Texture Similarity

December 16, 2016

1 Texture similarity for Fashion Images using Gram matrix

Reference: Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In Advances in Neural Information Processing System, pages 262–270,201

Gram matrix summarizes the feature space provided by a high-performing deep neural network using only one spatial summary statistic: the correlations between feature responses in each layer of the network.

To characterise a given vectorised texture of an image x, x is through the convolutional neural network and compute the activations for each layer l in the network. Since each layer in the network can be understood as a non-linear filter bank, its activations in response to an image form a set of filtered images (so-called feature maps).

A layer with N^l distinct filters has N^l feature maps each of size M_l when vectorised. These feature maps can be stored in a matrix F^l, where F^l_{jk} is the activation of the jth filter at position k in layer l.

Textures are per definition stationary, so a texture model needs to be agnostic to spatial information. A summary statistic that discards the spatial information in the feature maps is given by the correlations between the responses of different features (gram matrix). A gram matrix at layer l is given by

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}.$$

The model is quite clever. The summary statistics from gram matrix seem to capture relevant information without storing large feature space.

The model used in this illutration is based on VGG-19 trained model in caffe. VGG-19 model consits of 16 convolutional layers and 3 fully connected layers (hence the name VGG 19). The list of all layers and the output blob dimnedtion at each layer is given below where the output of each blob is represented as a 4 dimentional tensor. (b,d,h,w) = (batch size, depth, hight, width)

data (1, 3, 256, 256) conv1_1 (1, 64, 256, 256) conv1_2 (1, 64, 256, 256) pool1 (1, 64, 128, 128) conv2_1 (1, 128, 128, 128) conv2_2 (1, 128, 128, 128) pool2 (1, 128, 64, 64) conv3_1 (1, 256, 64, 64) conv3_2 (1, 256, 64, 64) conv3_3 (1, 256, 64, 64) conv3_4 (1, 256, 64, 64) pool3 (1, 256, 32, 32) conv4_1 (1, 512, 32, 32) conv4_2 (1, 512, 32, 32) conv4_3 (1, 512, 32, 32) conv4_4 (1, 512, 32, 32) pool4 (1, 512, 16, 16) conv5_1 (1, 512, 16, 16) conv5_2 (1, 512, 16, 16) conv5_3 (1, 512, 16, 16) conv5_4 (1, 512, 16, 16) pool5 (1, 512, 8, 8)

The objective of this study is to define a Metric for texture similarity based on gram metrix definitiion for deep-convet. We define the distance between two textures at layer l as,

$$d_l(T_1, T_2) = \left| \frac{G^l(T_1)}{\|G^l(T_1)\|_F} - \frac{G^l(T_2)}{\|G^l(T_2)\|_F} \right|_F .$$

.

The distance metric can be extended over a set of layers:

$$d_{\mathcal{L}}(T_1, T_2) = \sum_{l \in L} \left| \frac{G^l(T_1)}{\|G^l(T_1)\|_F} - \frac{G^l(T_2)}{\|G^l(T_2)\|_F} \right|_F$$

.

A dictionary of a images is created. The key of the dictionary is the image location and its value is a gram matrix obtained at a layer l. T An imput image T_I is compared against the dictionary. The images

in the dictionary are ranked according the distance metric. First $K = 10$ images are presented as closest images w.r.t. the input image.

The texture has strong locality. It is expected that the texture information is stored in the initial layers of the covnet. Thus, this illutration focus on pool1 and pool2 layers. We study pool1, pool2 and agreemate distance metric of pool1 + pool2 for the study.

We also study cropped images to check if removing any background non-texture information can help. It is observed that simple cropping of the images by 80% of the input size does not work well comparaed to default method of using complete input image.

```
In [7]: # import the necessary packages
        from DeepDescriptor import DeepDescriptor
        from searcher import Searcher
        import argparse
        import cv2
        import os, random
        import matplotlib.pyplot as plt
        import math;

        %matplotlib inline
        # construct the argument parser and parse the arguments
        result_path = '../../fashion_train';
```

```
In [8]: nQuery_images = 10;
        Query_list = [];
        # load the query image and describe it
        for counter in xrange(nQuery_images):
          qfile = random.choice(os.listdir(result_path));
          query = (result_path + "/" + qfile);
          Query_list.append(query);
        print "Query images"
        plt.figure(figsize=(15,15));
        lensq = math.ceil(math.sqrt(nQuery_images));
        qidx = 1;
        for ilen in range(len(Query_list)):
          query = Query_list[ilen];
          img = cv2.imread(query);
         # print query;
          plt.subplot(lensq,lensq,qidx);
          plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB));
          qidx += 1;

        plt.show();
```

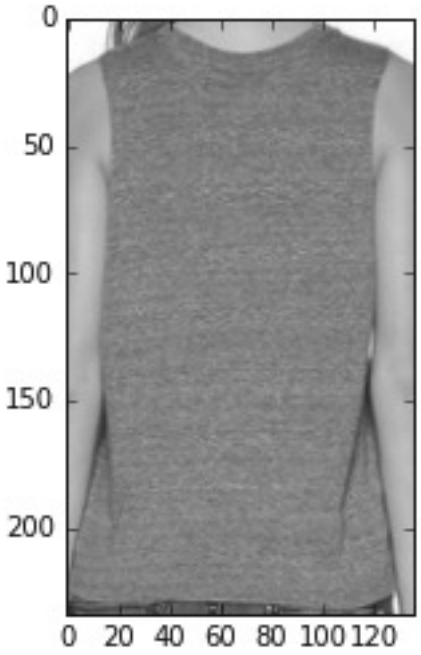
```
Query images
```

1.1 Using Pool 2 activations

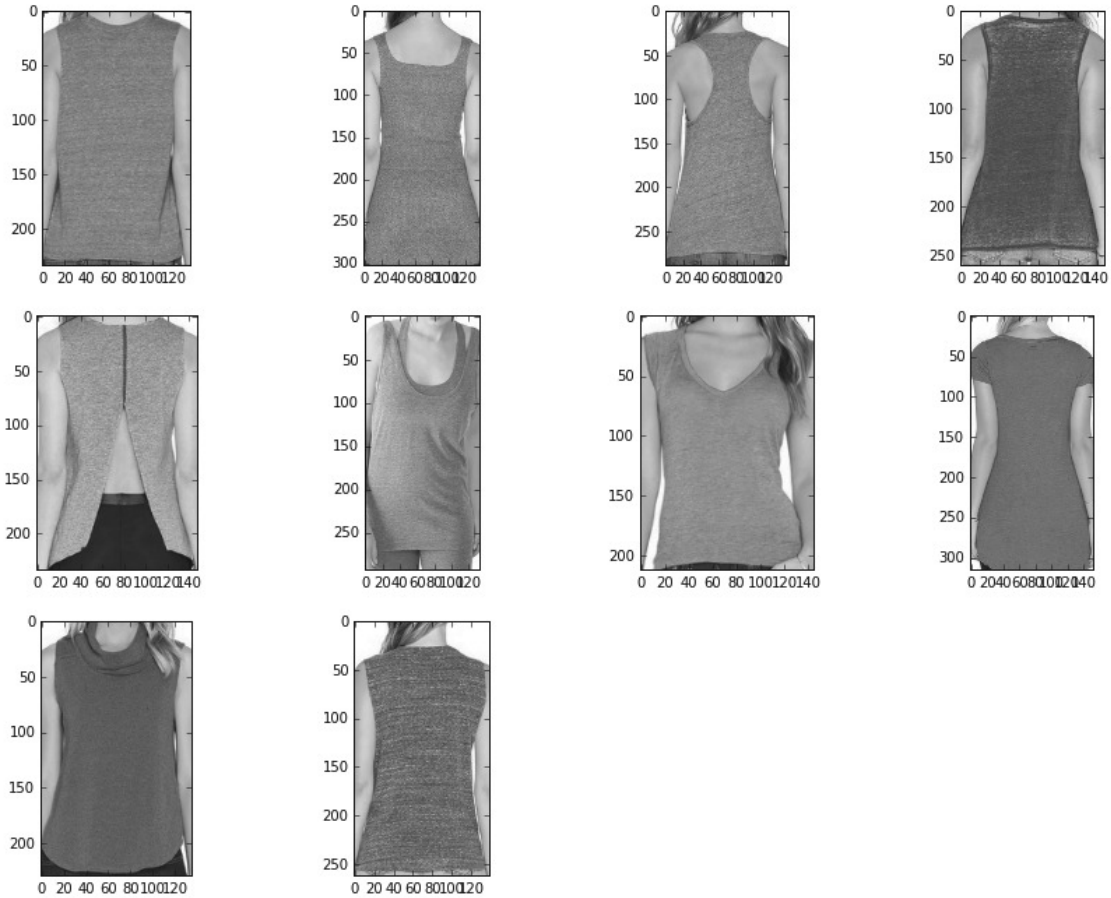
```
In [9]: index =  'index_pool_2.csv'
         # initialize the image descriptor
        cd = DeepDescriptor('pool2','VGG')

In [10]: for ilen in range(len(Query_list)):
           query = Query_list[ilen];
           img = cv2.imread(query);
           print "Query"
         # display the query
           plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB));
           plt.show();
           features = cd.describe(query);

         # perform the search
           searcher = Searcher(index,"pool2")
           results_pool_2 = searcher.search(features)
         #
           plt.figure(figsize=(15,15));
           lenf = len( results_pool_2);
           lensq = math.ceil(math.sqrt(lenf));
           resultidx = 1;
```

```
    #print  results_pool_2
    print "Similarity"
  # loop over the results
    for (score, resultID) in results_pool_2:
          # load the result image and display it
      result = cv2.imread(result_path + "/" + resultID);
      plt.subplot(lensq,lensq,resultidx);
      plt.imshow(cv2.cvtColor(result,cv2.COLOR_BGR2RGB));
      resultidx += 1;
    plt.show();
  #        cv2.waitKey(0)
```

