW object, and fit it. Finally, go through your evaluation files and plot the histograms of each type of data and see if you can visually see the difference between the classes of texture you select. I recommend starting with two that are very far apart like vertical and horizontal stripes.

3 K-Nearest Neighbour

For this exercise we will use the colour snippets dataset, you will need to include in your parsing arguments the location of the dataset, and the colours you wish to use in the dataset. For selecting the colours you can add the following argument:

```
parser.add_argument( '--classes', '-c', nargs="+", default=[] )
```

In this example your command line would look like *python <your file> –exr <your exercise> -c black white*. It will append these two colours to the flag classes for you to use. The pseudo-code for loading the data is as follows:

```
For the colours you selected get the filenames.
  for each filename load the RGB image and divide it by 255
    normalise the RGB image using the mean and standard deviation,
    both set to 0.5
    reshape the RGB image into a (N,3) matrix.
    create a label vector that stores the label for each pixel in
    your (N,3) matrix, such that you get a label vector (N,)

Use train_test_split with a test value of 0.3 to split your data up
     you'll need both the samples and the labels from this.
```

The next step is to code up your KNN class, which you will put in your library file. The following is the pseudo-code for the KNN class.

```
class KNN():
  def __init__(K)
    store K as a member
  def fit(X,y):
    using the RGB reshaped data as X and the labels as y
    store each as a member
  def euclid(x):
    calculate distance between your X data and your sample x
  def predict(D):
    create a vector of zeros called cls of shape (D.shape[0],)
```

```
    for all samples in D
      calculate the euclidean distance between and your member X
      use argsort on your distance vector to obtain the top K
    indexes in your distance array.
      get the member y values using those indexes.
      use from scipy.stats import mode to calculate the mode of
    this
      store the mode in your cls vector
    return cls
```

Now you can create your object based on your KNN class. You can play around with the K value to see what gives you good results. Once you have predicted using your evaluation set you can print out the confusion matrix. Try it with different colours to see what values you get. Why didn't we use the accuracy score here? Could we use the precision recall score on this?