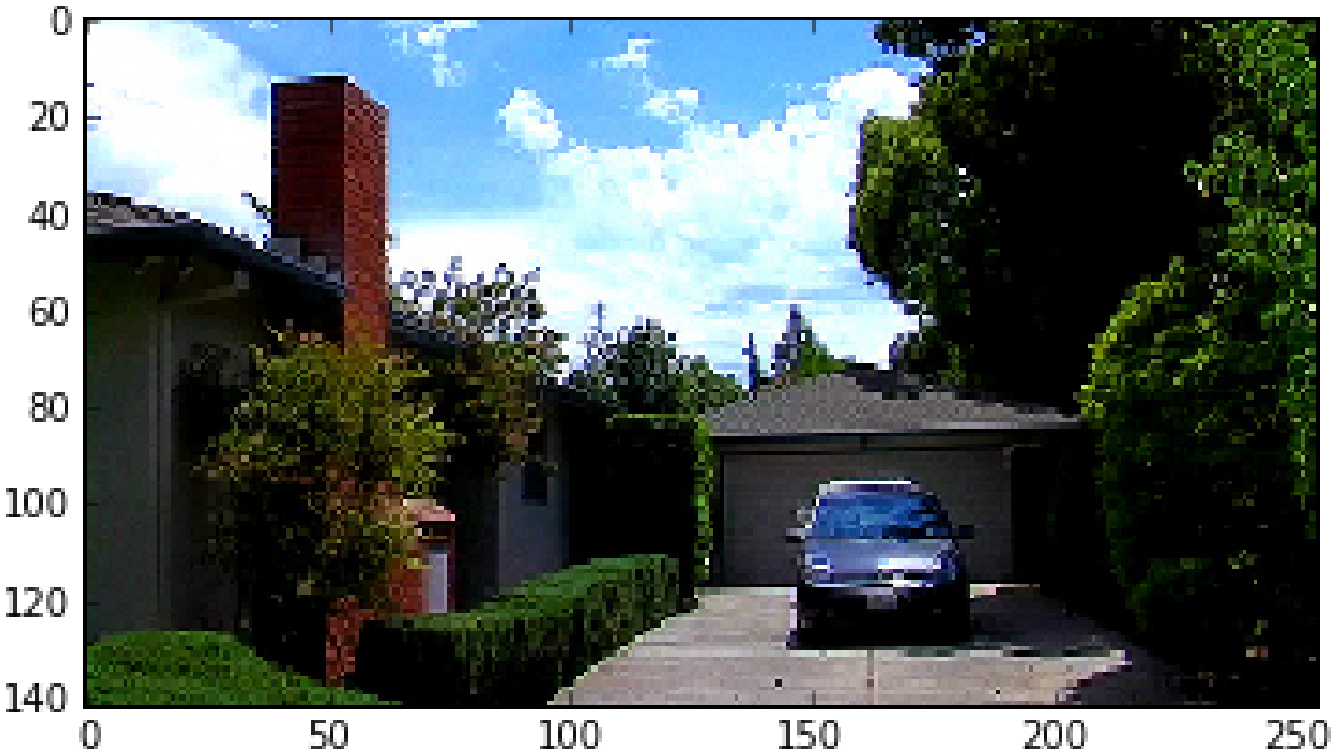
Interest Point Detection

sample image

```
1 test = crop_black(images[10])
2 imshow(test)
```

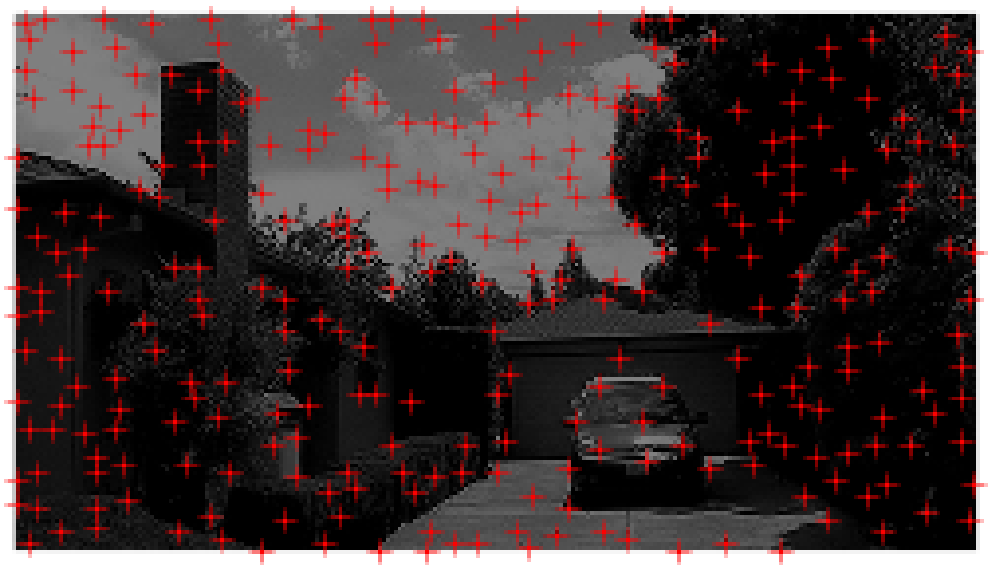


Harris corner detection with OpenCV

```
import cv2
from scipy.ndimage import filters
def harris_corners(image,r=8):
    # we do corner detection on grayscale images
    if image.ndim==3: image = mean(image,axis=2)
    # this implements the standard Harris corner detector
    result = cv2.cornerHarris(image,2,3,0.04)
    # avoid spurious flat regions
    result = result + 1e-6*randn(*result.shape)
    # smooth the results slightly
    result = filters.gaussian_filter(result,1.0)
    # this finds the locations of local maxima, separated by r
    locations = (result==filters.maximum_filter(result,r))