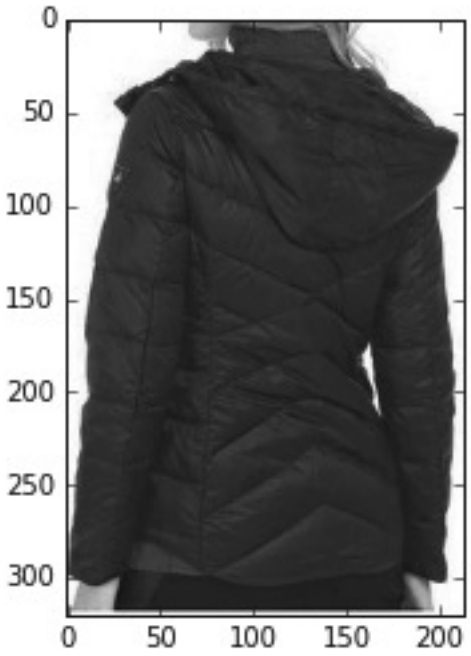
Query



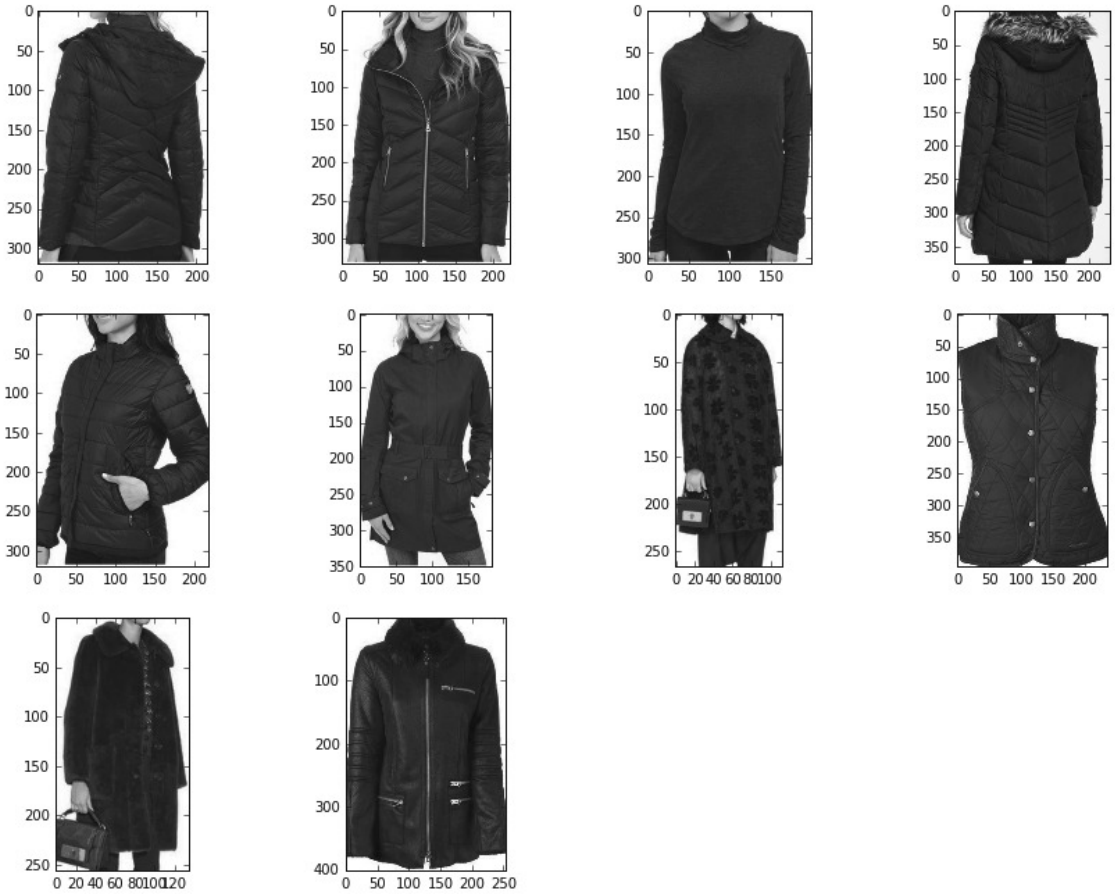
Similarity



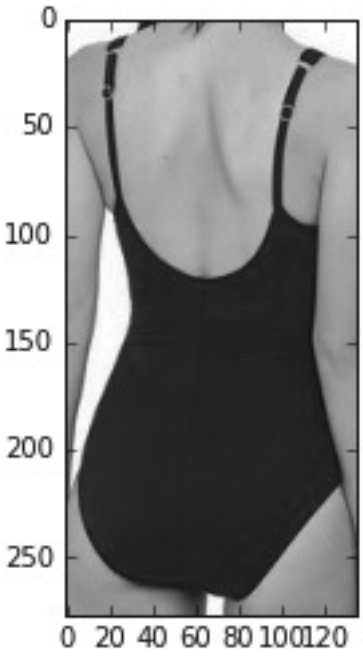
Query



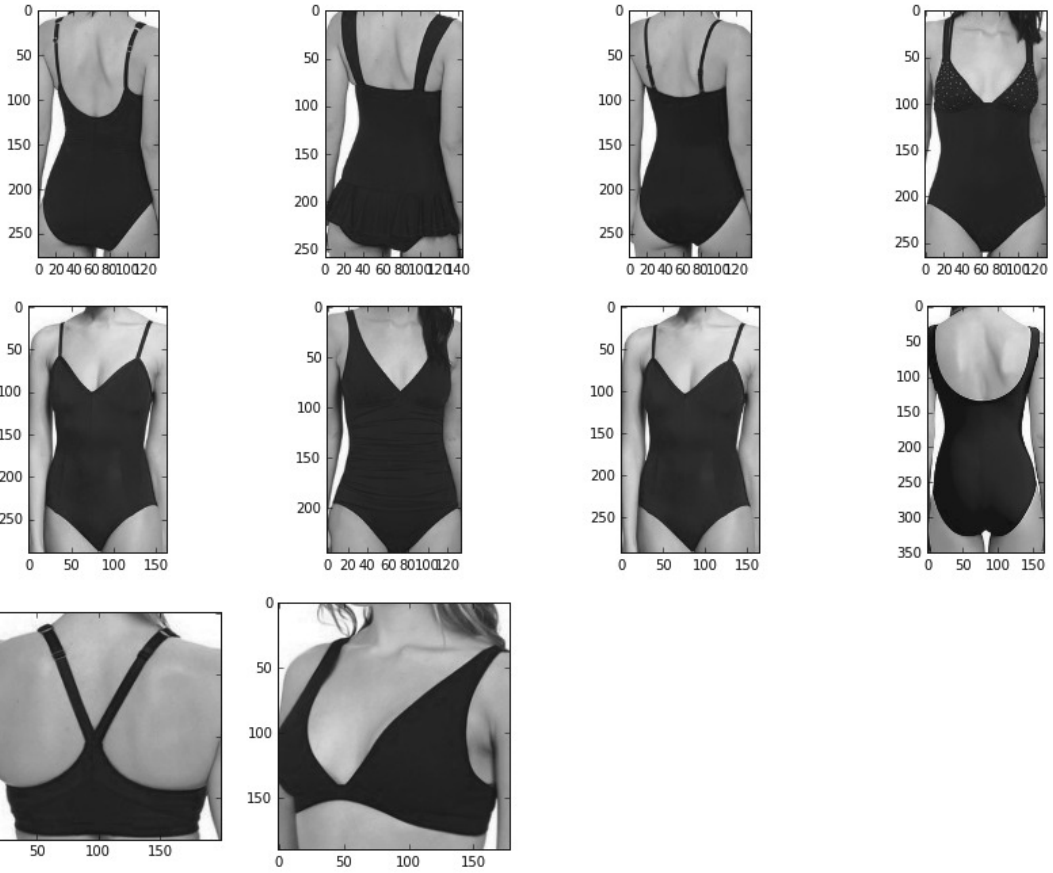
Similarity



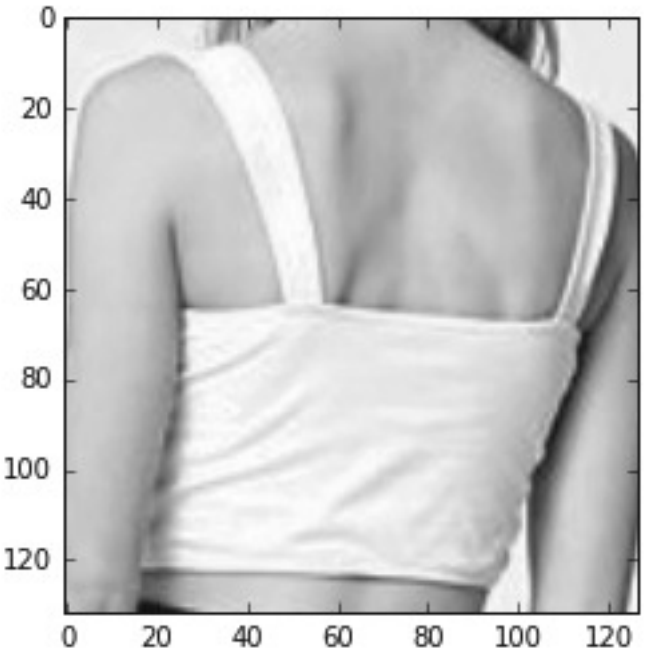
Query



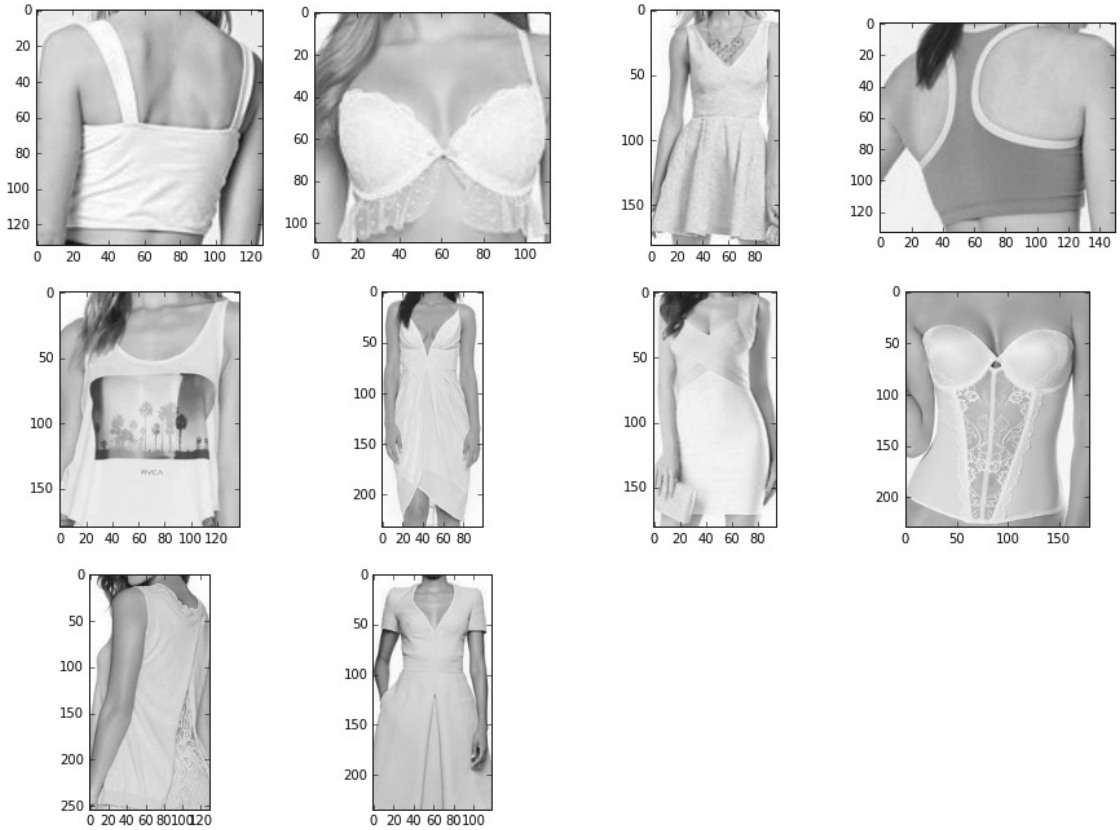
Similarity



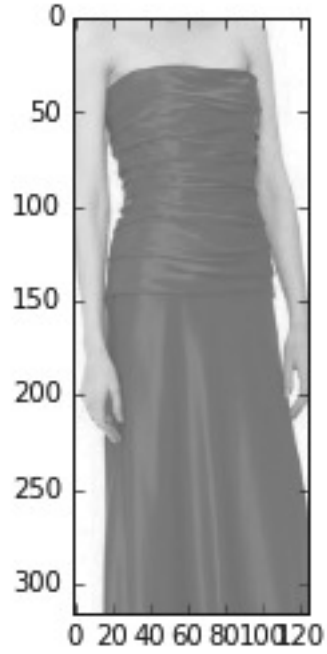
Query



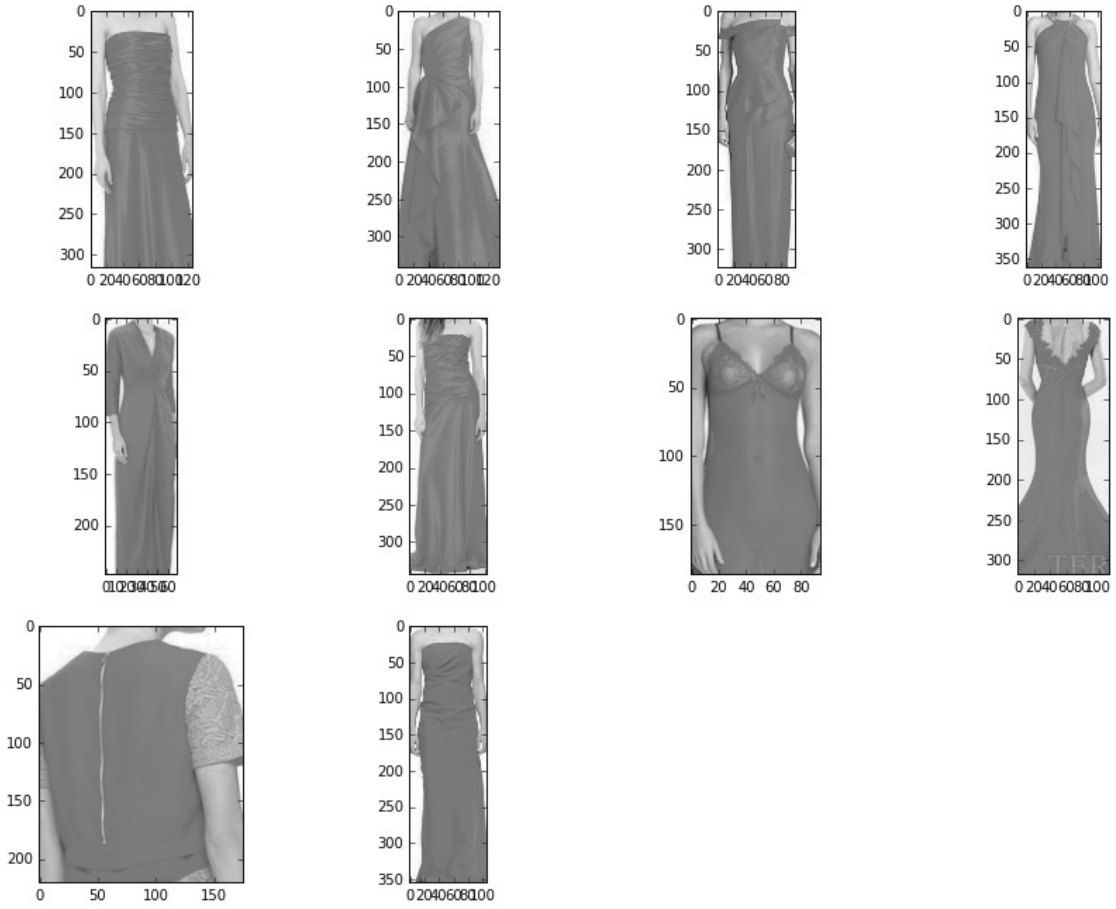
Similarity



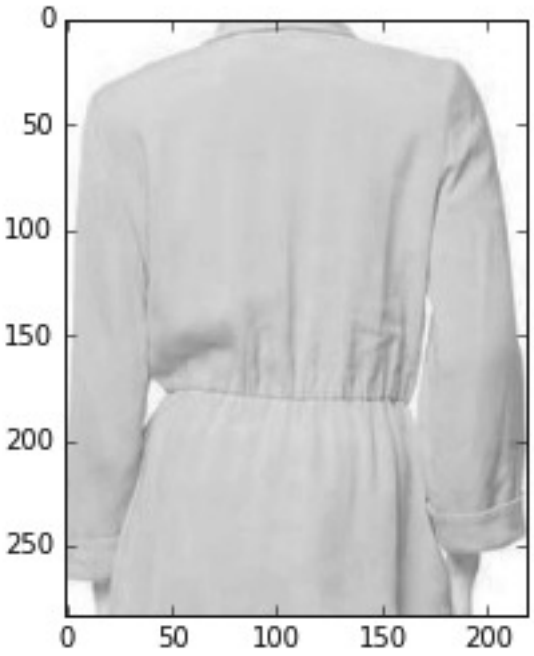
Query



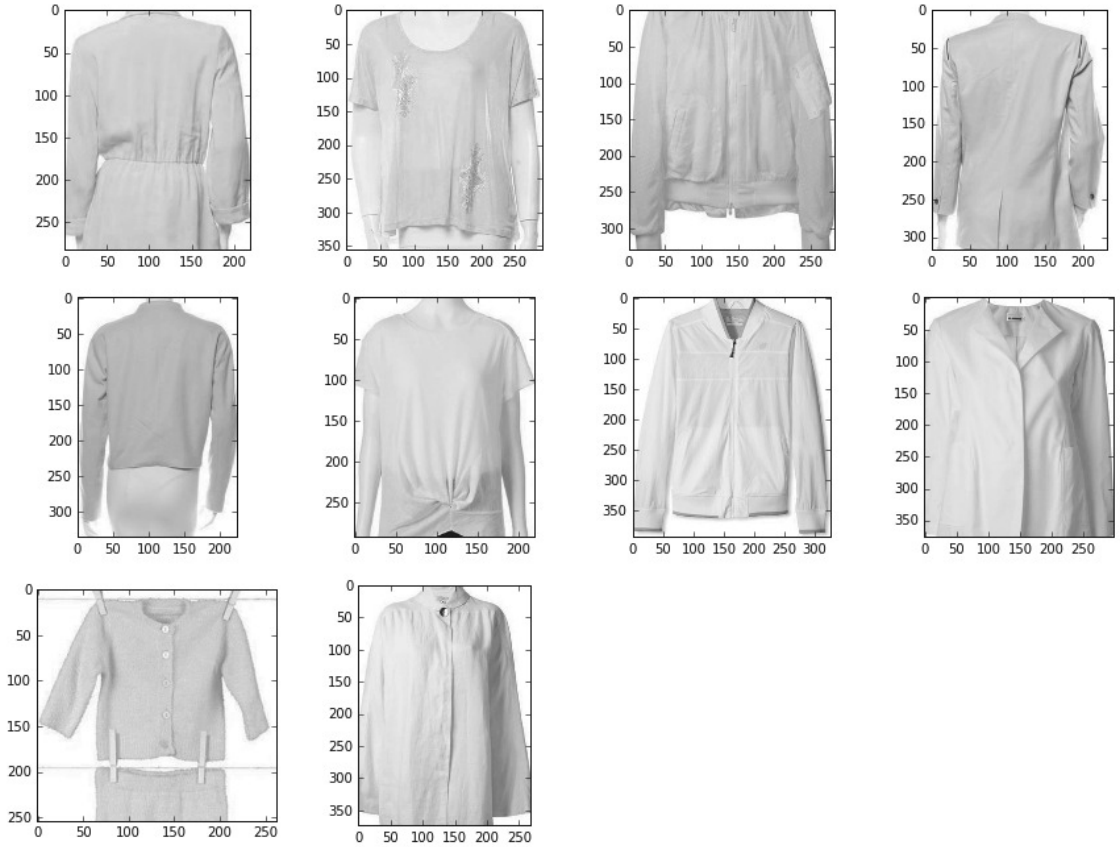
Similarity



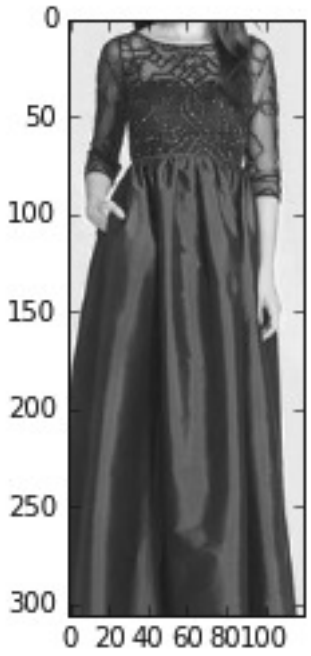
Query



Similarity

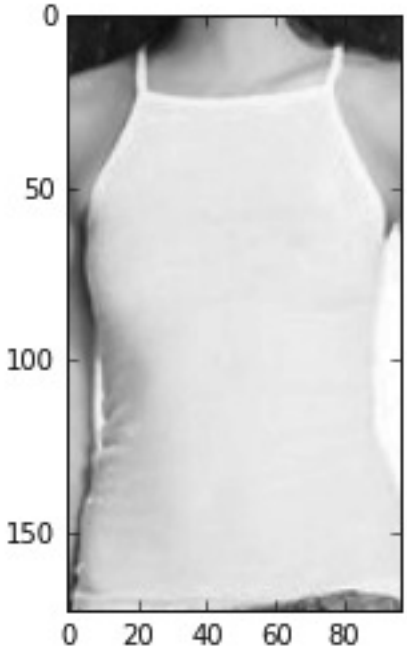


Query

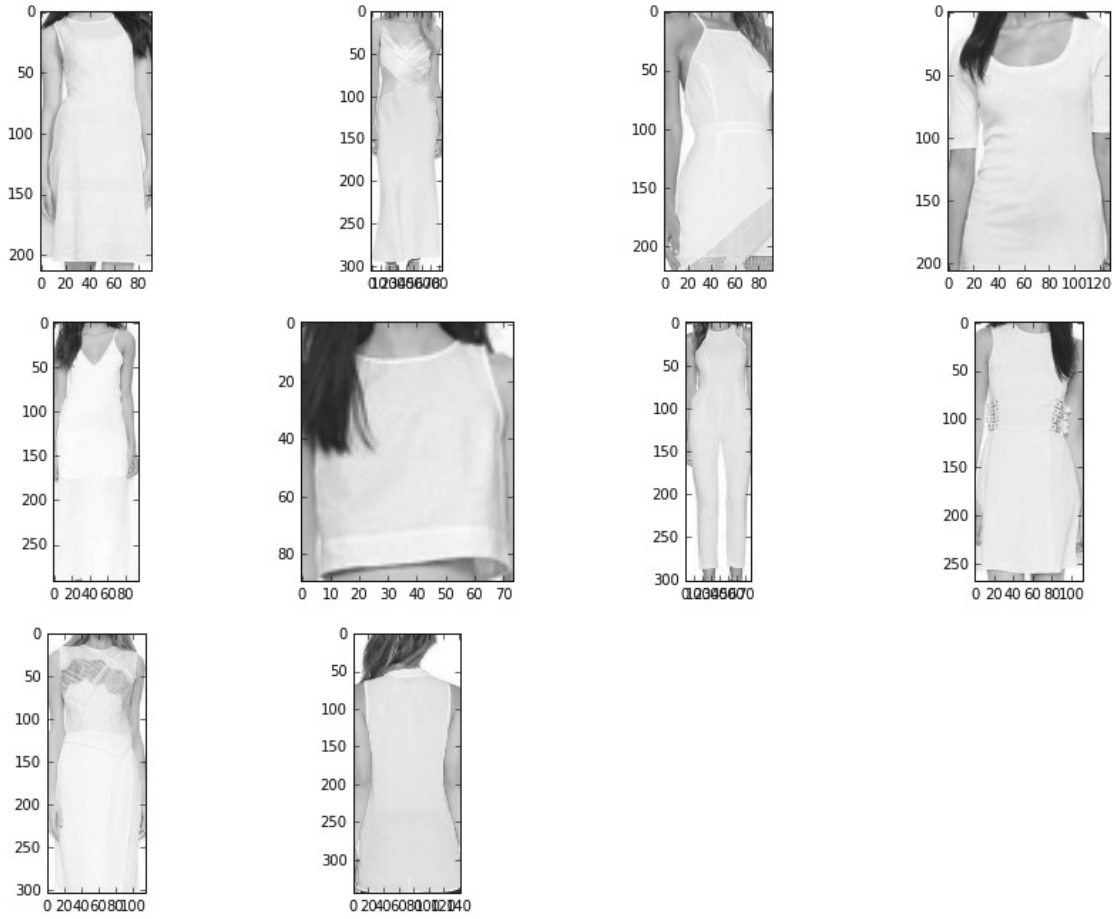


Similarity

