Texture Analysis

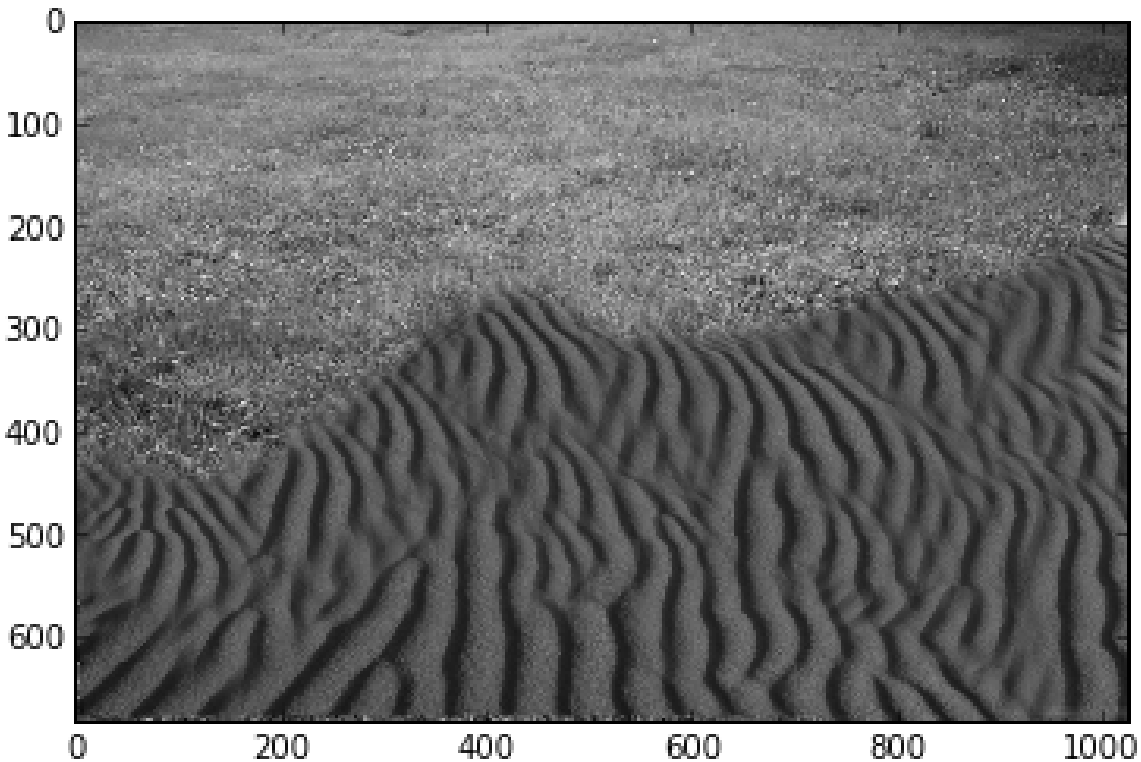
Thomas Breuel

UniKL

Texture Segmentation

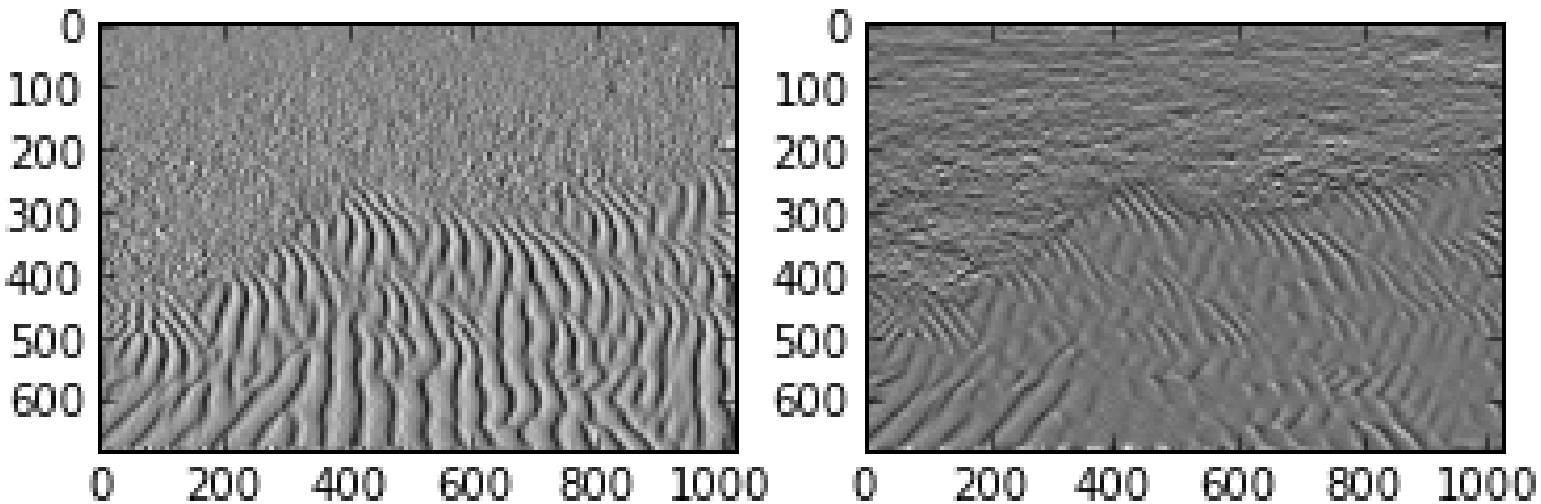
Images are often composed of multiple *textures*.

```
1 textures = mean(imread("textures.jpg"),axis=2)
2 gray(); imshow(textures); h,w = textures.shape
```