    # extract the locations and their strengths
    ys = find(locations)//locations.shape[1]
    xs = find(locations)%locations.shape[1]
    strengths = [result[i,j] for i,j in zip(ys,xs)]
    return c_[ys,xs],array(strengths)
```

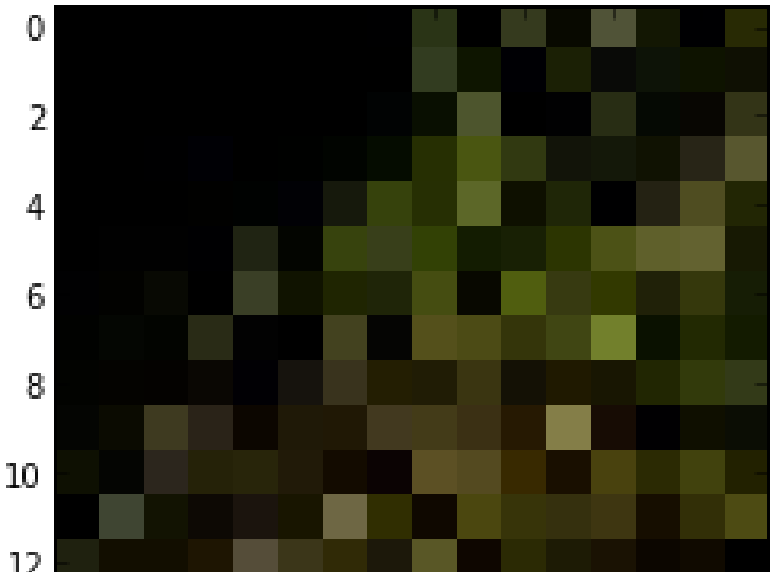
testing the corner detector

```
cs,_ = harris_corners(test)
axis("off")
imshow(mean(test,axis=2),vmax=2.0)
plot(cs[:,1],cs[:,0],'r+')
```



patch extraction using subscripting

```
1  def get_patch(image,i,j,r=16):
2      x,y = i-r//2,j-r//2
3      if image.ndim==2:
4          result = image[x:x+r,y:y+r]
5      elif image.ndim==3:
6          result = image[x:x+r,y:y+r]
7      assert result.shape[:2]==(r,r),(result.shape,(x,y),(i,j),r,
           image.shape)
8      return result
9
10 imshow(get_patch(test,90,30))
```