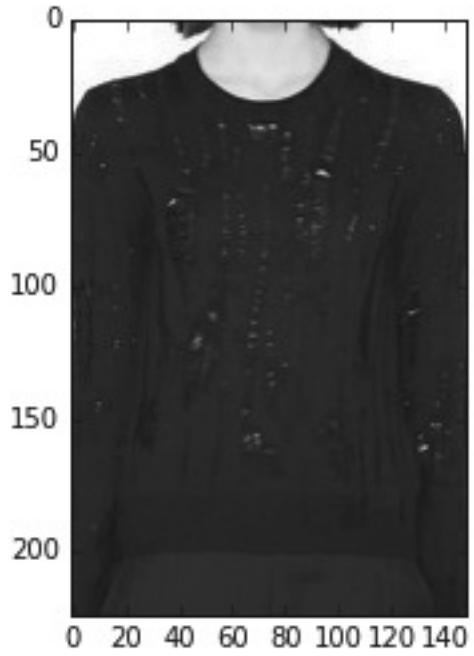
Query



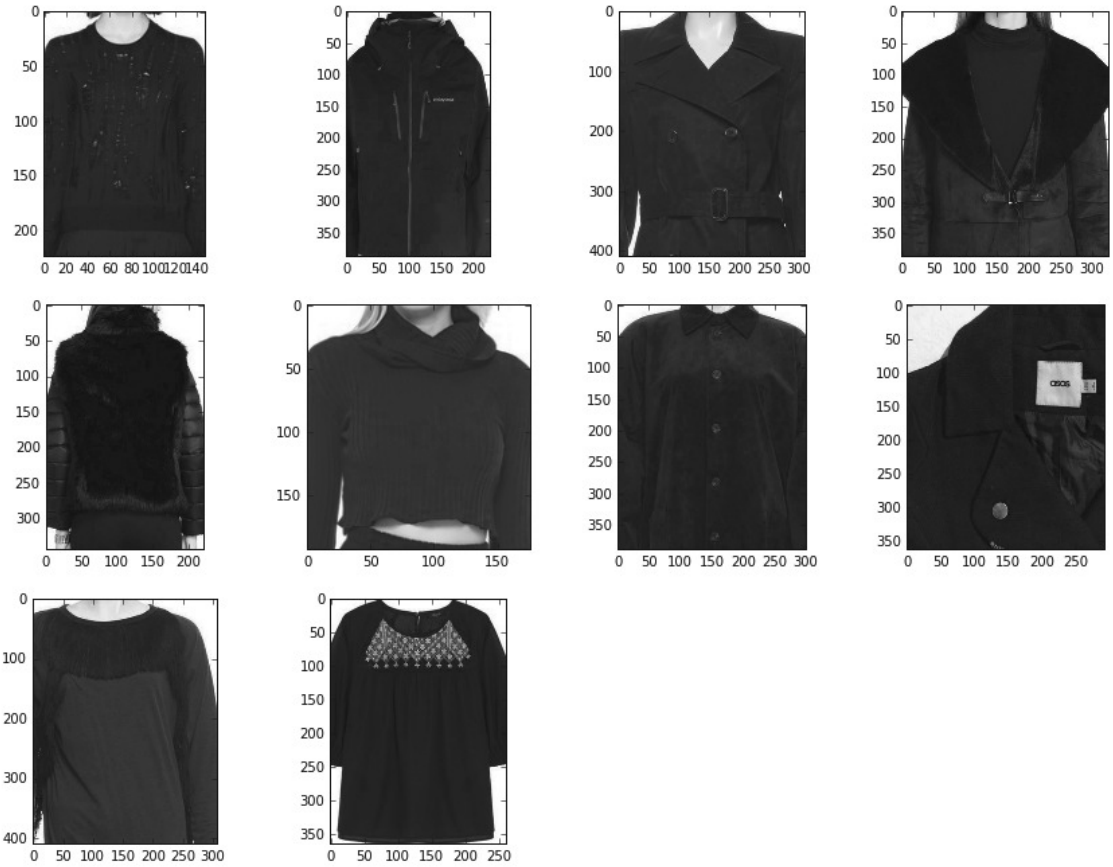
Similarity



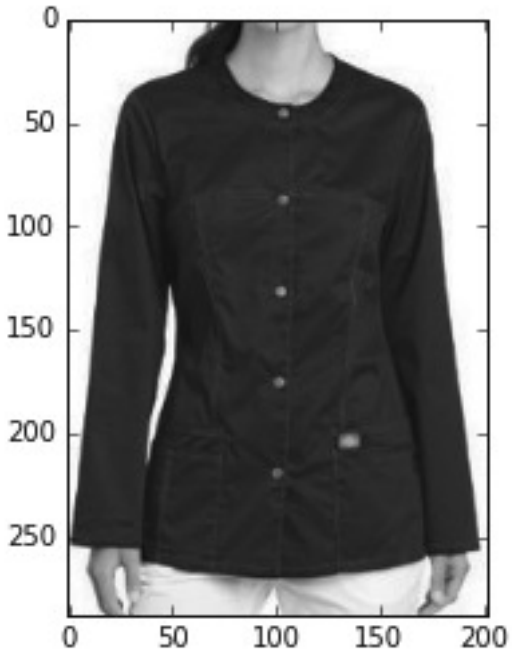
Query



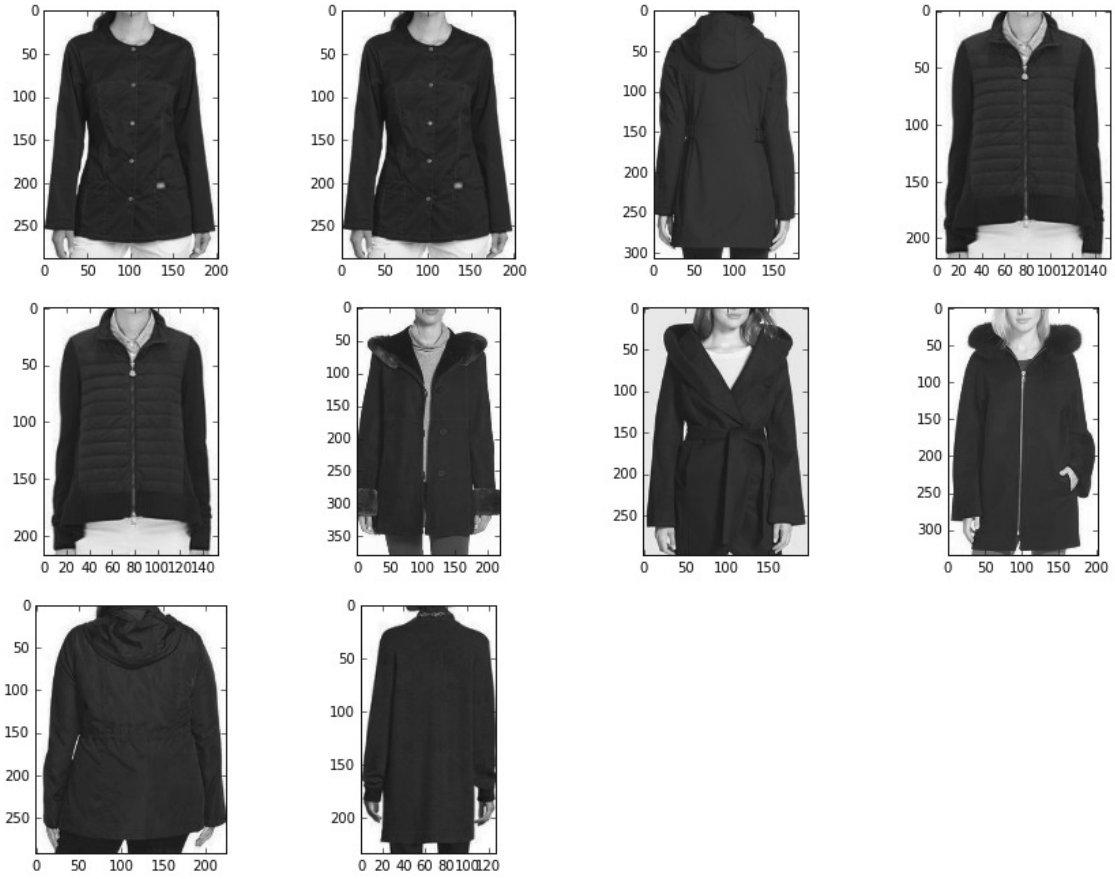
Similarity



Query



Similarity

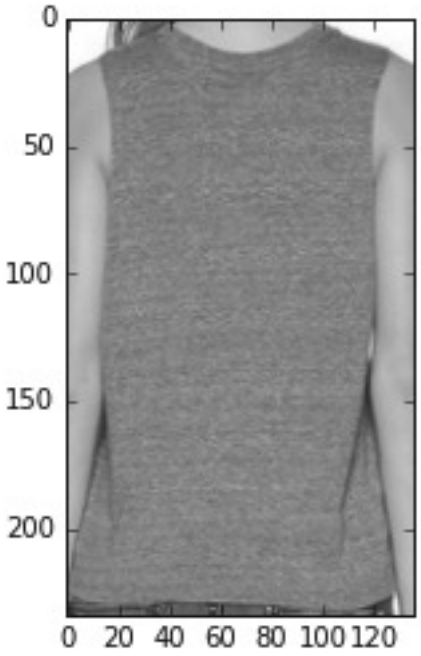


1.2 Pool 1 results

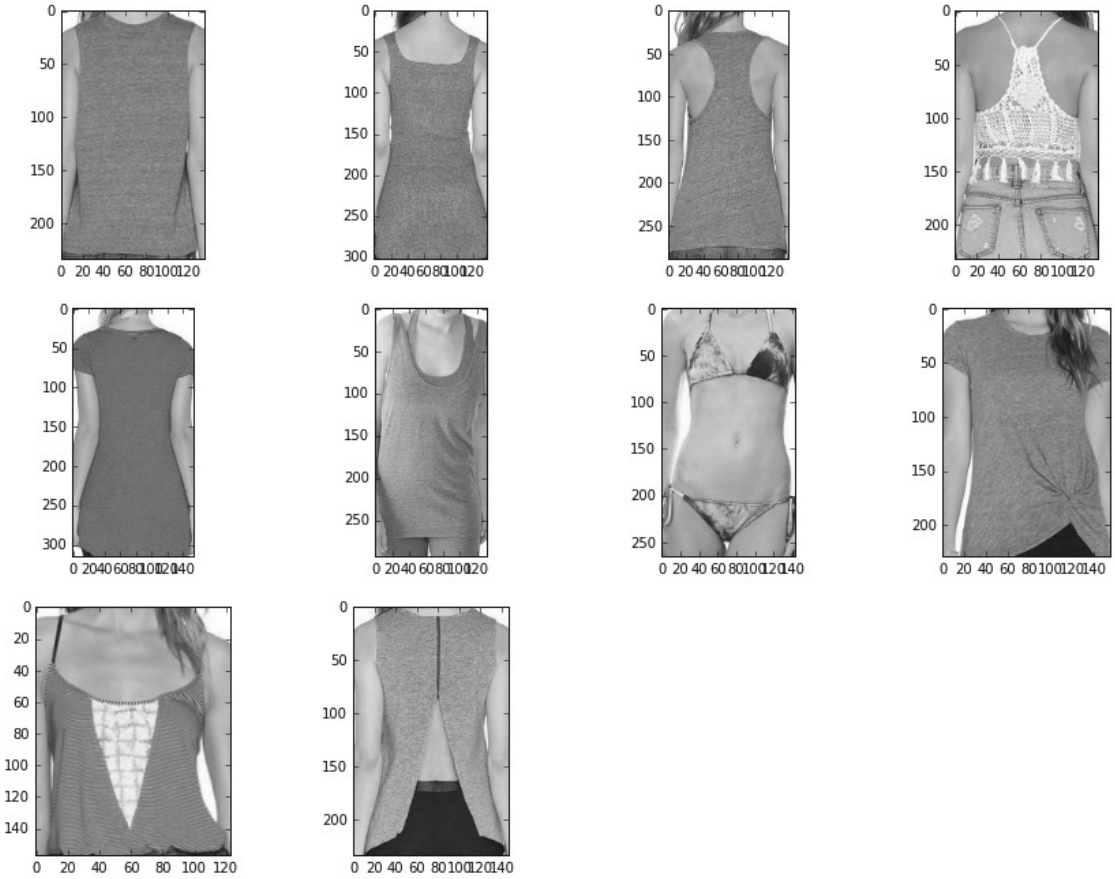
```
In [12]: index =  'index_pool_1.csv'
         # initialize the image descriptor
         cd = DeepDescriptor('pool1','VGG')

         for ilen in range(len(Query_list)):
           query = Query_list[ilen];
           img = cv2.imread(query);
           print "Query"
         # display the query
           plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB));

           plt.show();
           features = cd.describe(query);
           print features.shape

         # perform the search
           searcher = Searcher(index,"pool1");
           results_pool_1 = searcher.search(features)
         #
           plt.figure(figsize=(15,15));
           lenf = len(results_pool_1);
```

```
  lensq = math.ceil(math.sqrt(lenf));
  resultidx = 1;
  print "Similarity"
# loop over the results
  for (score, resultID) in results_pool_1:
        # load the result image and display it
    result = cv2.imread(result_path + "/" + resultID);
    plt.subplot(lensq,lensq,resultidx);
    plt.imshow(cv2.cvtColor(result,cv2.COLOR_BGR2RGB));
    resultidx += 1;
  plt.show();
#        cv2.waitKey(0)
```