Textures can often be described well by the output from *filter banks*. Usually, we start with gradient filters at a particular *spatial scale*.

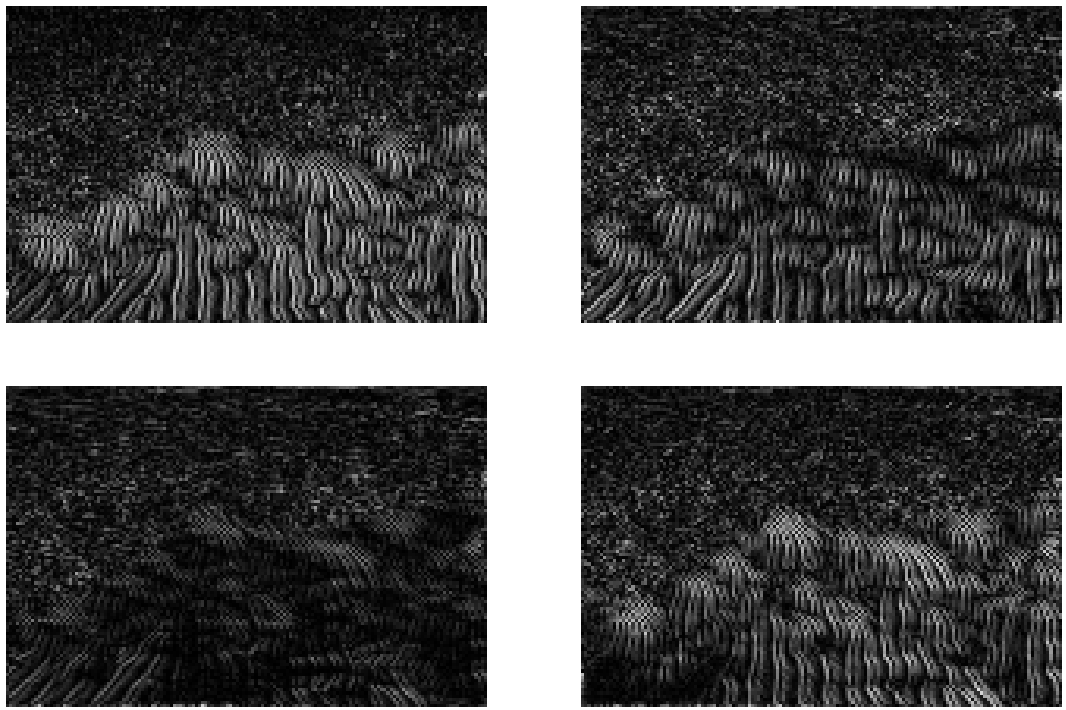
```
scale = 3.0
dy = filters.gaussian_filter(textures,scale,order=(1,0))
dx = filters.gaussian_filter(textures,scale,order=(0,1))
subplot(121); imshow(dx); subplot(122); imshow(dy)
```