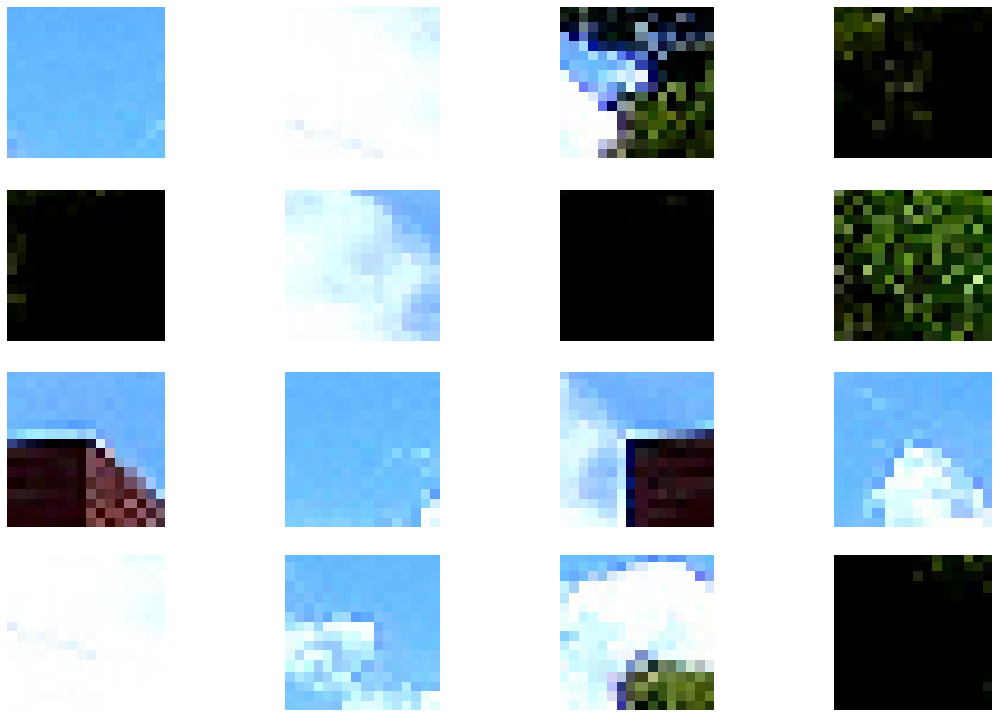


given an image, yield a sequence of patches, one at each detected interest point

```
def get_corner_patches(image,r=16):
    corners,_ = harris_corners(image,r//2)
    h,w = image.shape[:2]
    assert len(corners)<h*w*4.0/r*r
    for i,j in corners:
        if i<=r//2 or j<=r//2: continue
        if i-r//2>=h-r or j-r//2>=w-r: continue
        yield get_patch(image,i,j,r)
```

sample patches for the test image

```
for i,patch in enumerate(list(get_corner_patches(test))[:16]):
    subplot(4,4,i+1); axis("off"); imshow(patch)
```

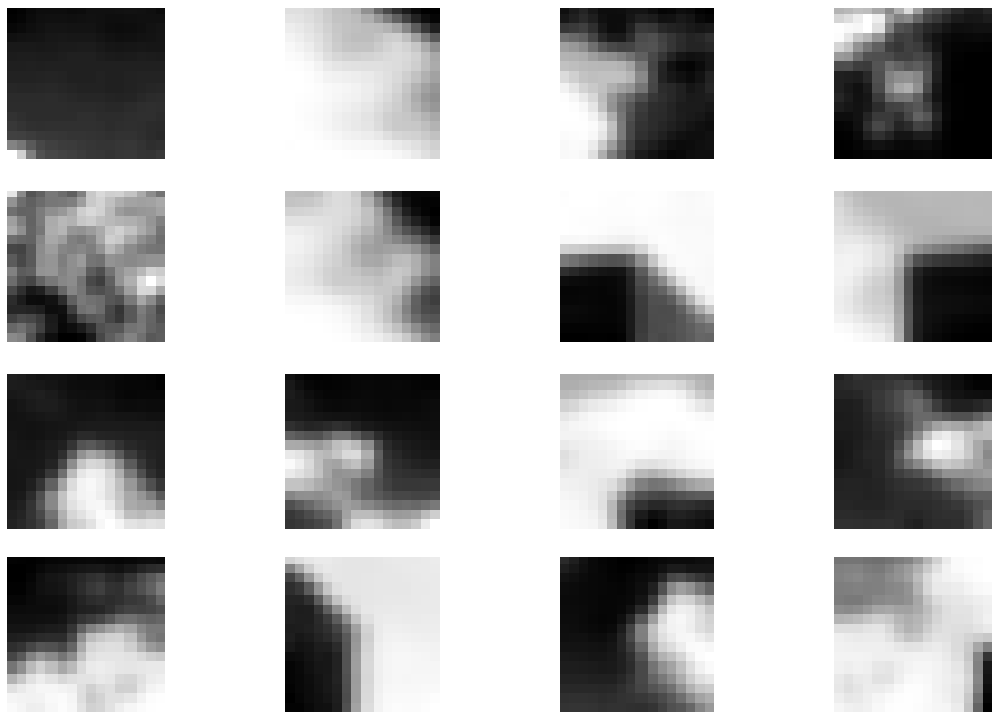


patches may need to get normalized before being used for indexing

```
1
2  def pnorm(patch):
3      """Normalize the patch for subsequent processing."""
4      if patch.ndim==3: patch = mean(patch,2)
5      patch = filters.gaussian_filter(patch,1.0)
6      return patch - mean(patch)
7
8  def get_normalized_corner_patches(image,r=16):
9      """Return a list of normalized corner patches for a given image
           ."""
10     for p in get_corner_patches(image,r=r):
11         result = pnorm(p)
12         if amax(result)-amin(result)<0.1: continue
13         yield result
```

let's check some sample patches

```
for i,patch in enumerate(list(get_normalized_corner_patches(test))
    [:16]):
    subplot(4,4,i+1); axis("off"); imshow(patch)
pshape = list(patch.shape)
```