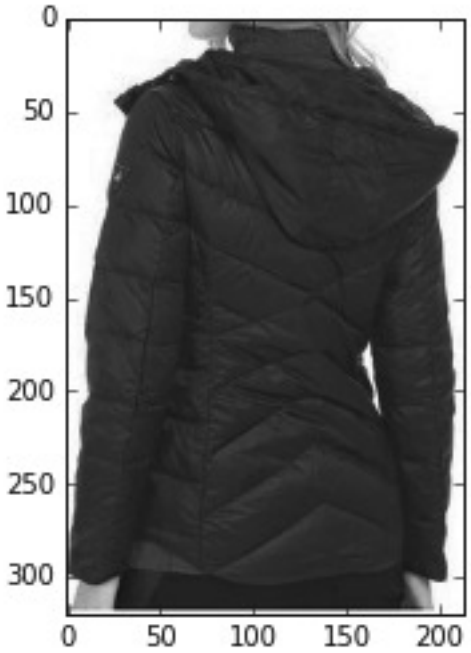
```
Query
```



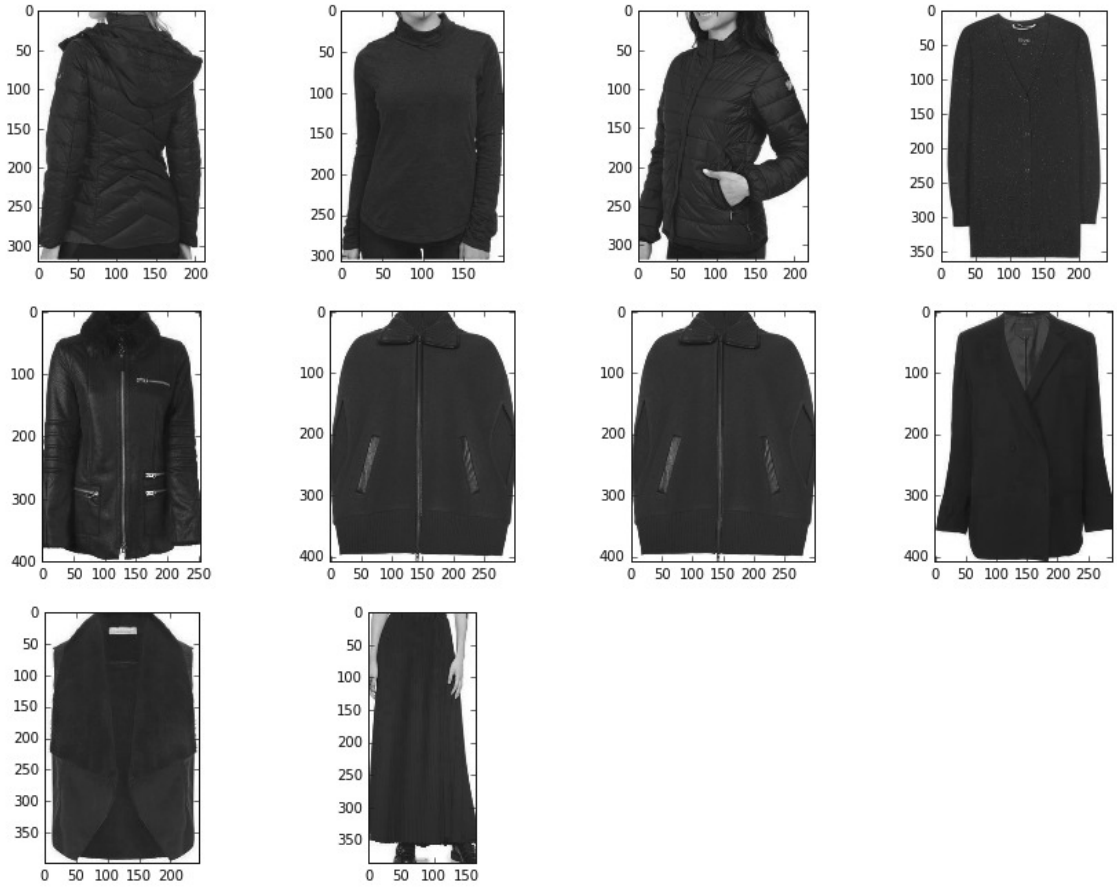

```
(64, 64)
Similarity
```



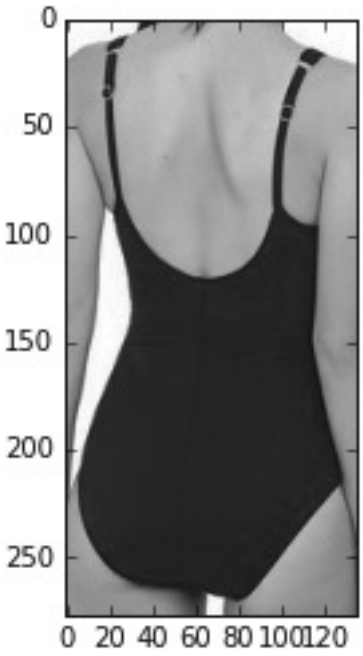
Query



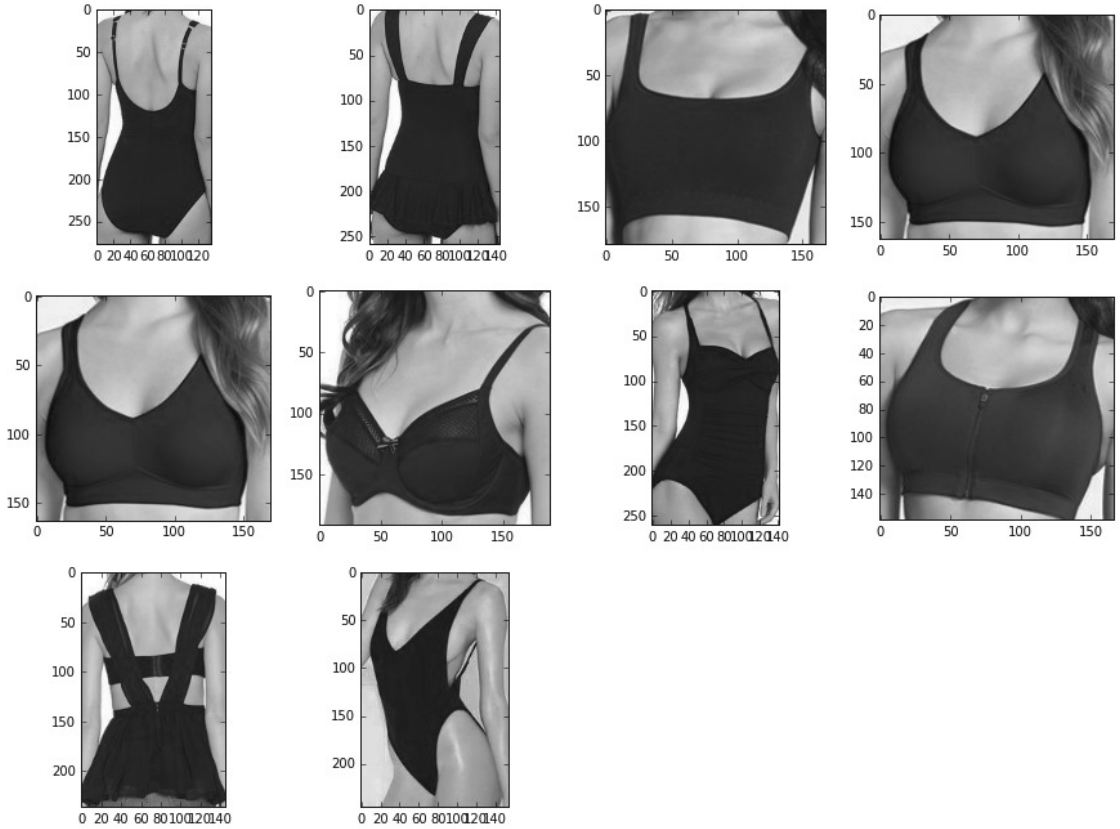
(64, 64)
Similarity



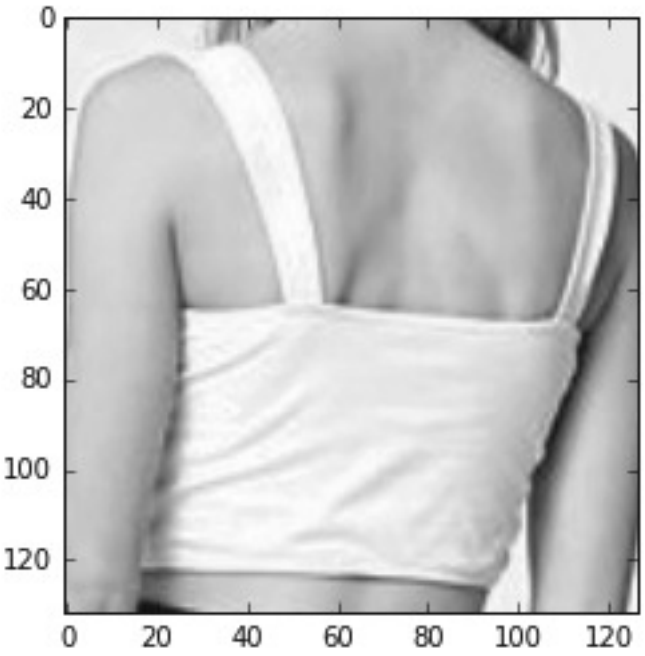
Query



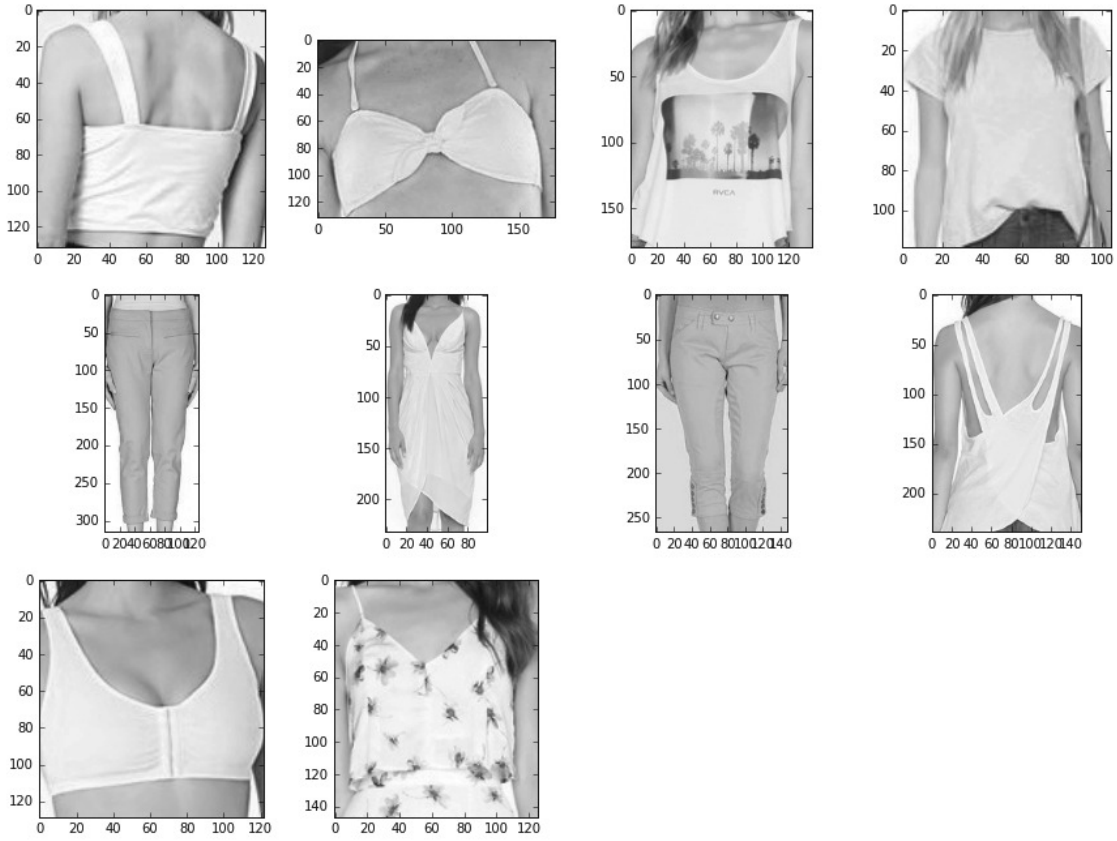
(64, 64)
Similarity



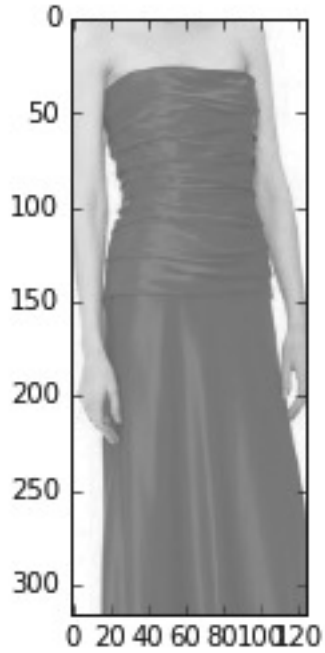
Query



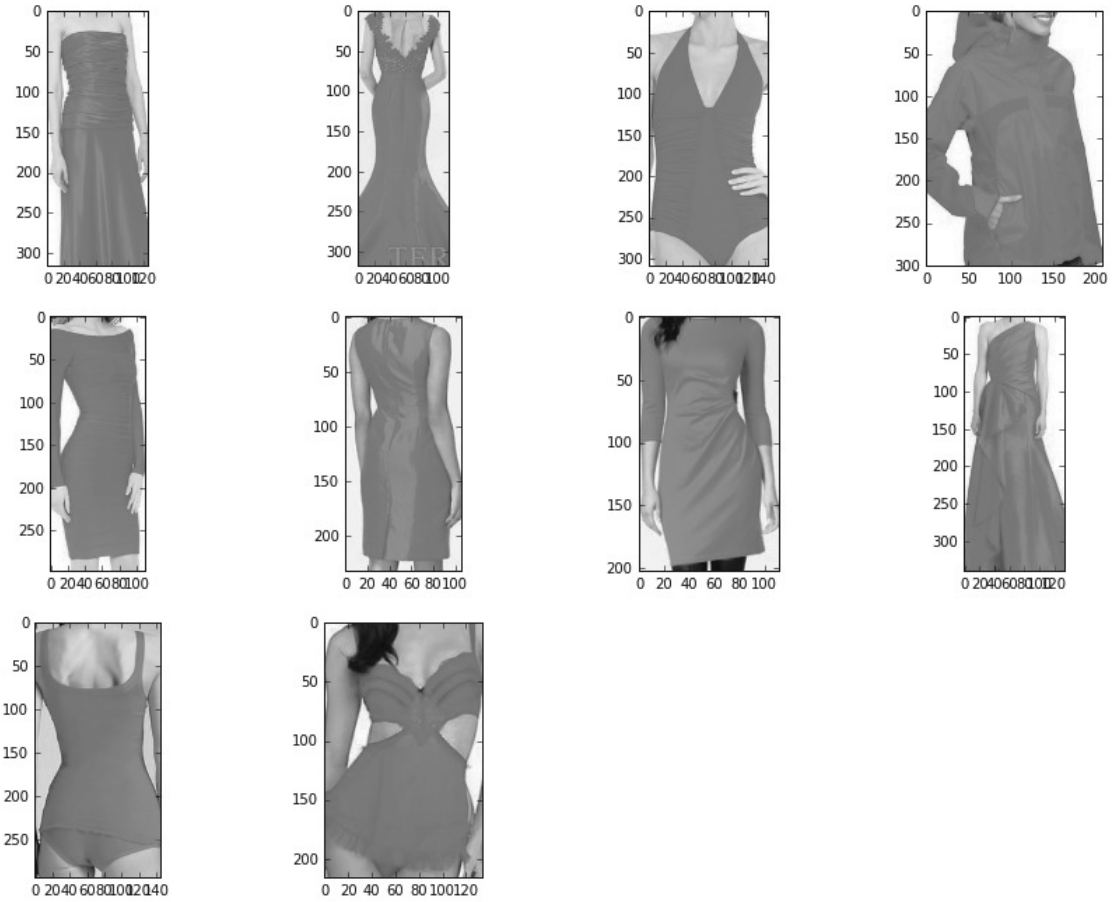
(64, 64)
Similarity



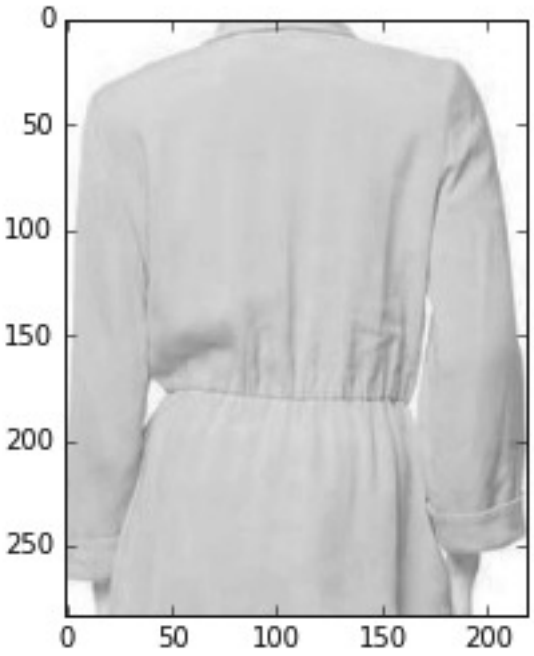
Query



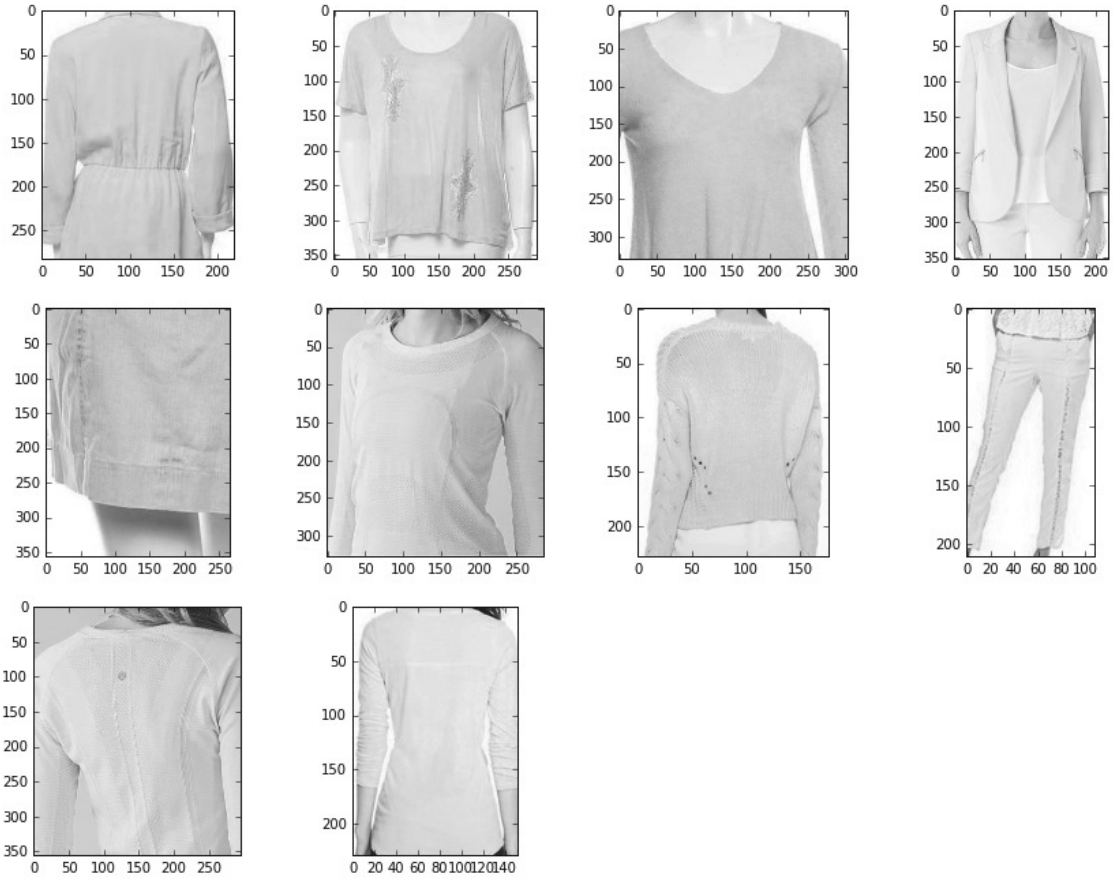
