Amirkabir University of Technology
(Tehran Polytechnic)
Department of Computer Engineering

Computer Vision

Assignment III

Soroush Mahdi
99131050

supervised by
Dr. Reza Safabakhsh

December 5, 2021

Contents

1 Gabor filterbank

First i divided images to train and test sets.

```
#this function extract name of an image from its path
path_to_name = lambda path: path.split('.')[0].split('/')[1].lower()
#this function reads an image from its path and then
#change it to grayscale and resize it
preprocess = lambda path: cv2.resize(cv2.cvtColor(cv2.imread(path), \
     cv2.COLOR_BGR2GRAY), (200,150), interpolation=cv2.INTER_AREA)

random.seed(2)
categories = ['bricks', 'grass', 'gravel']
#this dict contains preprocessed images as values and their names as keys
images = {path_to_name(path):preprocess(path) for path in glob.glob("dataset/*")}
#this lists contains name of 2 photos from each category that are selected randomly
tests = [f'{c}_{i+1}' for c in categories for i in random.sample(range(7), 2)]
#this list contains name of photos that are not in the test set
trains = [name for name in images if name not in tests]
```

Listing 1: splitting images to test and train sets

and then I created gabor filters with different parameters and applied them on train photos one by one.

```
#this function create gabor filters based on input parameters
def makeGfilters(params):
    filters = []
    #for all combination of parameters
    for param in itertools.product(*params.values()):
        kernel = cv2.getGaborKernel(*param, ktype=cv2.CV_32F)
        filters.append({
            'param': param,
            'kernel': kernel,
            })
    return filters

#parameters for creating gabot filters
filterbank = makeGfilters({
    'ksize': [(3,3)  , (6,6)  ] ,# (6,6) ],
    'sigma': [2],
    'theta': [round(i*np.pi, 5) for i in [0,1/2 , 1/4 , 3/4]],
    'lambda': [3, 5 ],
    'gamma': [ 1,3],# , 2.0],
    'psi': [ .4 , .6 ]# , .8],
})
```

Listing 2: creating gabor filters and extracing features

i tried different values for parameters and we can see result of applying these kernels on train photos.for each kernel, the labels are showing value of parameters except sigma that is 2 for every kernel. caption of each image is showing the filter number and i will use these number to mention filters.