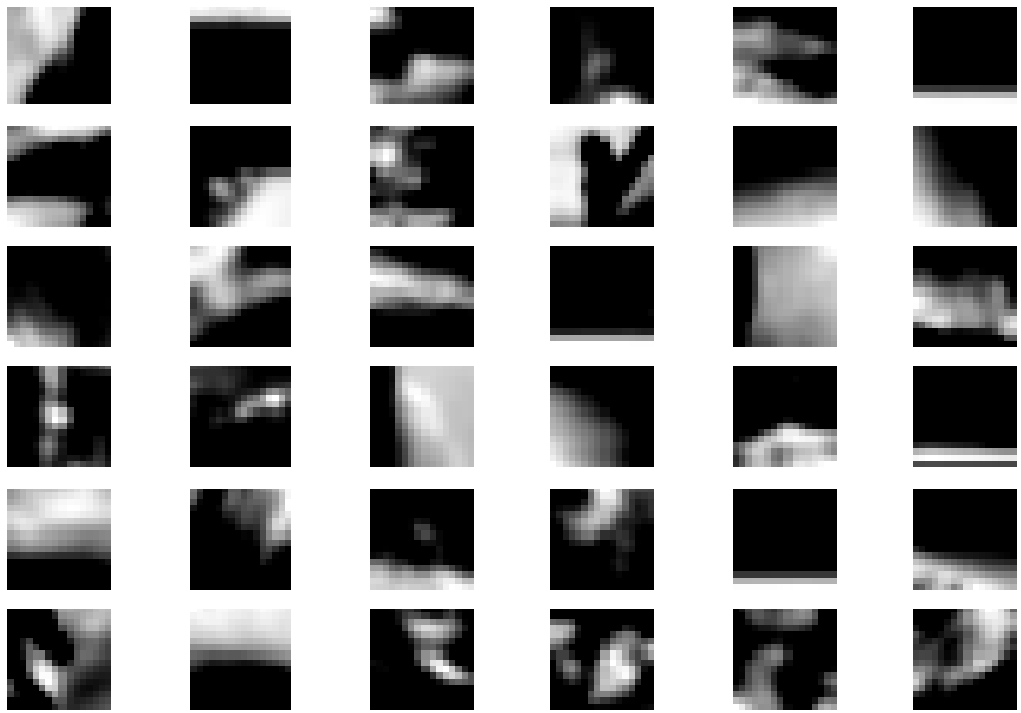


extract a sample of 100000 patches across the database

```
r = 16
data = zeros([100000]+pshape,'f')
i = 0
l = list(images)
shuffle(l)
for image in l:
    patches = list(get_normalized_corner_patches(image,r))
    for patch in patches:
        if i>=len(data): break
        data[i] = patch
        i += 1
    if i>=len(data): break
```

example patches displayed

```
for i,patch in enumerate(pyrand.sample(data,36)):
    subplot(6,6,i+1); axis("off"); imshow(clip(patch,0,1))
```



Compute Patch Dictionary

perform k-means clustering to create a shape inventory

```
K = 256
from sklearn.cluster import MiniBatchKMeans
km = MiniBatchKMeans(K)
km.fit(data.reshape(len(data),-1))
```

```
MiniBatchKMeans(batch_size=100, compute_labels=True, init='k-means++',
        init_size=None, k=None, max_iter=100, max_no_improvement=10,
        n_clusters=256, n_init=3, random_state=None,
        reassignment_ratio=0.01, tol=0.0, verbose=0)
```

```
print amin(km.cluster_centers_),amax(km.cluster_centers_)
for i,image in enumerate(km.cluster_centers_[:64]):
    subplot(8,8,i+1); axis("off"); imshow(image.reshape(*pshape)/
        max(amax(image),1.0))
```

```
-0.81164509058 0.807101726532
```