(64, 64)
Similarity



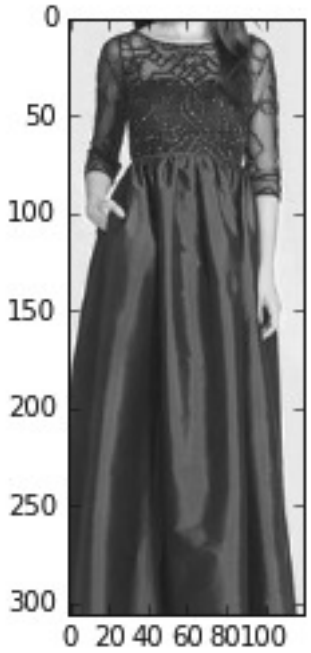
Query



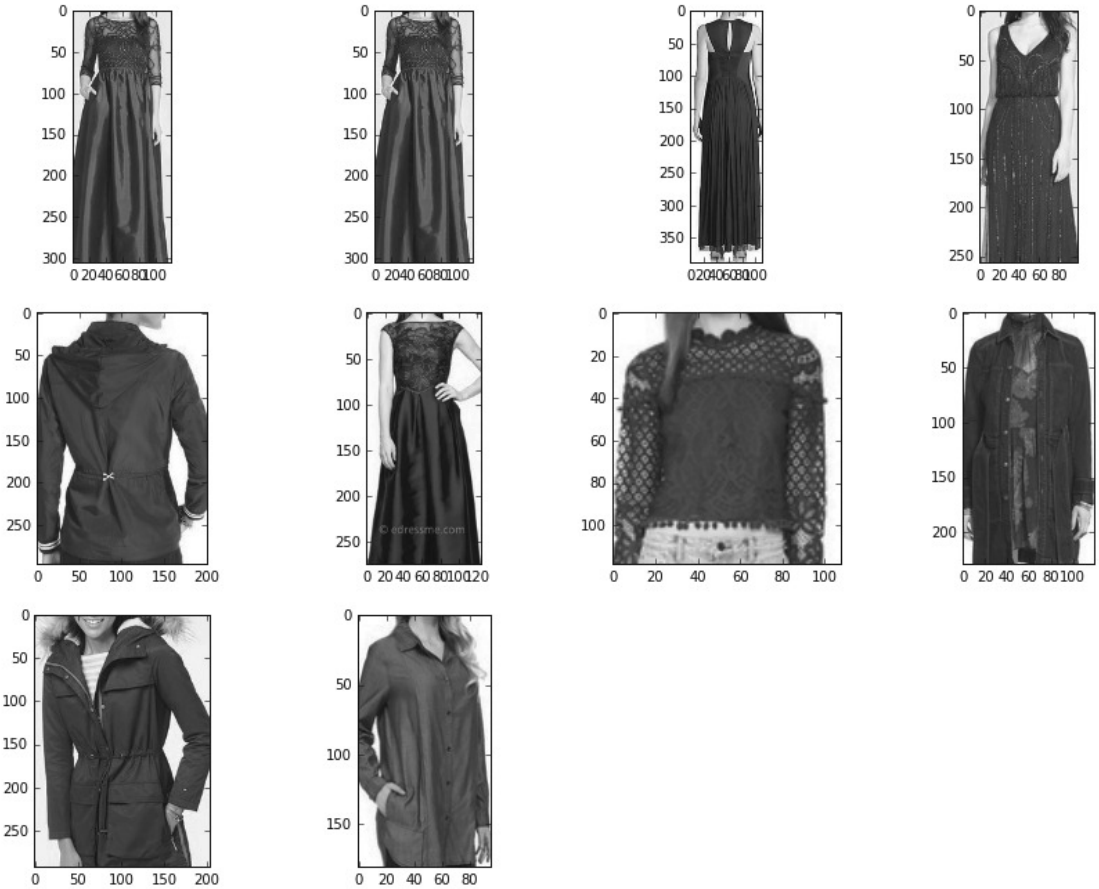

```
(64, 64)
Similarity
```



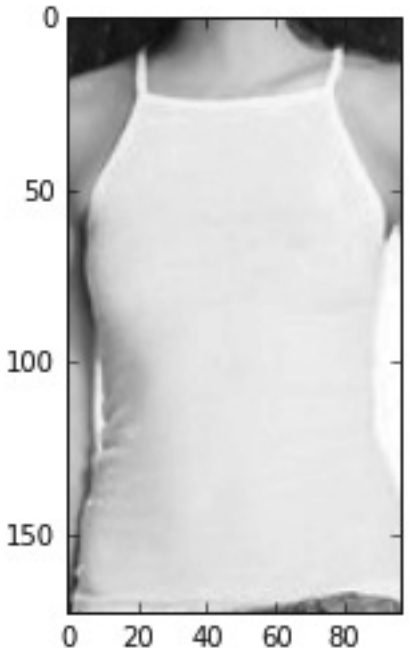
Query



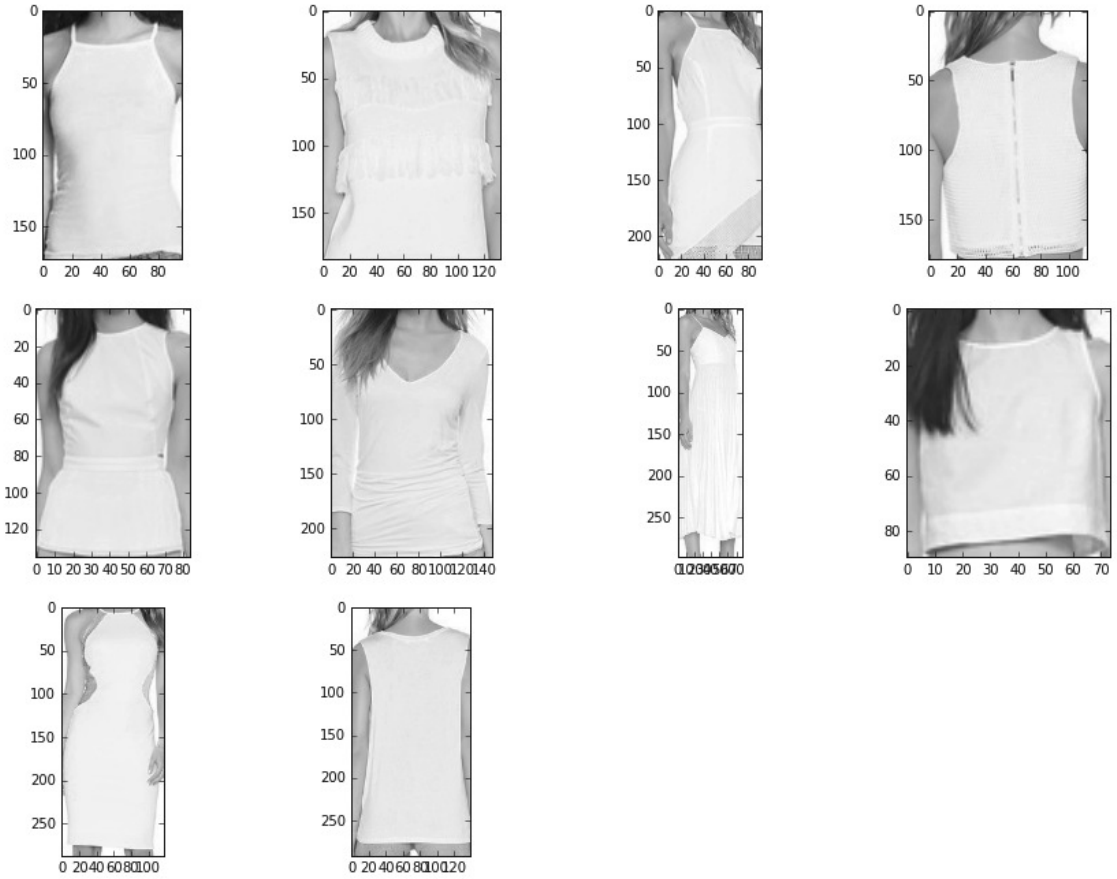
(64, 64)
Similarity



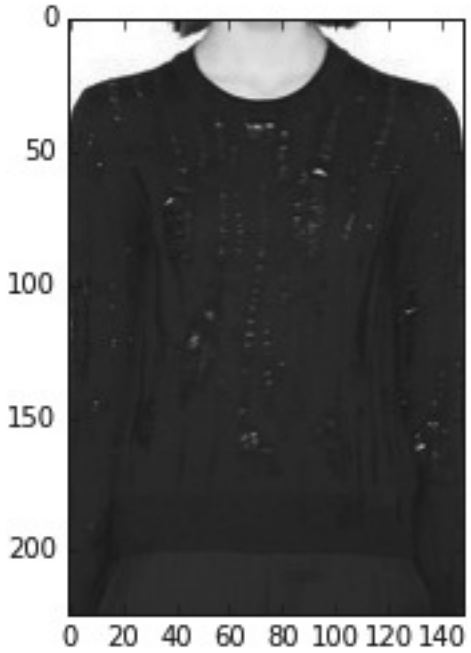
Query



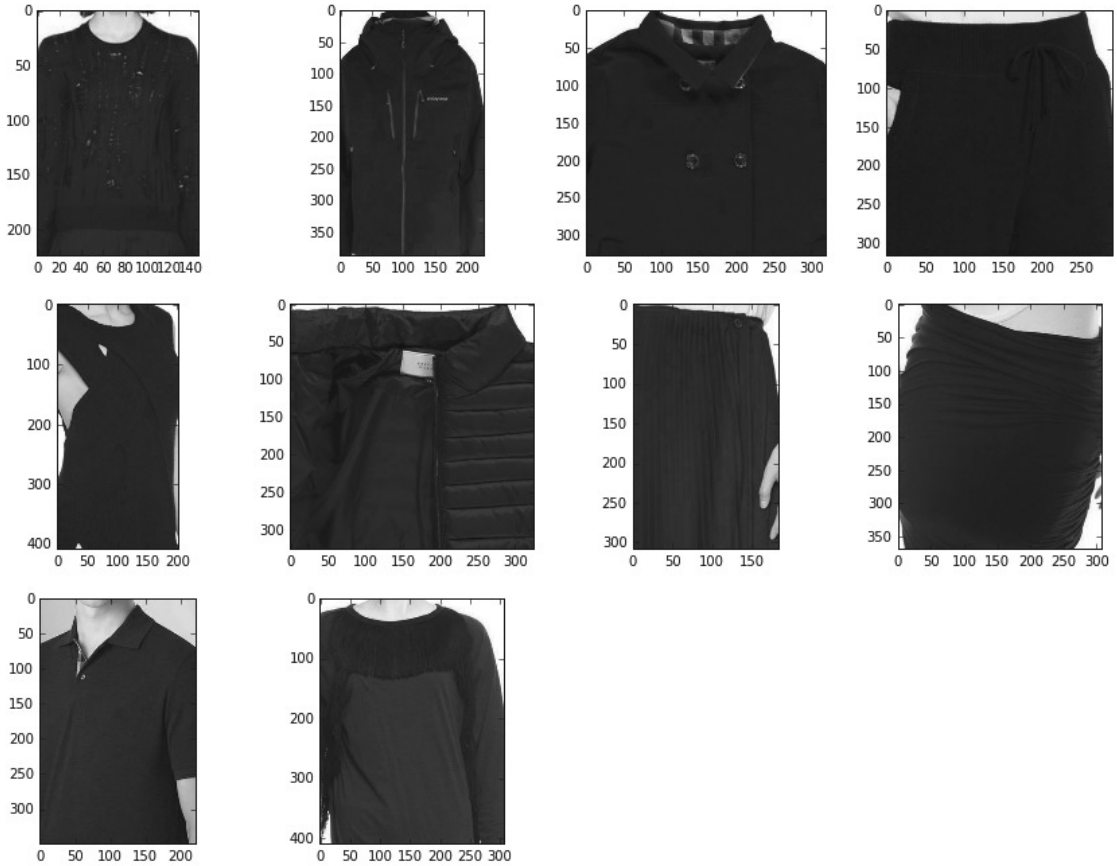

```
(64, 64)
Similarity
```



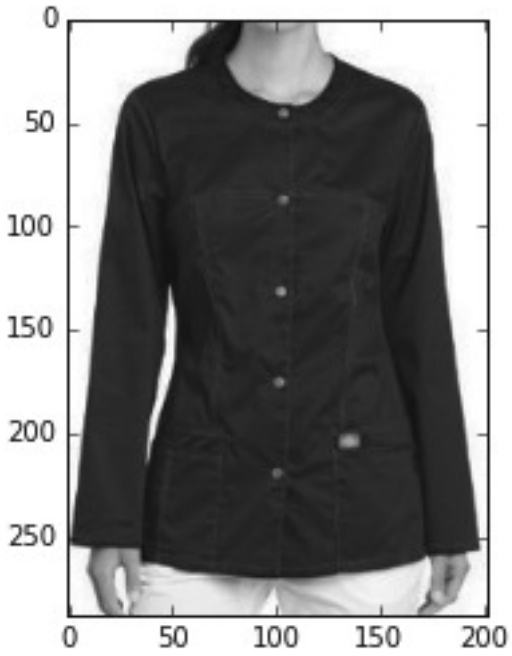
Query



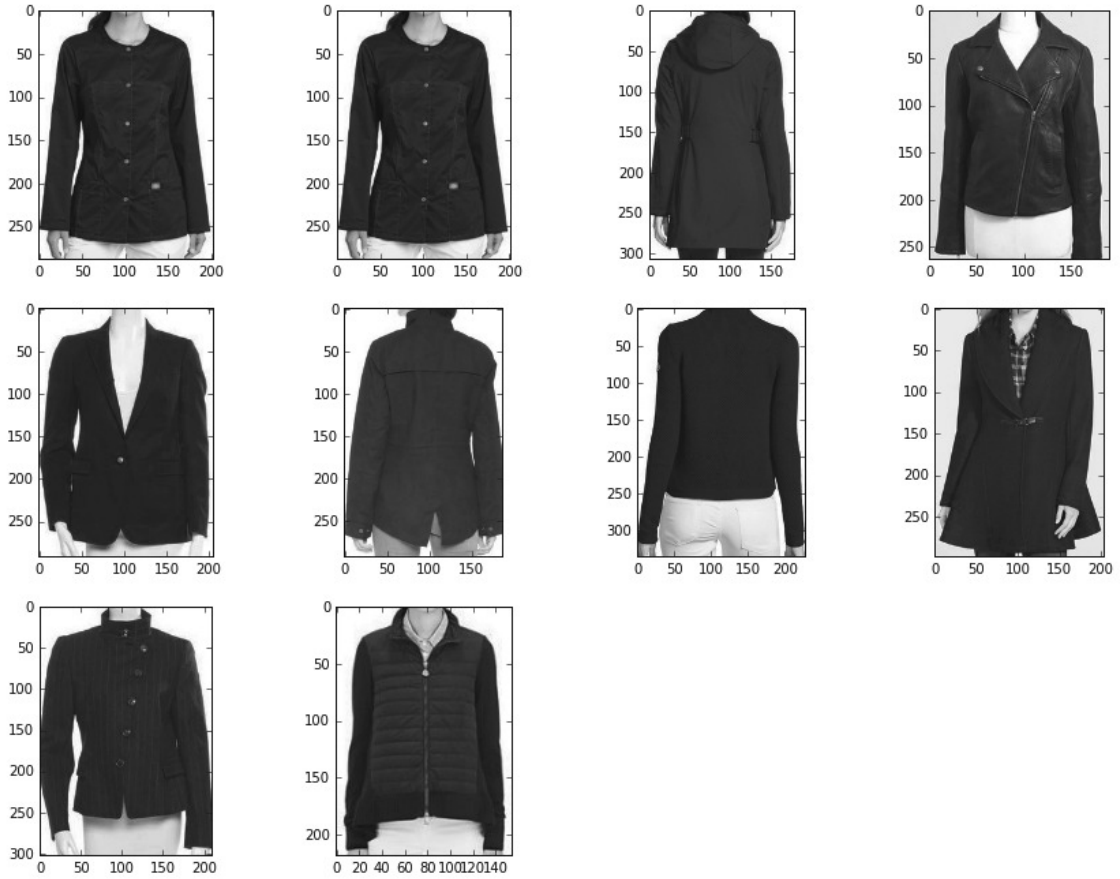
(64, 64)
Similarity



Query



(64, 64)
Similarity



In [12]:

