Results from

Feature Clustering

Jamal (Jamalsweden@yahoo.com)

Introduction

This exercise is focusing on feature clustering, which in this case is a method to use PCA (principal component analysis) to recognize and handle the characteristics of patterns and textures in an image.

Thoery:

If we generate a set of Gabor-filters and convolve each of these filters with a copy of an image, then the convolved images will hold different responses for each filter. If the source image contains different areas of textures, then each of these areas will most likely have different responses to each of the different Gabor filters. The characteristics of a pattern or a texture in the image could therefore be described by the way that each Gabor filter interacts with it. If we could identify a set of parameters that describe how the filters affect each texture, then we should be able to use this to identify the areas in the image where the different textures are.

In this exercise we use the eigenvectors from a correlation matrix for the filtered images, to identify the set of characteristics for each texture. By selecting only the most significant eigenvectors, we reduce the amount of data that needs to be processed.

Part 1: Feature extraction

- **what happens to the shape of the filters as you increase the frequency? (look both at the frequency and at the image domains)**

In the image domain, an increase in frequency will generate a more defined horizontal or vertical line through the spectrum. In the frequency domain, an increase of the frequency results in a slightly less defined pulse that is shifted in position.

- **what is the relation between the real and imaginary part of a filter?**

The real and imaginary parts of the filter make up the components of the complex values that the frequency spectrum contains.

- **what happens when the orientation is changed?**

If the orientation is changed, then the frequency response in the frequency spectrum is rotated around the center of the spectrum. In the image domain, this is visualized by the filter being rotated.

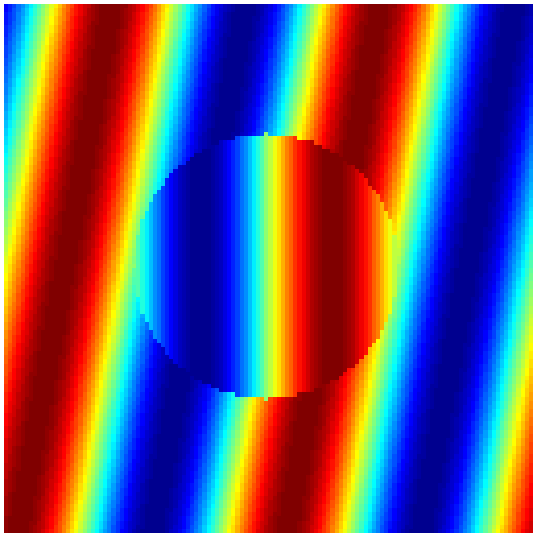

Image1: created with function makesinusoid(64,10,-10,128)

The Gabor decomposition can be used to obtain an invariant (constant) response to sinusoidal waves. Makesinusoid.m generates a sinusoidal wave with a rotated region in the middle. Try displaying one image (look at the comments).

Complete the code in invariance.m. This script will display the Gabor decomposition (have a look at decomposition.m) of the sinusoidal images considered.You have to code the line that forces the invariant response to sinusoids.

The left image below is created without having applied the "fix" to the file invariance.m, we can se how the filter passes through a sinus signal without modification, which is probably because the signal is a perfect match with the filter. This effect occurs because we only represent the real part of the result from our calculation.

After changing the line ***`gabor=real(gabor)`*** to ***`gabor=abs(gabor)`***, in the file invariance.m, we end up with the image to the right. Here we can now see that the real result from the filter operation generated a solid, more continuous response, to where the filter matched a signal in the image.

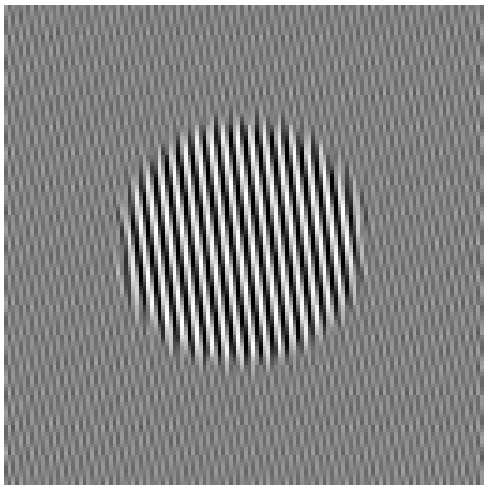

Image 2:

Image 3:

Conclusion:
The real part of the Gabor decomposition is NOT invariant. However, the absolute value of the Gabor decomposition is, and could be used to represent the area where the filter matches a signal.

Part 2: Clustering applied to Texture Segmentation

…

Do the same completion (invariant response to sinusoids) on texture3.m. Now go to pca3.m and complete the missing lines - refer to the theory, take your time and try to understand what is going on with pen and paper.