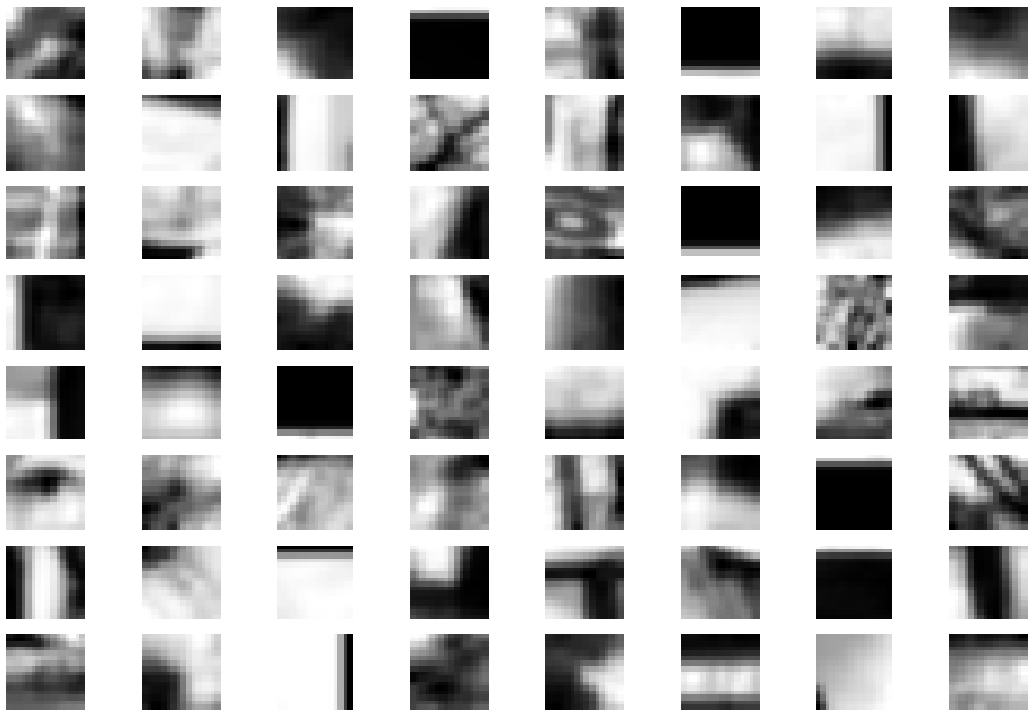


Image Description as Histogram of Patch Codes

a function to compute a histogram of the patches in an image

```
1 from collections import Counter
2 def patchhist(image):
3     data = array([patch for patch in get_normalized_corner_patches(
          image)])
4     codes = km.predict(data.reshape(len(data),-1))
5     result = zeros(K)
6     for k,v in Counter(codes).items(): result[k] = v
7     return result
```

```
array([  0.,   0.,   0.,   0.,   0.,   0.,   0.,   1.,   7.,   2.,   0.,
         0.,   0.,   0.,   0.,   0.,   2.,   0.,   4.,   0.,   1.,   0.,
         2.,   4.,   0.,   0.,   0.,   0.,   6.,   0.,   0.,   0.,   0.,
         2.,   3.,   2.,   4.,   0.,   4.,   0.,   1.,   2.,   0.,   3.,
         0.,   3.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   7.,
...
         3.,   1.,   1.,   4.,   3.,   4.,   0.,  13.,   0.,   0.,   2.,
         4.,   0.,   9.,   0.,   2.,   0.,   2.,   0.,   2.,   0.,   5.,
         0.,   0.,   1.,   5.,   1.,   5.,   0.,   3.,   8.,   3.,   3.,
         5.,   3.,  19.,   8.,   3.,   0.,   1.,   2.,   4.,   0.,  18.,
         0.,   1.,   0.])
```