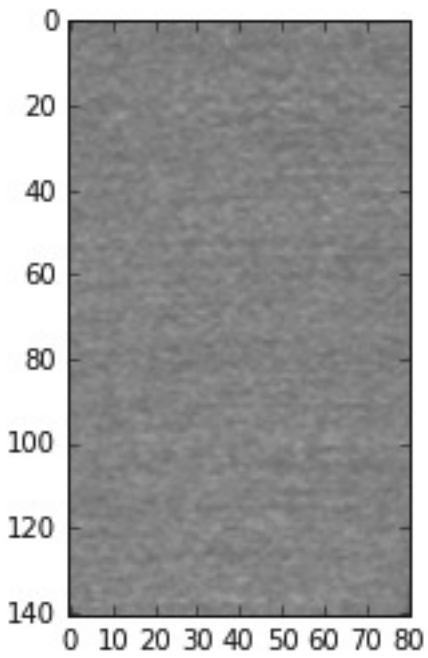
2 Using Pool 2 activations with cropped images¶

```
In [13]: index =  'index_crop_pool_2.csv'

         # initialize the image descriptor

         cd = DeepDescriptor('pool2','VGG')

         for ilen in range(len(Query_list)):
           query = Query_list[ilen];
           image = cv2.imread(query);
           (h, w) = image.shape[:2];
           r = 0.8;
           rh = math.ceil(r*h);
           wh = math.ceil(r*w);
           img = image[math.ceil((1-r)*h):math.ceil(r*h), math.ceil((1-r)*w):math.ceil(r*w)];

           print "query"
         # display the query
           plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB));
```

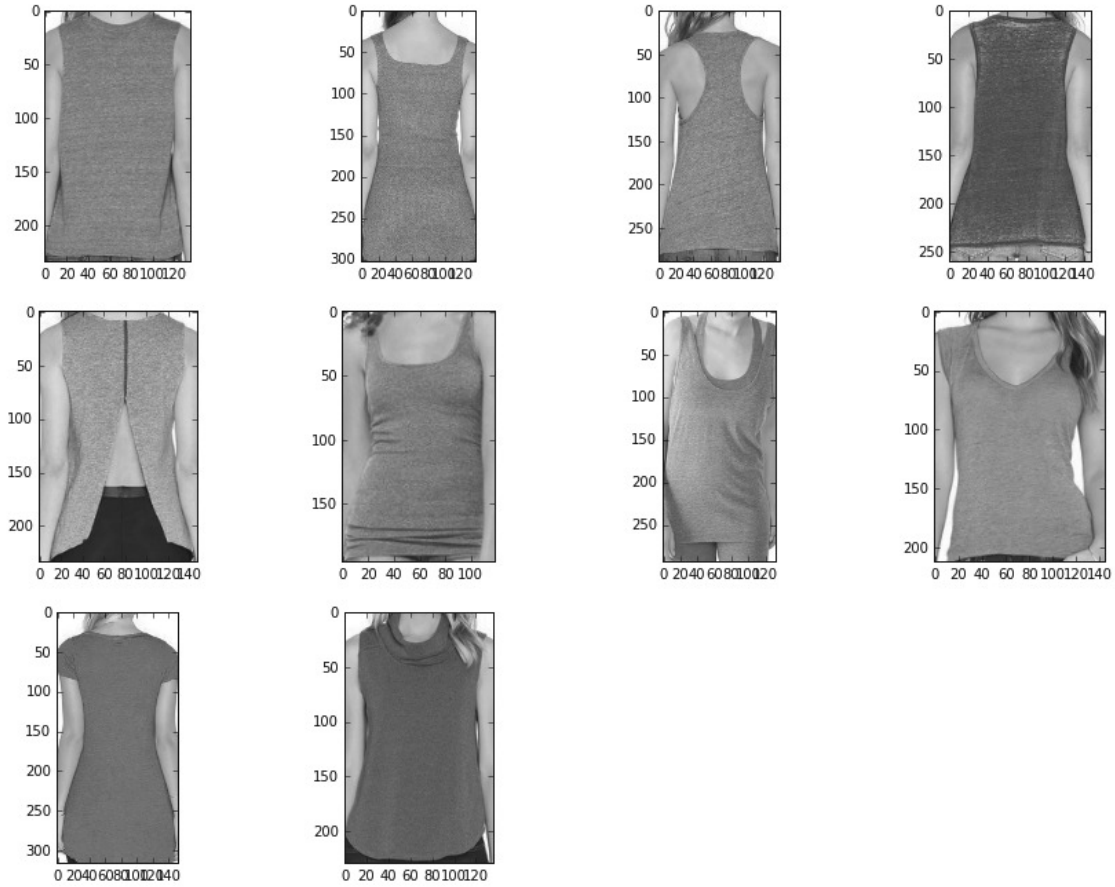
```
    plt.show();
    features = cd.describe(query);

  # perform the search
    searcher = Searcher(index,'pool2')
    results_pool_2_crop = searcher.search(features)
  #
    plt.figure(figsize=(15,15));
    lenf = len(results_pool_2_crop);
    lensq = math.ceil(math.sqrt(lenf));
    resultidx = 1;
    print "Similarity"
  # loop over the results
    for (score, resultID) in results_pool_2_crop:
          # load the result image and display it
      result = cv2.imread(result_path + "/" + resultID);
      plt.subplot(lensq,lensq,resultidx);
      plt.imshow(cv2.cvtColor(result,cv2.COLOR_BGR2RGB));
      resultidx += 1;
    plt.show();
```

query

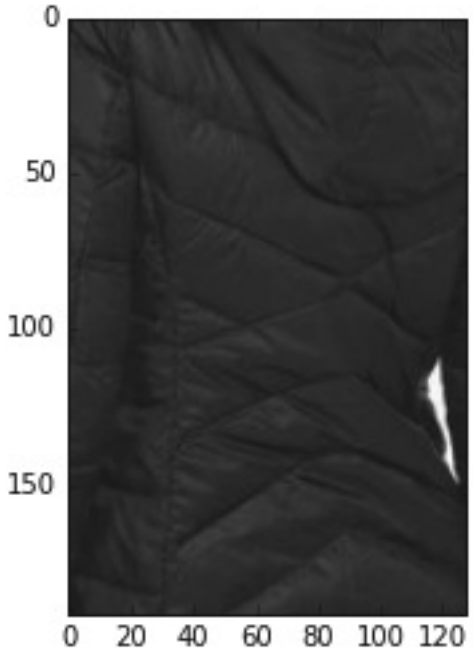
-c:15: VisibleDeprecationWarning: using a non-integer number instead of an integer will result in an er