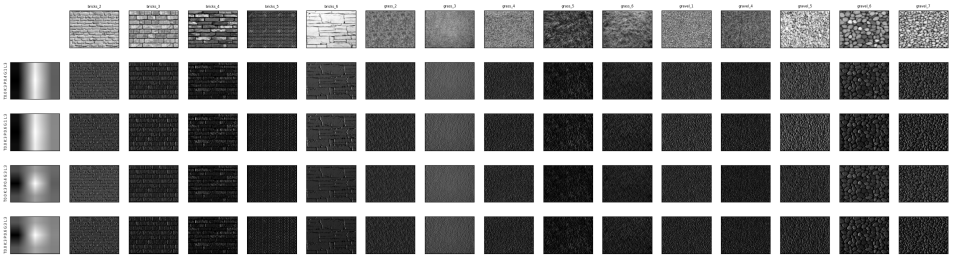

Figure 1: 0-3

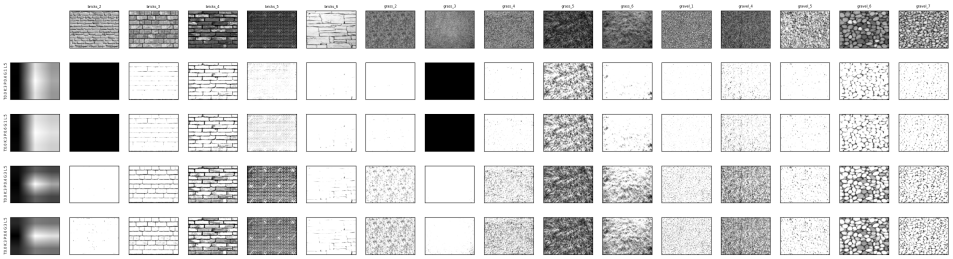

Figure 2: 4-7

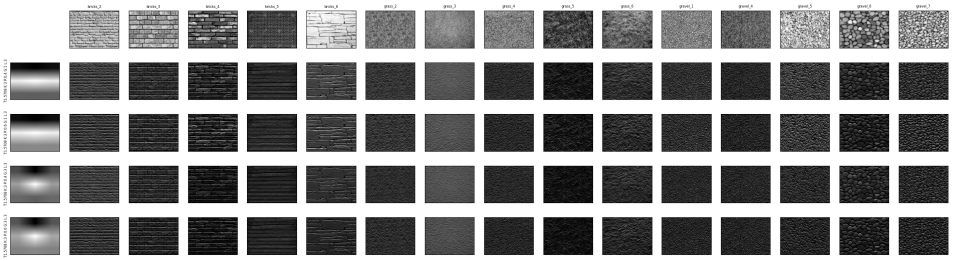

Figure 3: 8-11

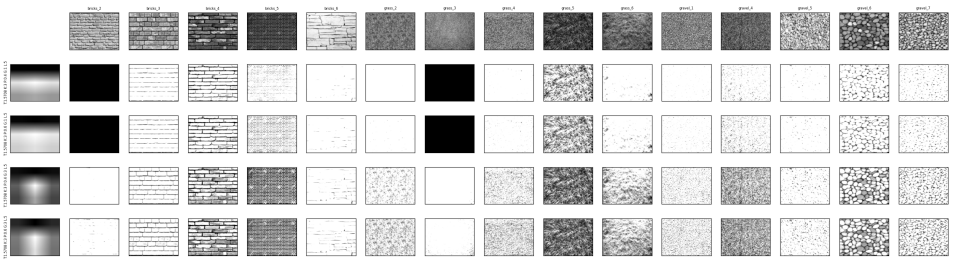

Figure 4: 12-15

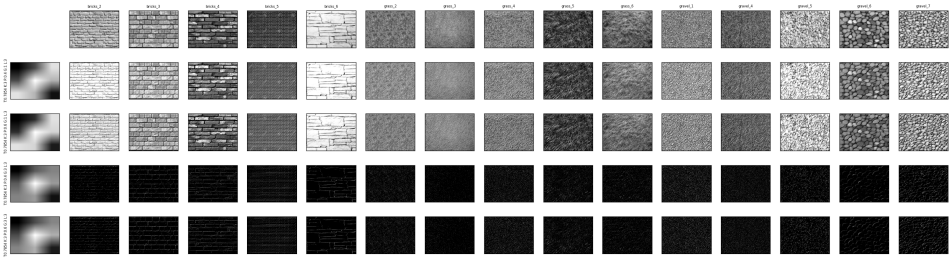

Figure 5: 16-19

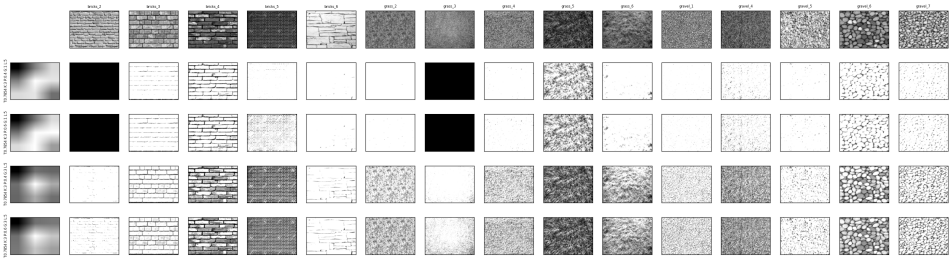

Figure 6: 20-23

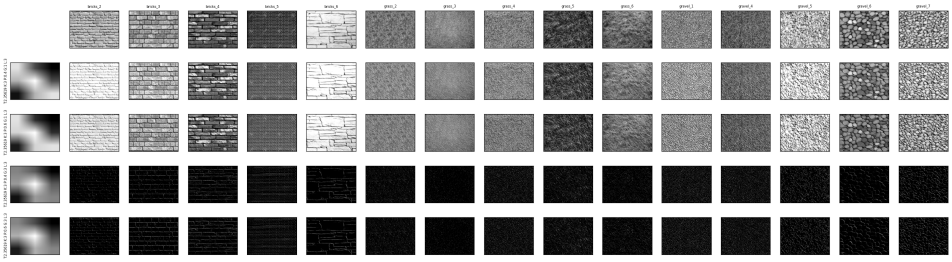

Figure 7: 24-27

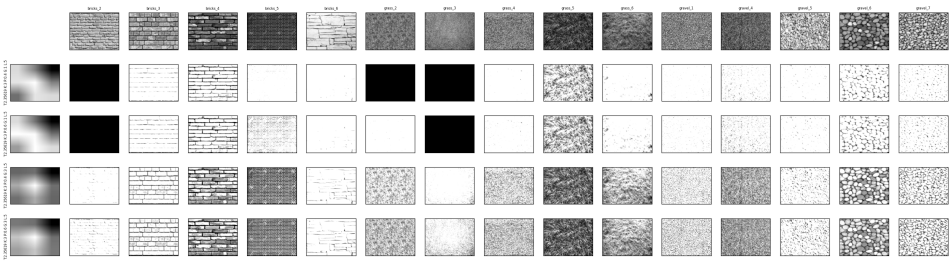