At first we add the same line as we added in the file invariance.m to the file texture3.m to get an invariant response when applying the filters.

```
gabor=abs(gabor);
```

In the file pca3.m we then add the following code:

```
f = corr(matrix_F')';
```

The function corr() extracts the correlation matrix. In our current code we do this instead of subtracting the mean, and multiplying the matrix by its transpose and a scalar.

We use the function eigs() to get the most significant eigenvectors/principal components and store their values to the *principal_componetns* matrix. The function eigs() returns a sorted matrix into *principal_components*, containing *nprcomps* rows with the most significant values.

```
[principal_components, pv]=eigs(f, nprcomps);
```

We then calculate the final result by multiplying the principal components by the matrix_F. The result is a matrix of size 256*256*nprcomps, which will be further processed.

Clustering

The Statistics Toolbox in Matlab implements functions pdist, linkage and cluster to perform clustering. Read the documentation for these functions.

Show with a simple computation that it is unfeasible to apply the clustering algorithm to all the feature vectors in our image directly.

The number of feature vectors in our entire image is equal to the number of pixels. The first step in our algorithm is to calculate the distance between every vector. Since this includes checking every vector with every other vector, the number of operations can be described by the formula for s_n. If we use all of our vectors in this step, the number of calculations will be too large to perform with an ordinary computer in a reasonable amount of time.

$$s_n = \sum_{k=1}^{n-1} k = n * \frac{(n-1)}{2} = \frac{n^2 - n}{2}, \qquad \text{n=256*256} \qquad s_n = 2.1475*10^9 \text{ calculations.}$$

What we do instead, is that we pick a number of randomly selected vectors from our feature vectors and perform the calculation on these.

Script clustering3.m guides you through these steps. Start by selecting a random subsample of 500 feature vectors. These are clustered by calling functions pdist, linkage and cluster. Select 'Euclidean' distance for pdist and 'average' distance for linkage. You can now visualize the cluster tree using function "dendrogram". Can you identify the depth at which there are exactly 7 clusters (which is the number of different regions in the image)?

```
%...
```

```
m=pdist(sample','euclidean');
```

```
%...
```

```
cluster_tree=linkage(m, 'average');
```

```
%...
```

Next, compute the average vector (the centroid) of each cluster. Scan all the feature vectors (= all the image pixels); for each vector, find the closest cluster centroid and assign the pixel to that region. Display the resulting segmented image.

```
average=zeros(nprcomps, nregions);

for cl=1:nregions
    indexes = find(clusters==cl);
    dim=size(indexes);
    avg = zeros(nprcomps, 1);
    for i=1:dim(1)
        avg = avg + sample(:, indexes(i));
    end
    if (dim(1) > 0)
        avg=avg./dim(1);
    end
    average(:, cl)=avg;
end
```

Once we have the average for each cluster, we compute its distance to the feature vector for each pixel in the image. The cluster with the smallest distance to a pixels feature vector, is the cluster that we assign the pixel to. With each cluster given a different color, we can see the result of the texture recognition in image 5.

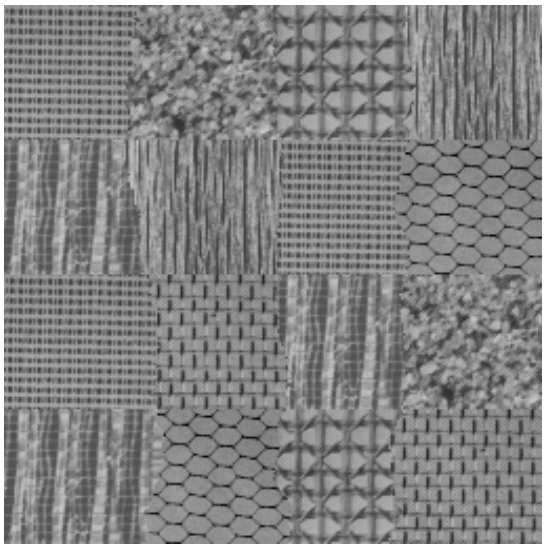

Image 4

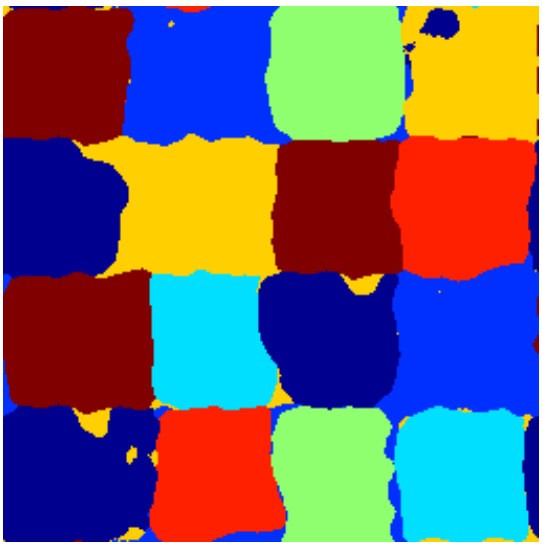

Image 5