We compute the response of filters at a particular scale at multiple orientations.

```
def orientation_channels(image,sigma,bins,spread=0):
    dy = filters.gaussian_filter(textures,sigma,order=(1,0))
    dx = filters.gaussian_filter(textures,sigma,order=(0,1))
    result = []
    for theta in linspace(0,pi,bins+1)[:-1]:
        delta = abs(cos(theta)*dx + sin(theta)*dy)
        if spread>0: delta = filters.maximum_filter(delta,spread)
        result.append(delta)
    return result
```

```
l = orientation_channels(textures,3.0,4)
for i in range(4): subplot(2,2,i+1); axis("off"); imshow(l[i])
```



Finally, a texture descriptor consists of the response of filters at multiple scales and orientations.

```
all_filters = []
for sigma,bins in [(3.0,4),(6.0,8)]:
    all_filters += orientation_channels(textures,sigma,bins,3*sigma
        )
for sigma in [1.0,3.0,6.0,9.0]:
    all_filters += [filters.maximum_filter(filters.gaussian_laplace
        (textures,sigma),3*sigma)]
k = len(all_filters)
```

Such texture filter banks give a vectorial response at each pixel in an image.

```
1 stack = transpose(array(all_filters),[1,2,0])
2 stack.shape
```

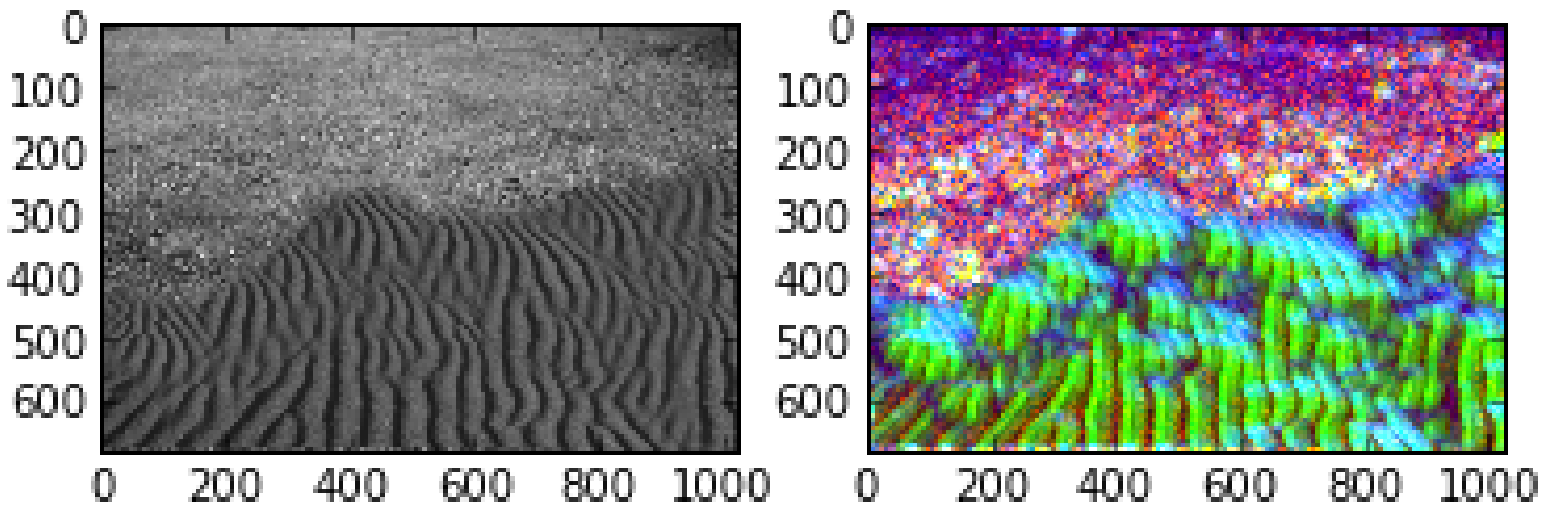
```
(682, 1024, 16)
```

Analogy to Color Channels

Texture filter bank responses are somewhat analogous to color channels. We can visualize this by performing PCA, reducing the 16-dimensional filter bank response to three color channels.

```
from sklearn.decomposition import PCA,RandomizedPCA
pca = RandomizedPCA(3,whiten=1)
k = stack.shape[-1]
pca.fit(stack.reshape(-1,k))
tex3 = pca.transform(stack.reshape(-1,k)).reshape(h,w,3)
ntex = clip(tex3/(4*mean(abs(tex3)))+0.5,0,1)
```

```
1 subplot(121); imshow(textures)
2 subplot(122); imshow(ntex)
```



Windowed Fourier Transform for Textures

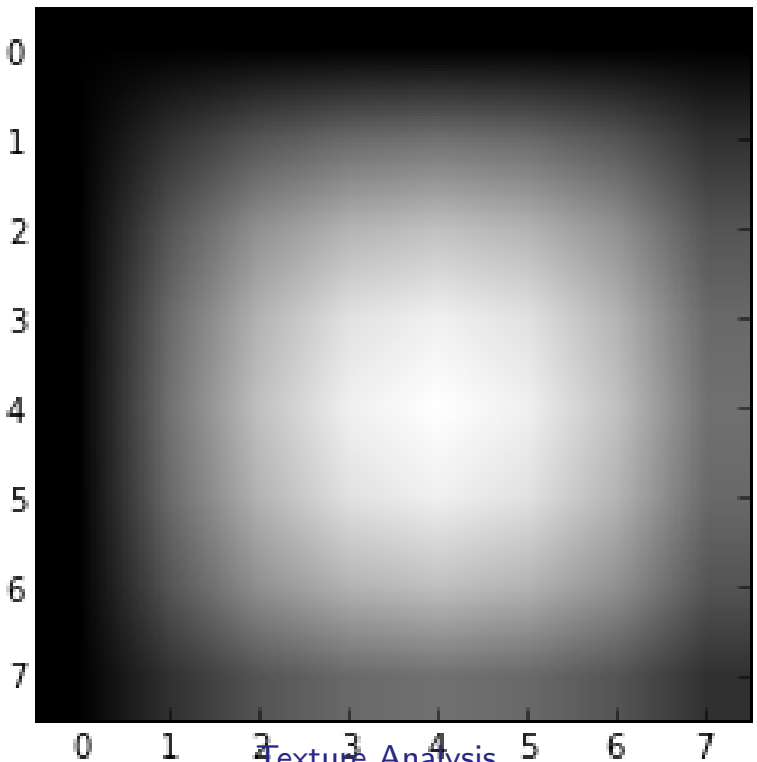
One problem with the filter bank approach above is that it results in spatially varying responses.

What we really want is something that characterizes the texture within a patch largely independent of phase.

We can accomplish this by (1) combining phase shifted versions of the filters above, or equivalently (2) performing local windowed Fourier transforms and computing the spectrum.