Figure 8: 28-31

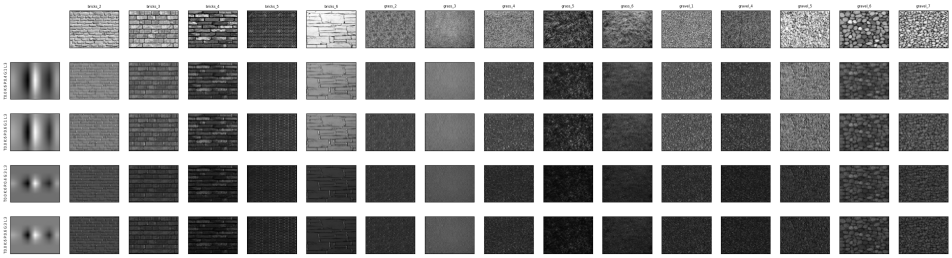

Figure 9: 32-35

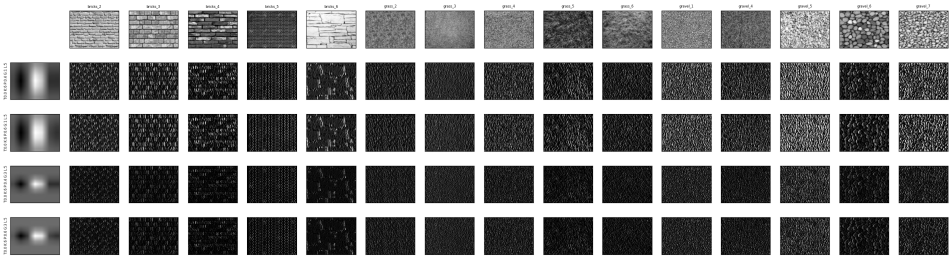

Figure 10: 36-39

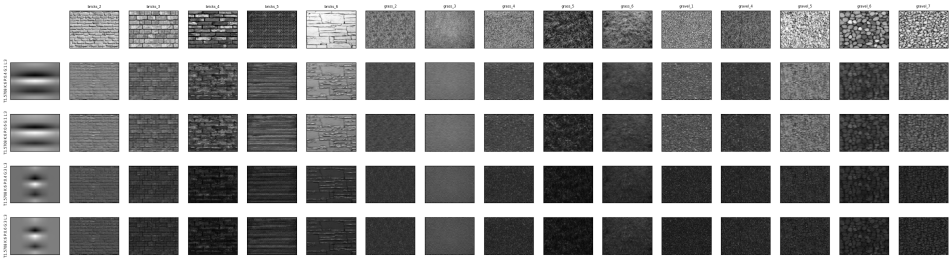

Figure 11: 40-43

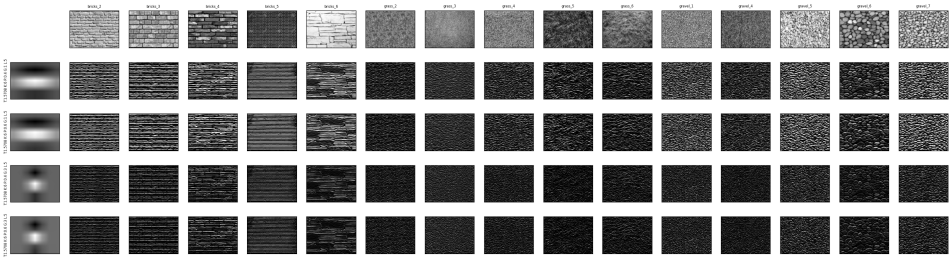

Figure 12: 44-47

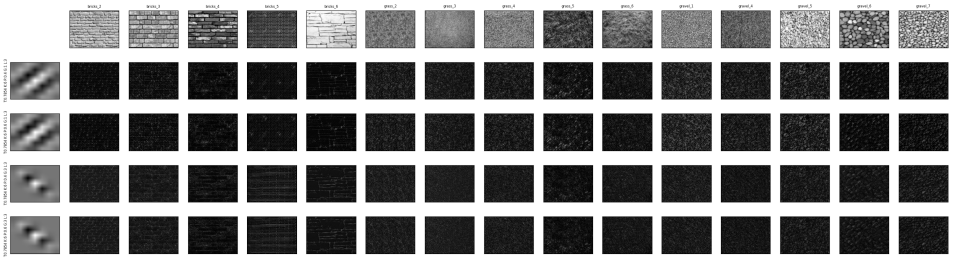

Figure 13: 48-51

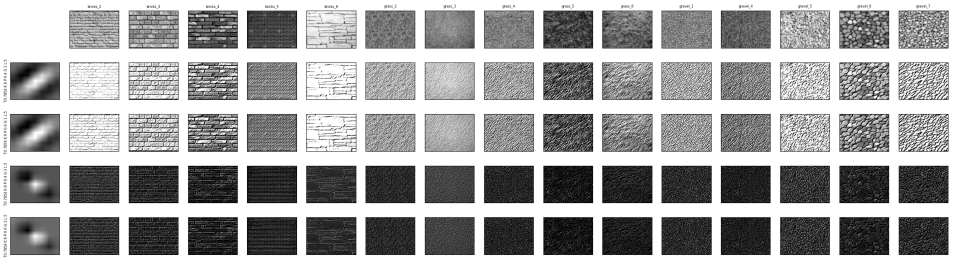

Figure 14: 52-55

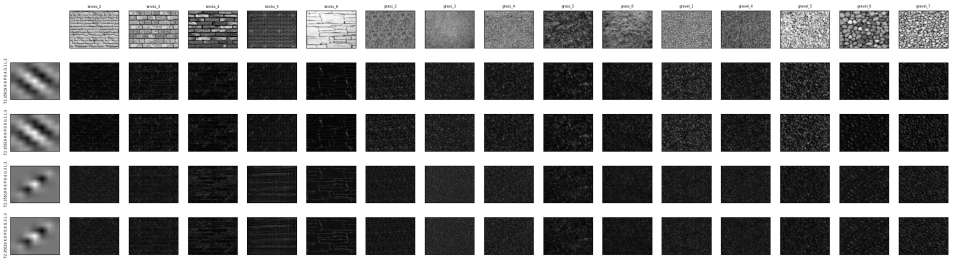

Figure 15: 56-59

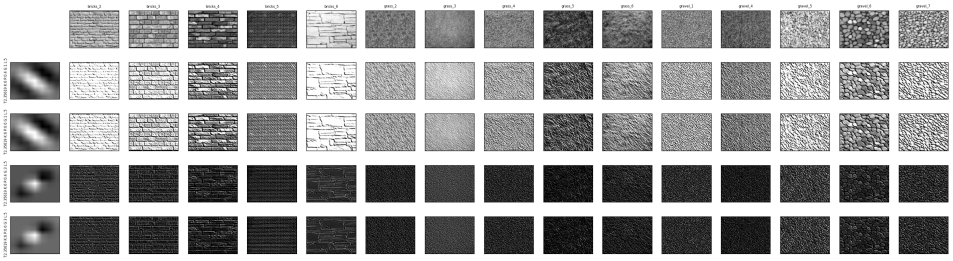

Figure 16: 60-63

i have selected these kernels for classification task:

- 6: it seems that this kernel can separate bricks from other classes and also can separate 2 of grass images from gravels.
- 17: it can separate bricks from other classes
- 40, 41: these kernels can separate 2 images of gravel class from other classes.
- 15: for 3 of grass images it can produce good features
- 16: this kernel can separate bricks too