an example patch histogram

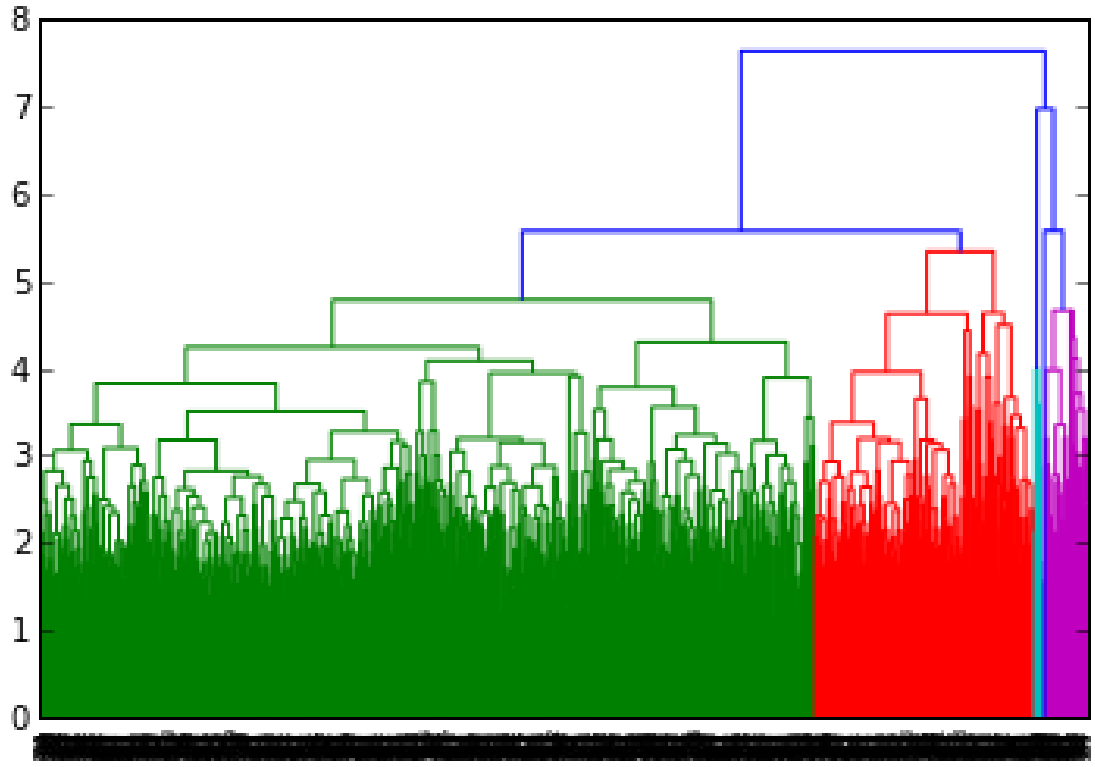
```
1 _=hist(patchhist(images[0]))
```

compute patch histograms for the entire database

```
1 phs = [patchhist(image) for image in images]
```

Hierarchical Clustering of Database

```
from scipy.spatial.distance import cdist
dists = cdist(phs,phs,metric='cosine')
from scipy.cluster.hierarchy import *
lm = linkage(dists,method='average')
_=dendrogram(lm)
```



```
clusters = fcluster(lm,300,criterion='maxclust')
Counter(clusters).most_common(10)
```

```
[(33, 24),
 (41, 21),
 (42, 15),
 (34, 14),
 (47, 14),
 (6, 12),
 (16, 12),
 (44, 12),
 (105, 12),
 (185, 12)]
```

```
for row,(cl,count) in enumerate(Counter(clusters).most_common(6)):
    l = [i for i,c in enumerate(clusters) if c==cl]
    shuffle(l)
    for j,im in enumerate(l[:6]):
        subplot(6,6,6*row+j+1); axis("off"); imshow(mean(images[im
            ],2))
```

Classification with Logistic Regression

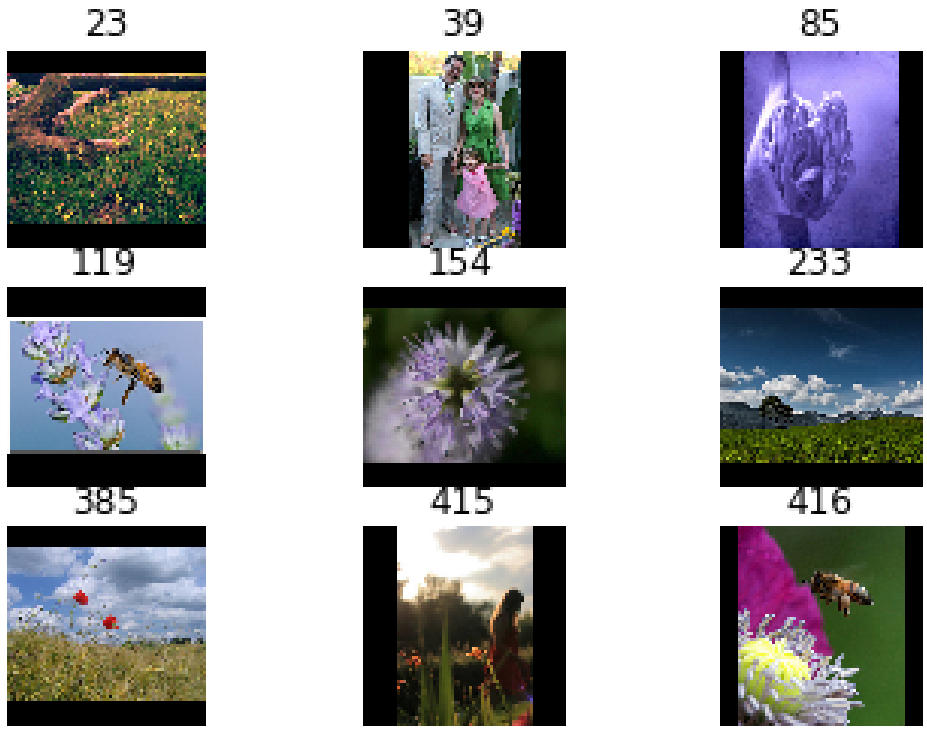
```
1 list(hdf.root.tagnames)
```

```
['flower', 'car', 'clouds', 'dog']
```

```
1 myclass = 0
2 targets = [myclass in hdf.root.tags[i] for i in range(len(hdf.root.
      tags))]
3 print sum(targets),len(targets)
```

```
264 1000
```

```
for i,c in enumerate(find(targets)[:9]):
    subplot(3,3,i+1); axis("off"); imshow(images[c]); title("%d"%
        hdf.root.ids[c])
```