window function

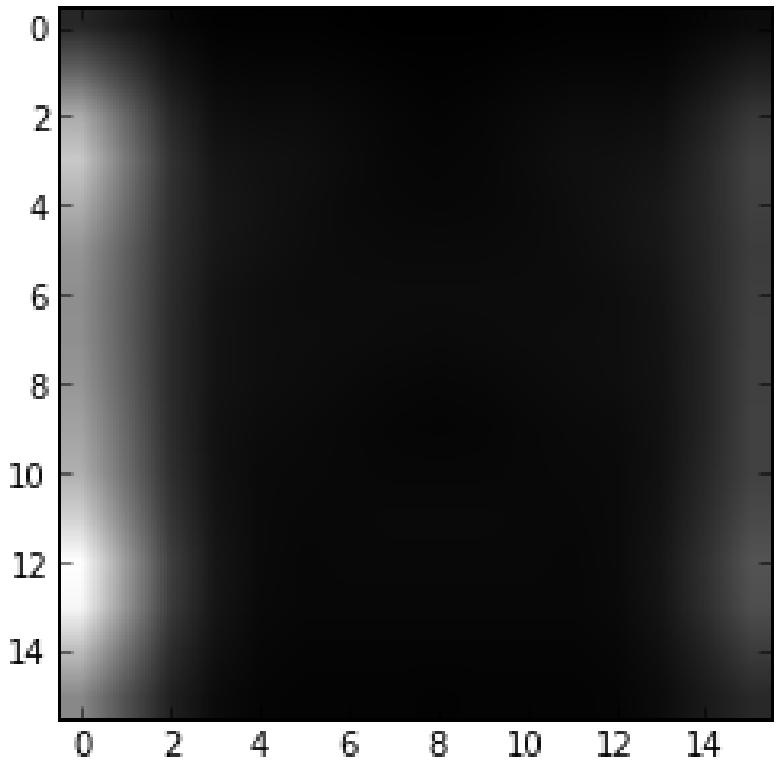
```
def window1(r):
    w = (r/2.0-arange(r))**2
    w = 1-w/amax(w)
    return w
def window(d0,d1):
    return outer(window1(d0),window1(d1))
imshow(window(8,8))
```



computing windowed spectra over local patches

```
def lspec(image,s=4,p=16,b=0):
    h,w = image.shape
    result = zeros((h//s,w//s,p,p))
    for i in range(0,h-p,s):
        for j in range(0,w-p,s):
            patch = image[i:i+p,j:j+p]
            patch -= mean(patch)
            sp = fft.fft(patch*window(*patch.shape))
            sp = abs(sp)
            if b>0: sp = filters.gaussian_filter(sp,b)
            result[i//s,j//s,:,:] = sp
    return result
result = lspec(textures,b=1)
```

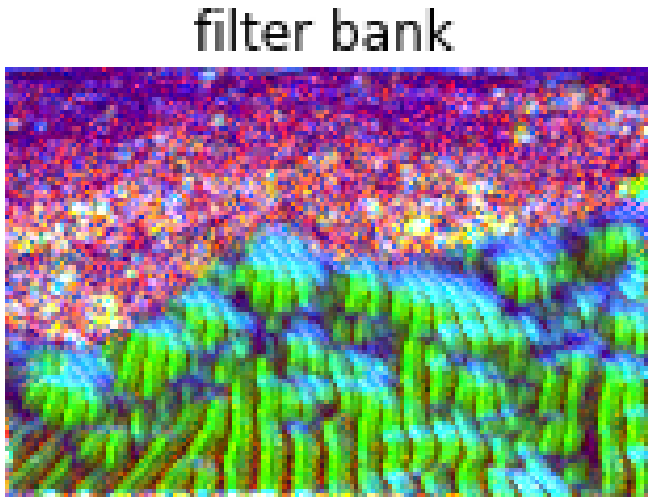
```
1 imshow(result[20,20])
```



```
k = prod(result.shape[-2:])
desc = result.reshape(-1,k)
dh,dw,_,_ = result.shape
pca = RandomizedPCA(3,whiten=1)
pca.fit(desc)
```

```
RandomizedPCA(copy=True, iterated_power=3, n_components=3, random_state=None,
       whiten=1)
```

```
1 out = pca.transform(desc)
2 out = out.reshape(dh,dw,3)
3 out = clip(out/3*mean(abs(out))+0.5,0,1)
4 subplot(121); axis("off"); imshow(out); title("windowed spectra")
5 subplot(122); axis("off"); imshow(ntex); title("filter bank")
```



Segmentation of Spatially Varying Textures

Texture segmentation requires spatial integration

- local information not sufficient for global segmentation in general
- a single texture may gradually vary across the image
- rotation, scale (due to perspective)
- analogous to motion integration