2 Classification with Gabor filterbanks

Here is the code for classification

```
#find label based on mse errors on features
find_label = lambda test, trains, features: trains[np.argmin([mean_squared_error(
    features[test] , features[train]) for train in trains])][:-2]
#create feature planes on image with appliing kernels on them
get_features = lambda image , kernels: [cv2.filter2D(image, cv2.CV_8UC3, kernel) for
    kernel in kernels]

#selected gabor filters:
selcted_kernels =  [6,17,41 ,15,16,40]
_filterbank = [filterbank[i] for i in selcted_kernels]
#printing parameters of selected kernels
for i in selcted_kernels:
    print('parameters of gabor filter '+str(i))
    print(filterbank[i]['param'])
    print()

#extracting features
kernels = [filter['kernel'] for filter in _filterbank]
features = {name : get_features(images[name] , kernels)  for name in images}
#concating feature planes for each image
for name in features:
    feature = np.array([])
    for i in features[name]:
        feature = np.concatenate([feature , i.flatten()])
    features[name] = feature
#predicting labels using mse
predicted_labels = []
for label in tests:
    print(label)
    print(find_label(label , trains , features))
    print()
    predicted_labels.append(find_label(label , trains , features))
```

Listing 3: Classification

in this code i first extracted features from images using selected kernel and then used these feature planes for classification.

here are the results of applying selected filters on test set. and also we can see true label and predicted label for each image on top of the image.

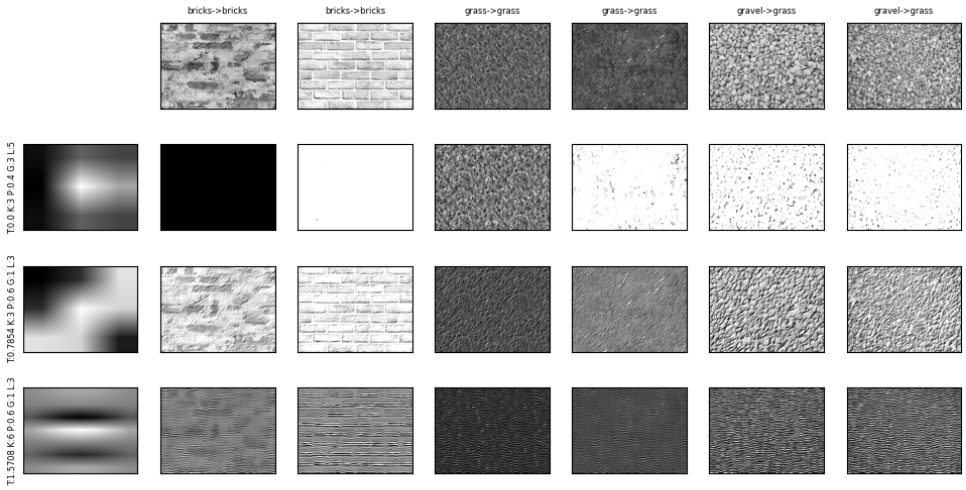

Figure 17: 6-17-41 kernels

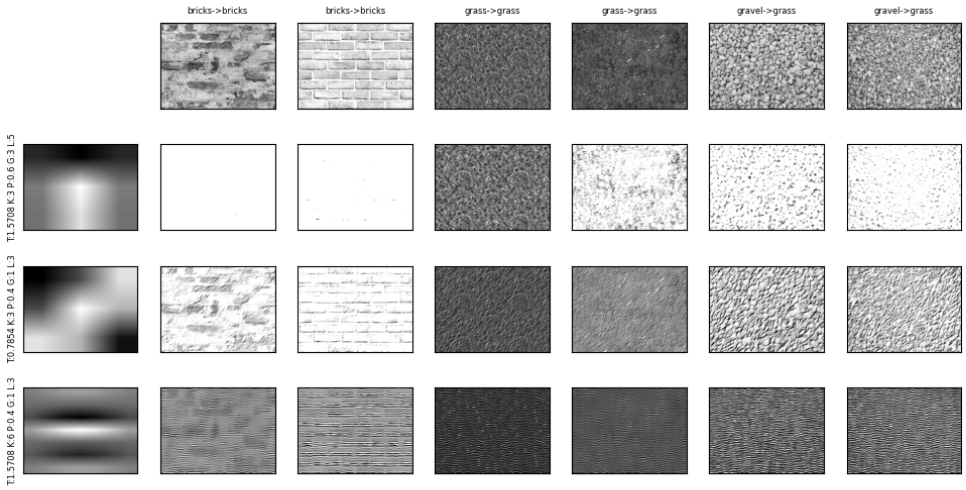

Figure 18: 15-16-40 kernels

3 Analyzing Gabor filters

We can see that our approach can't separate gravels from grass.