fitting and testing the model

```
1 from sklearn.linear_model import LogisticRegression
2 lr = LogisticRegression()
3 lr.fit(phs,targets)
4 # for a serious evaluation, we should really use the cross
      validated error
5 pred = lr.predict(phs)
6 print sum(pred!=targets)*1.0/len(pred)
```

```
0.106
```

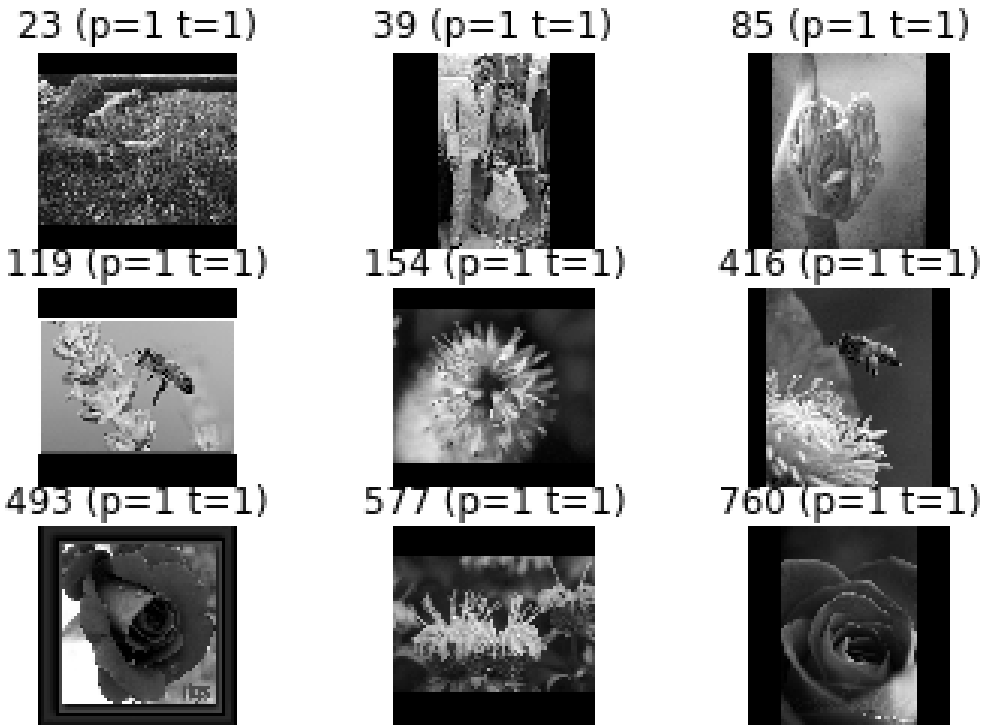
chance performance

```
1 junk = pred.copy()
2 shuffle(junk)
3 print sum(junk!=targets)*1.0/len(junk)
```