one of the problems in working with gabor filters are selecting parameters because we have to select them manually based on our problem.but this property help us to customize filters based on our problem. also, we can see that these filters can't separate gravel from grass.

and using these filters is easy and fast

4 LM, S, and MR filterbanks

here is the code for making LM filterbank:

```
#first derivatives of Gaussians
def gaussian1d(sigma, mean, x, ord):
    x = np.array(x)
    x_ = x - mean
    var = sigma**2
    g1 = (1/np.sqrt(2*np.pi*var))*(np.exp((-1*x_*x_)/(2*var)))
    g = [g1, -g1*((x_)/(var)), g1*(((x_*x_) - var)/(var**2))][ord]
    return g
#second derivatives of Gaussians
def gaussian2d(sup, scales):
    var = scales * scales
    shape = (sup,sup)
    n,m = [(i - 1)/2 for i in shape]
    x,y = np.ogrid[-m:m+1,-n:n+1]
    g = (1/np.sqrt(2*np.pi*var))*np.exp( -(x*x + y*y) / (2*var) )
    return g
#Laplacian of Gaussian
def log2d(sup, scales):
    var = scales * scales
    shape = (sup,sup)
    n,m = [(i - 1)/2 for i in shape]
    x,y = np.ogrid[-m:m+1,-n:n+1]
    g = (1/np.sqrt(2*np.pi*var))*np.exp( -(x*x + y*y) / (2*var) )
    h = g*((x*x + y*y) - var)/(var**2)
    return h

def makefilter(scale, phasex, phasey, pts, sup):
    gx = gaussian1d(3*scale, 0, pts[0,...], phasex)
    gy = gaussian1d(scale,   0, pts[1,...], phasey)
    image = np.reshape(gx*gy,(sup,sup))
    return image

def makeLMfilters():
    sup     = 49
    scalex  = np.sqrt(2) * np.array([1,2,3])
    norient = 6
    nrotinv = 12

    nbar  = len(scalex)*norient
    nedge = len(scalex)*norient
    nf    = nbar+nedge+nrotinv
    hsup  = (sup - 1)/2

    x = [np.arange(-hsup,hsup+1)]
    y = [np.arange(-hsup,hsup+1)]

    [x,y] = np.meshgrid(x,y)

    orgpts = [x.flatten(), y.flatten()]
    orgpts = np.array(orgpts)

    edge, bar, spot = [], [], []
    for scale in range(len(scalex)):
        for orient in range(norient):
            angle = (np.pi * orient)/norient
            c = np.cos(angle)
            s = np.sin(angle)
```

```
            rotpts = [[c+0,-s+0],[s+0,c+0]]
            rotpts = np.array(rotpts)
            rotpts = np.dot(rotpts,orgpts)
            bar.append(makefilter(scalex[scale], 0, 1, rotpts, sup))
            edge.append(makefilter(scalex[scale], 0, 2, rotpts, sup))

    scales = np.sqrt(2) * np.array([1,2,3,4])

    for i in range(len(scales)):
        spot.append(gaussian2d(sup, scales[i]))

    for i in range(len(scales)):
        spot.append(log2d(sup, scales[i]))

    for i in range(len(scales)):
        spot.append(log2d(sup, 3*scales[i]))

    return edge, bar, spot

filterbanks = {}
filterbanks['LM'] = [{'kernel': filter} for filter in itertools.chain(*makeLMfilters
    ())]
```

Listing 4: making ML filterbank

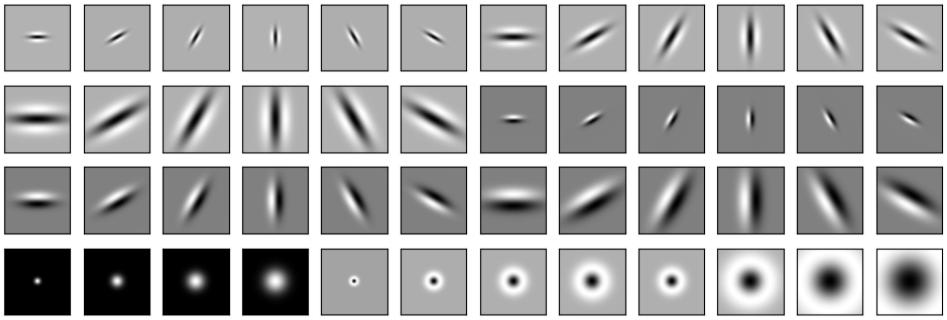

Figure 19: LM filterbank

code for MR filtebank:

```
def makeRFSfilters(radius=24, sigmas=[1, 2, 4], n_orientations=6):

    def make_gaussian_filter(x, sigma, order=0):
        if order > 2:
            raise ValueError("Only orders up to 2 are supported")
        # compute unnormalized Gaussian response
        response = np.exp(-x ** 2 / (2. * sigma ** 2))
        if order == 1:
            response = -response * x
        elif order == 2:
            response = response * (x ** 2 - sigma ** 2)
        # normalize
        response /= np.abs(response).sum()
        return response

    def makefilter(scale, phasey, pts, sup):
        gx = make_gaussian_filter(pts[0, :], sigma=3 * scale)
        gy = make_gaussian_filter(pts[1, :], sigma=scale, order=phasey)
        f = (gx * gy).reshape(sup, sup)
```

```
            # normalize
            f /= np.abs(f).sum()
            return f

        support = 2 * radius + 1
        x, y = np.mgrid[-radius:radius + 1, radius:-radius - 1:-1]
        orgpts = np.vstack([x.ravel(), y.ravel()])

        rot, edge, bar = [], [], []
        for sigma in sigmas:
            for orient in range(n_orientations):
                # Not 2pi as filters have symmetry
                angle = np.pi * orient / n_orientations
                c, s = np.cos(angle), np.sin(angle)
                rotpts = np.dot(np.array([[c, -s], [s, c]]), orgpts)
                edge.append(makefilter(sigma, 1, rotpts, support))
                bar.append(makefilter(sigma, 2, rotpts, support))
        length = np.sqrt(x ** 2 + y ** 2)
        rot.append(make_gaussian_filter(length, sigma=10))
        rot.append(make_gaussian_filter(length, sigma=10, order=2))

        # # reshape rot and edge
        # edge = np.asarray(edge)
        # edge = edge.reshape(len(sigmas), n_orientations, support, support)
        # bar = np.asarray(bar).reshape(edge.shape)
        # rot = np.asarray(rot)[:, np.newaxis, :, :]
        return edge, bar, rot

filterbanks['RFS'] = [{'kernel': filter} for filter in itertools.chain(*
    makeRFSfilters())]
```