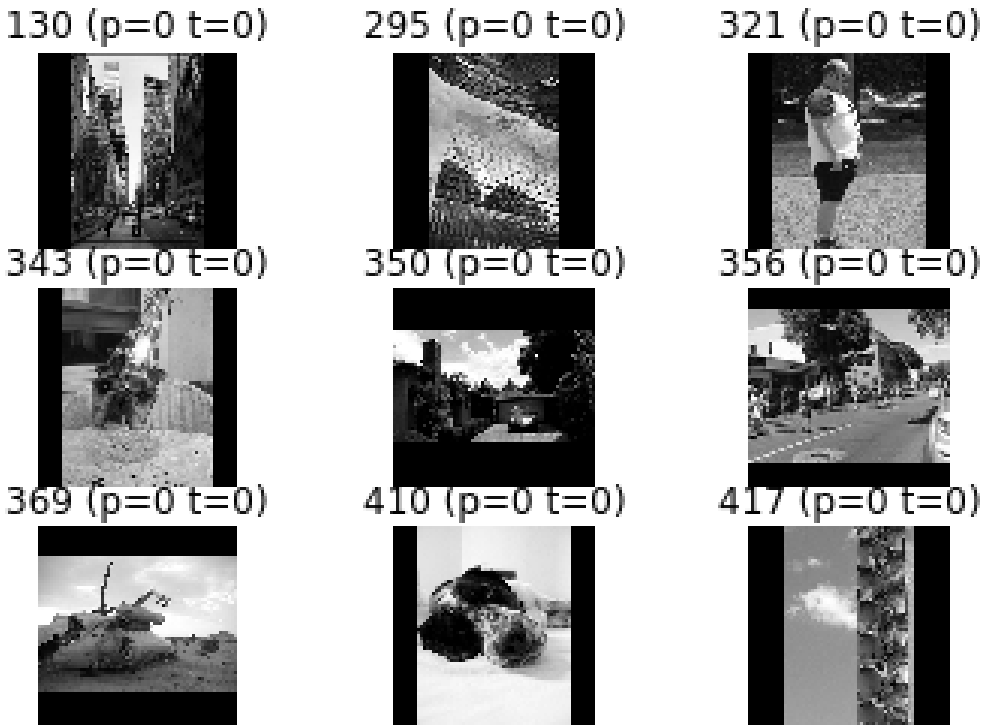
```
0.374
```

Error Analysis

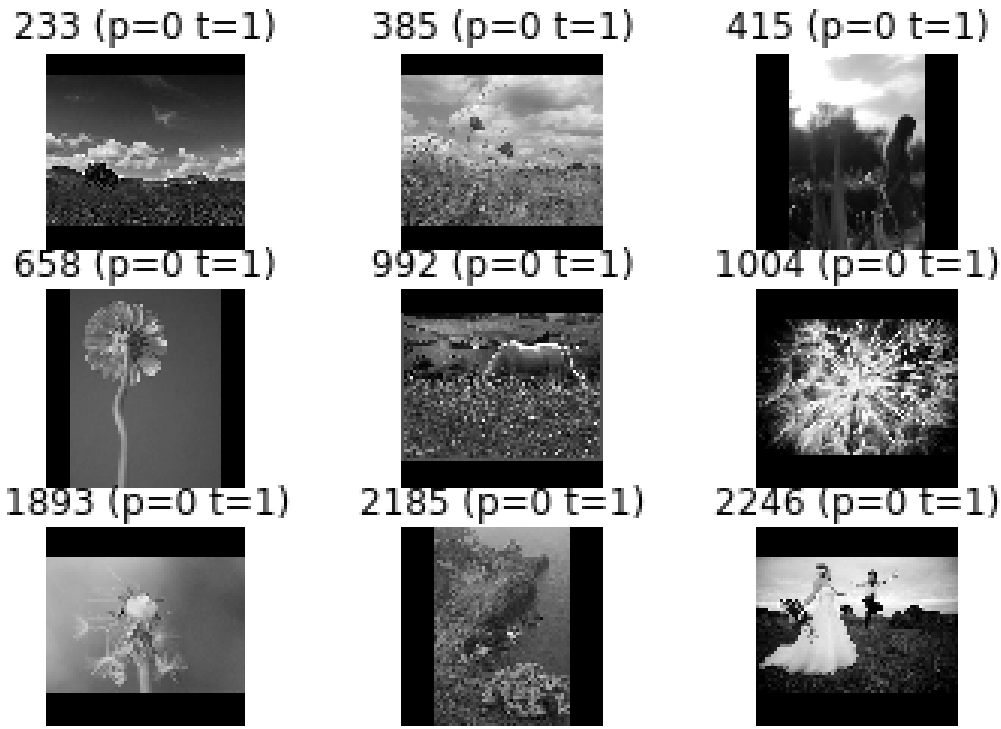
```
for i,c in enumerate(find(AND(pred,targets))[:9]):
    subplot(3,3,i+1); axis("off"); imgray(images[c])
    title("%d (p=%d t=%d)"%(hdf.root.ids[c],pred[c],targets[c]))
```



```
for i,c in enumerate(find(AND(NOT(pred),NOT(targets)))[:9]):
    subplot(3,3,i+1); axis("off"); imgray(images[c])
    title("%d (p=%d t=%d)"%(hdf.root.ids[c],pred[c],targets[c]))
```



```
for i,c in enumerate(find(AND(NOT(pred),targets))[:9]):
    subplot(3,3,i+1); axis("off"); imgray(images[c])
    title("%d (p=%d t=%d)"%(hdf.root.ids[c],pred[c],targets[c]))
```



```
for i,c in enumerate(find(AND(pred,NOT(targets)))[:9]):
    subplot(3,3,i+1); axis("off"); imgray(images[c])
    title("%d (p=%d t=%d)"%(hdf.root.ids[c],pred[c],targets[c]))
```