Listing 5: making MR filterbank

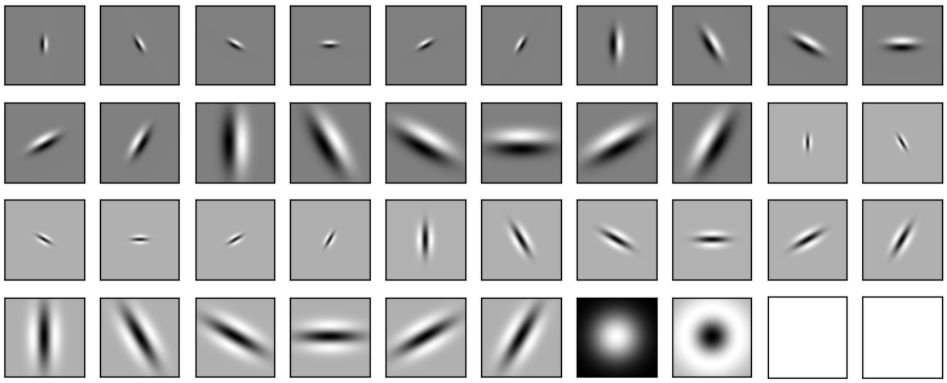

Figure 20: MR filterbank

code for S filtebank:

```
def makeSfilters():
    params = [(2,1),(4,1),(4,2),(6,1),(6,2),(6,3),(8,1),
              (8,2),(8,3),(10,1),(10,2),(10,3),(10,4)]
    filters = [makefilter(49, *param) for param in params]
    return filters

def makefilter(sup,sigma,tau):
    hsup  = (sup - 1)/2
    x = [np.arange(-hsup,hsup+1)]
    y = [np.arange(-hsup,hsup+1)]
    [x,y] = np.meshgrid(x,y)
    r=np.sqrt(x*x+y*y)
    f=np.cos(r*(np.pi*tau/sigma))*np.exp(-(r*r)/(2*sigma*sigma))
```

```
14      f=f-np.mean(f[:])
15      f=f/np.sum(np.abs(f[:]))
16      return f
17
18  filterbanks['S'] = [{'kernel': filter} for filter in makeSfilters()]
```

Listing 6: making S filterbank

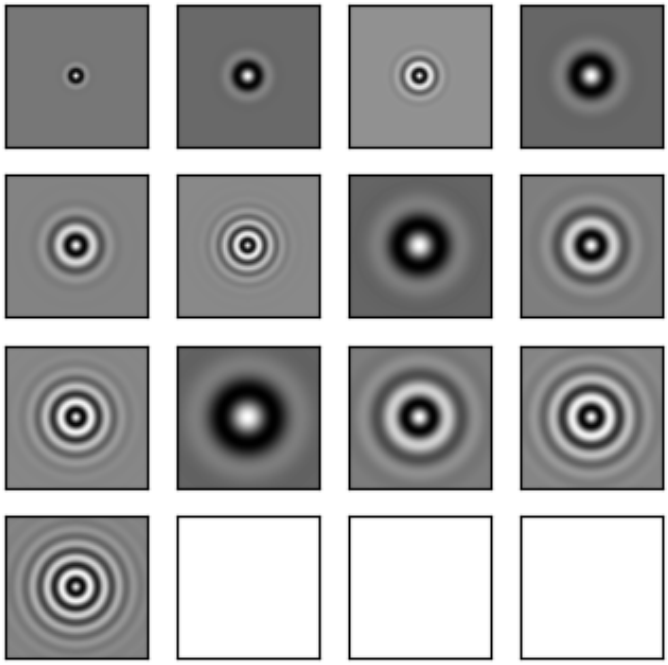

Figure 21: S filterbank

5 Clustring

function used for clustring:

```
def analyse(filters , k):
    results = []
    for name, kernels in filters.items():

        features = {name:np.array([cv2.filter2D(image, cv2.CV_8UC3, kernel) for
    kernel in kernels]).reshape((-1)) for name,image in images.items()}

        cluster_labels = KMeans(k).fit(list(features.values())).labels_
        cluster_purity = round(purity_score([name[:-2] for name in images], list(
    cluster_labels)), 2)

        results = sorted([*results, {
            'name': name,
            'kernels': kernels,
            'features': features,
            'cluster_labels': cluster_labels,
            'cluster_purity': cluster_purity,
            'title': param_to_str(['', 'cluster_purity='],
                                  [name, cluster_purity],'\n')
            }], key=lambda x: -x['cluster_purity'])

    return results
```

Listing 7: clustring

here are the result of clustering with different numbers of clusters:
clusters num: 2
MR cluster_purity=0.52
LM cluster_purity=0.48
S cluster_purity=0.38
clusters num: 3
LM cluster_purity=0.52
MR cluster_purity=0.52
S cluster_purity=0.43
clusters num: 4
MR cluster_purity=0.67
LM cluster_purity=0.57
S cluster_purity=0.48
clusters num: 5
MR cluster_purity=0.71
LM cluster_purity=0.62
S cluster_purity=0.52
clusters num: 6
LM cluster_purity=0.71
MR cluster_purity=0.71
S cluster_purity=0.57
best k for MR: 5,6
best k for LM: 6
best k for S: 6

6

based on experiments, seems that the best filterbank for clustering is MR filterbank and the second best is LM filterbank.

in Gabor filterbanks we had to pick parameters manually. so we can customize them based on our problem, and we can have any numbers of filters, but this feature has also the drawback that tuning these parameters may be hard.

overall it seems that for this problem gabor filters are better.