Similarity

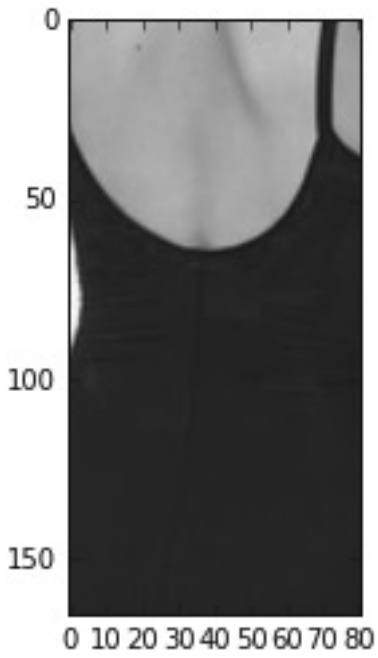


query

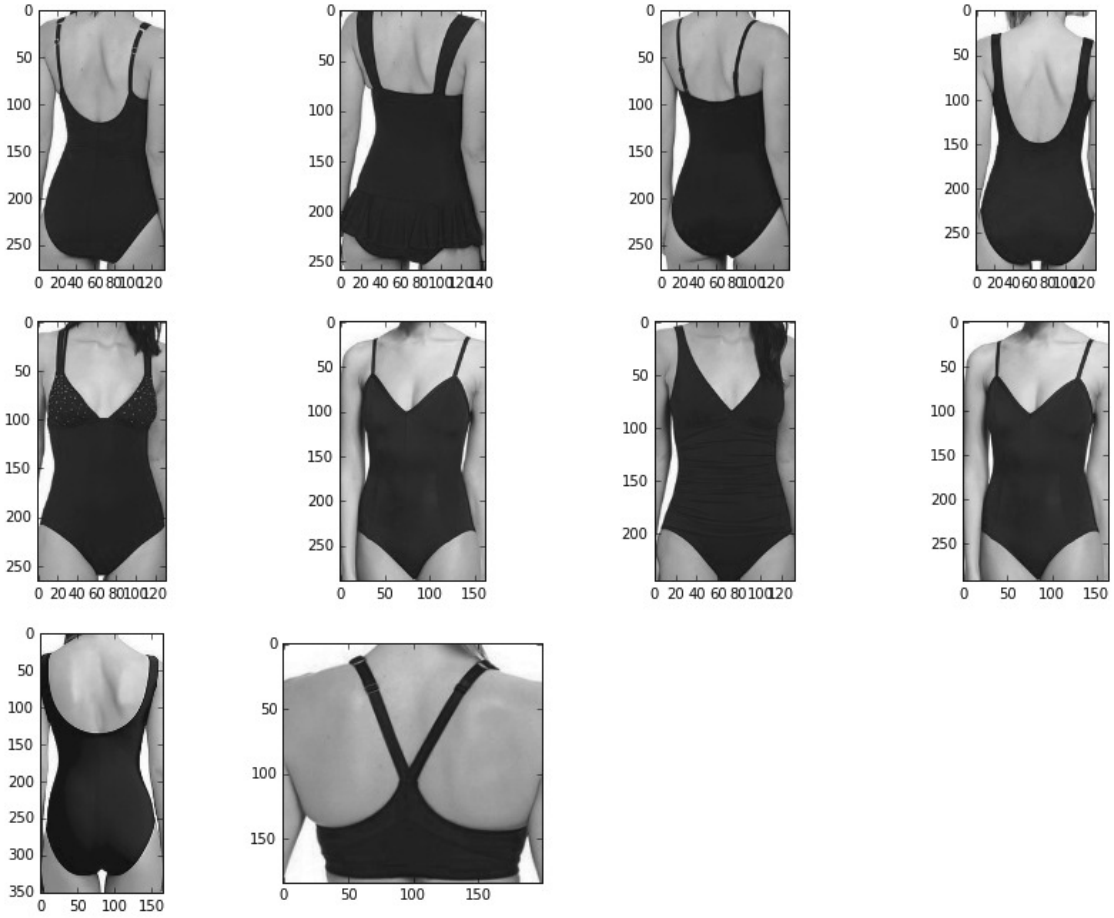


Similarity

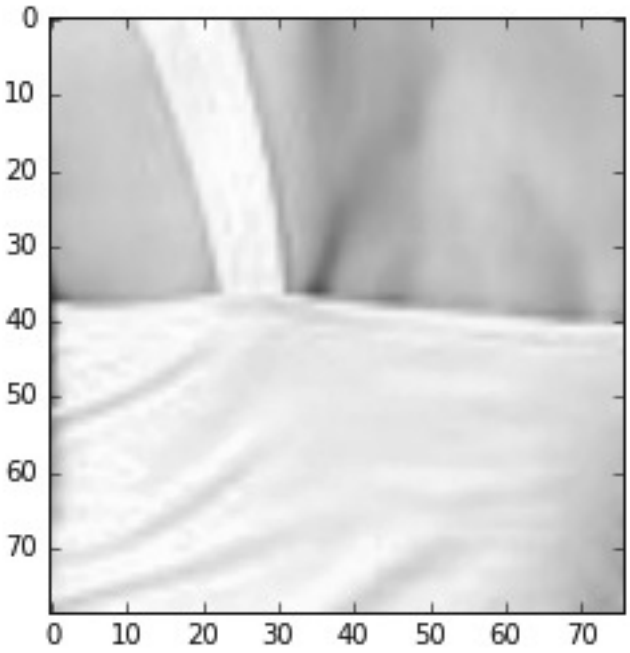
query



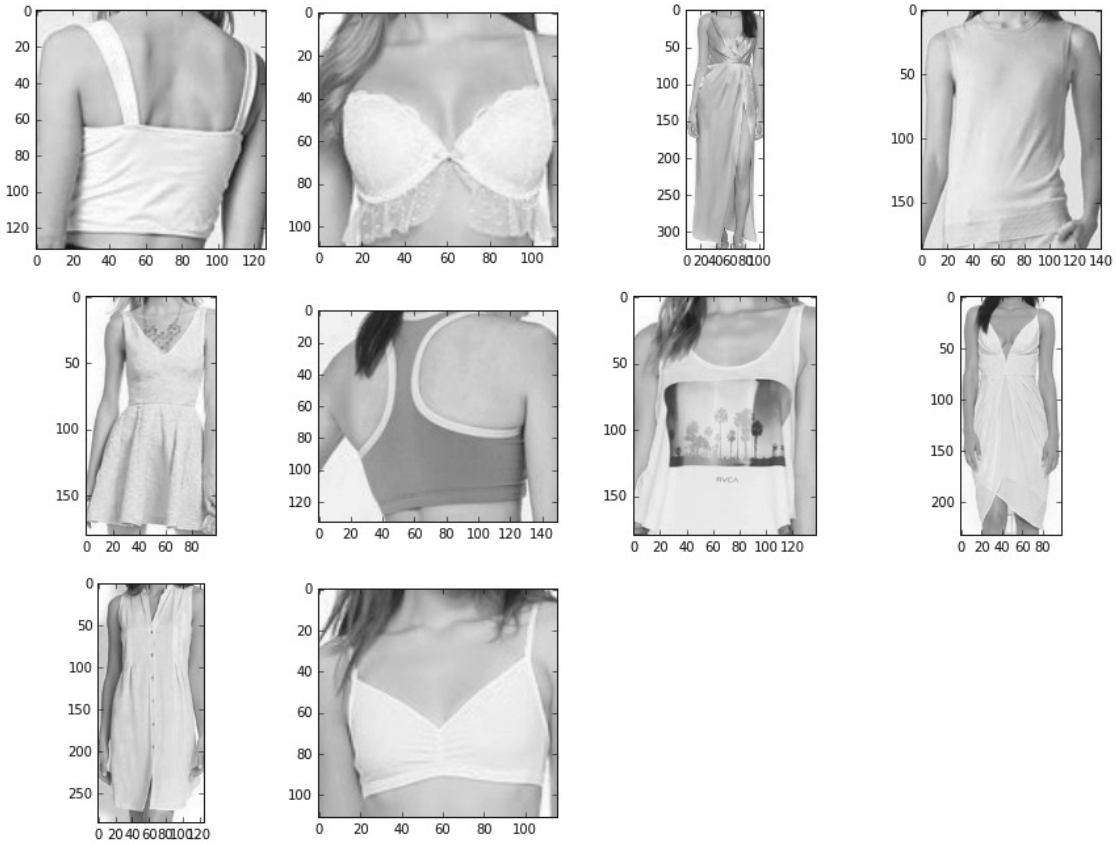
Similarity



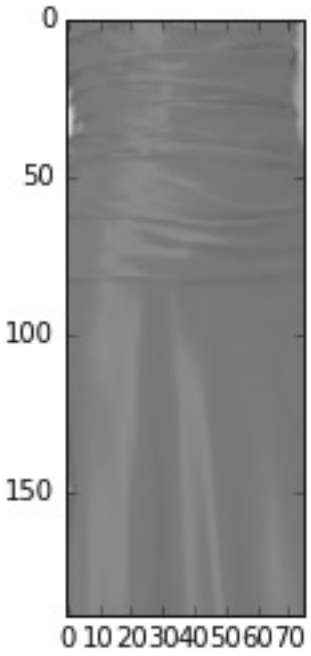
query



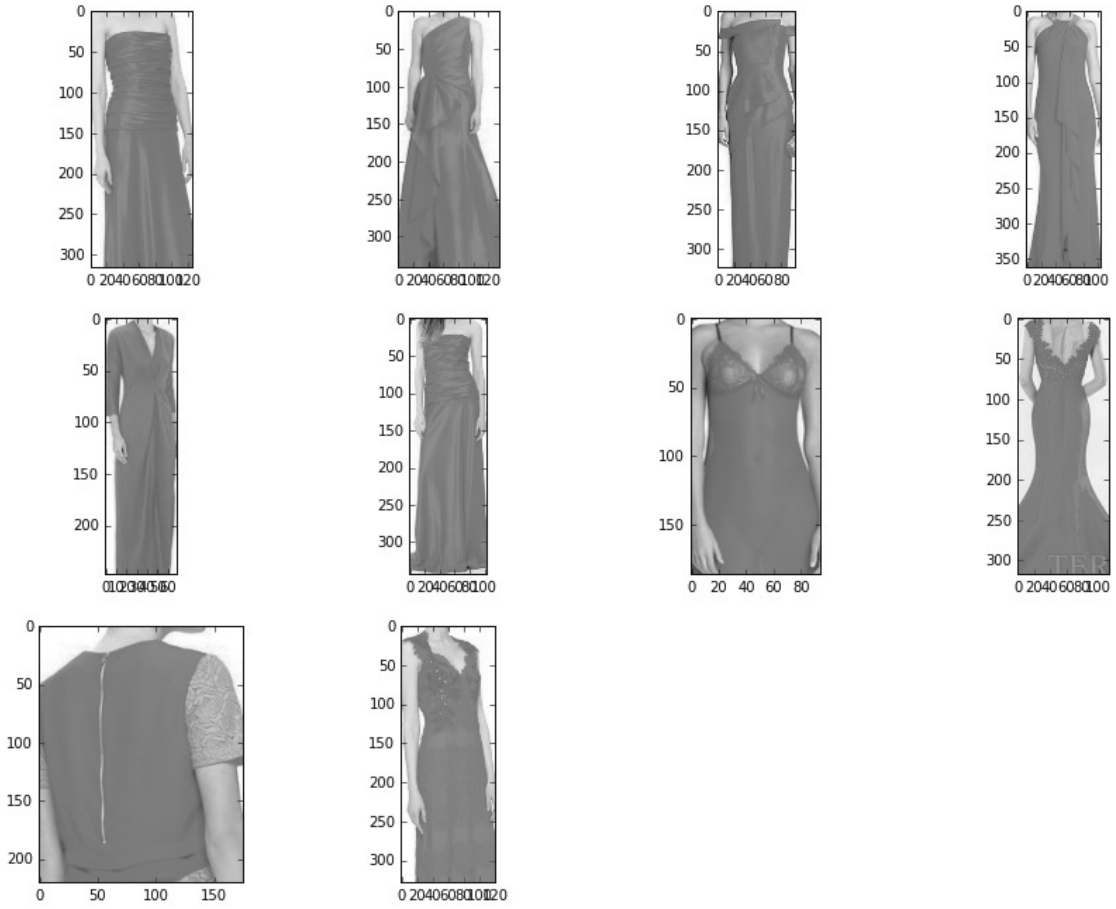
Similarity



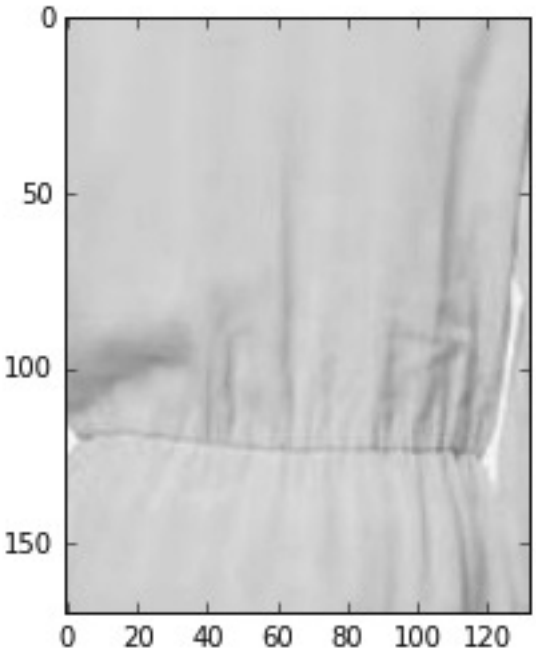
query



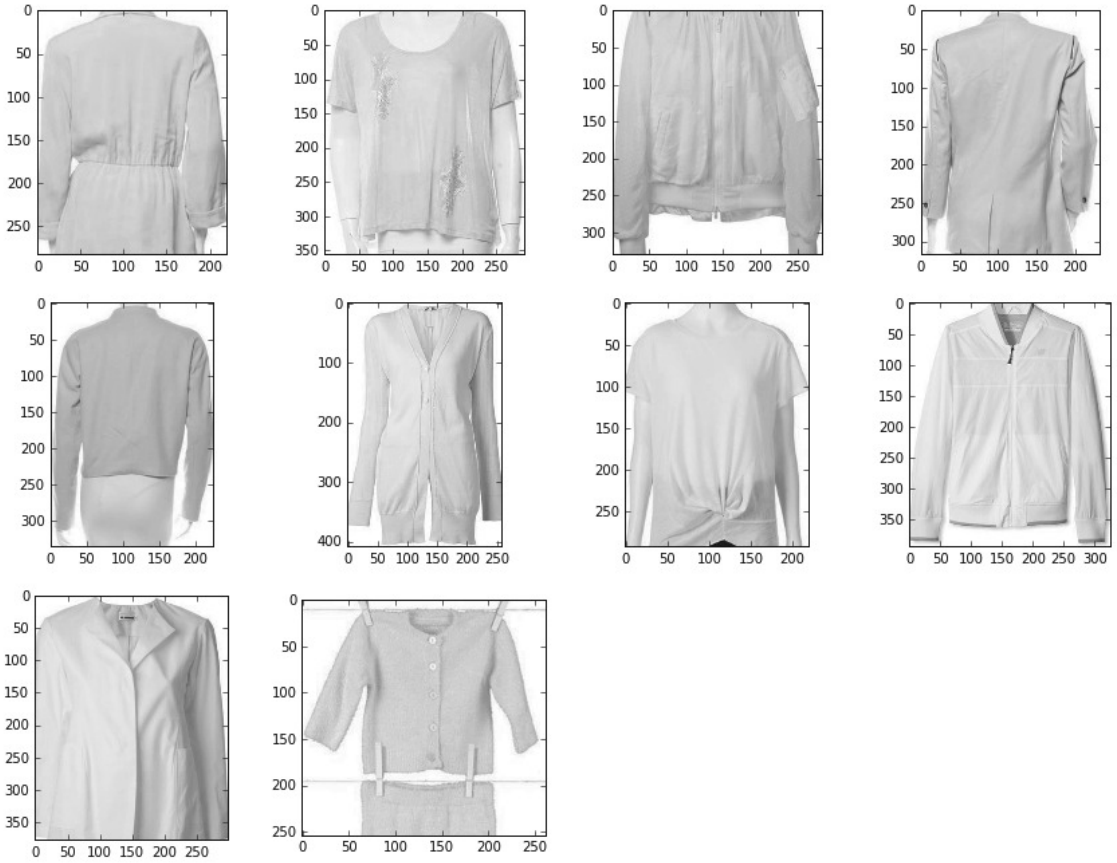
Similarity



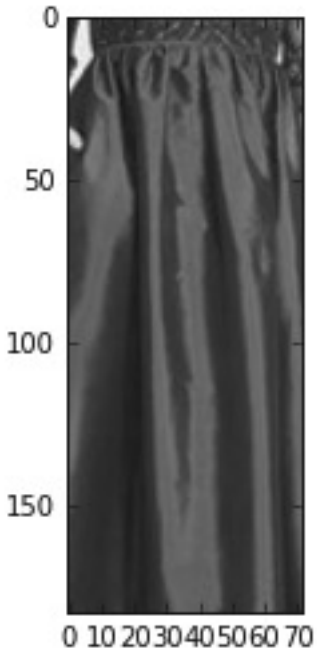
query



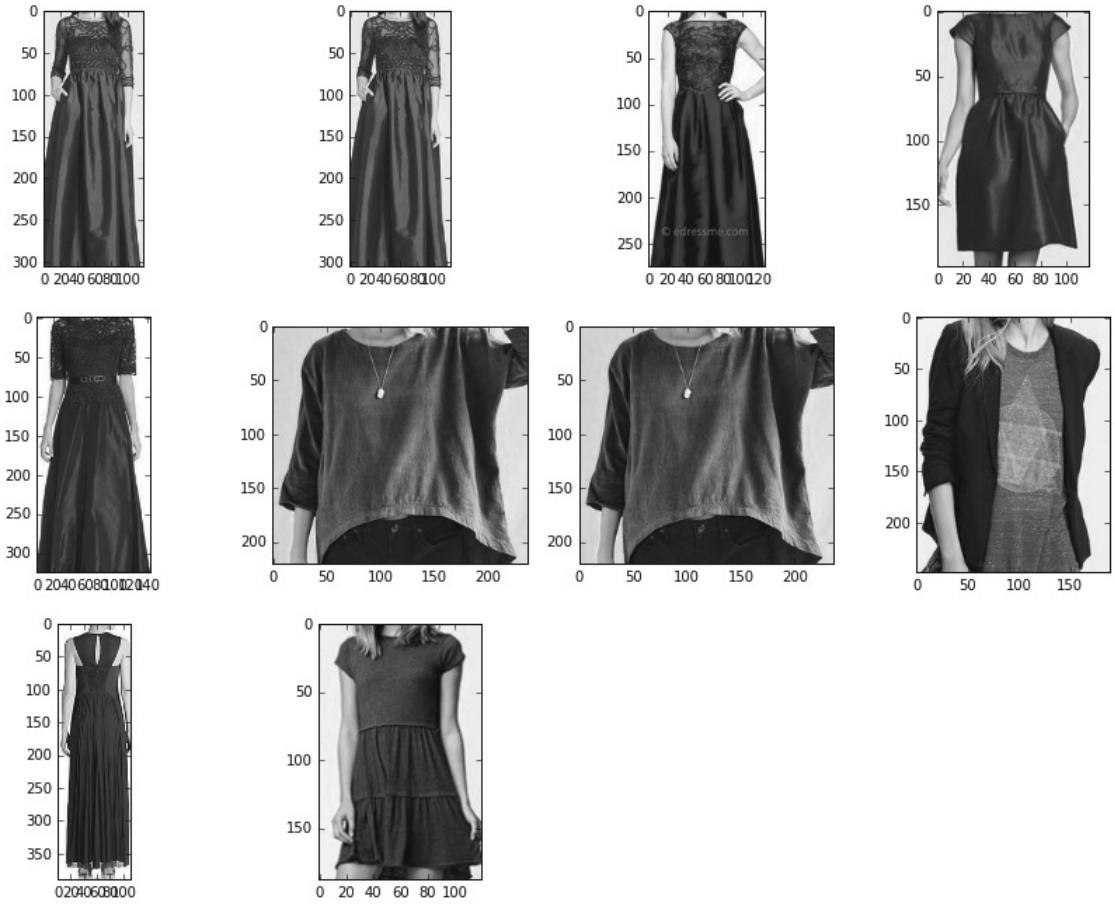
Similarity



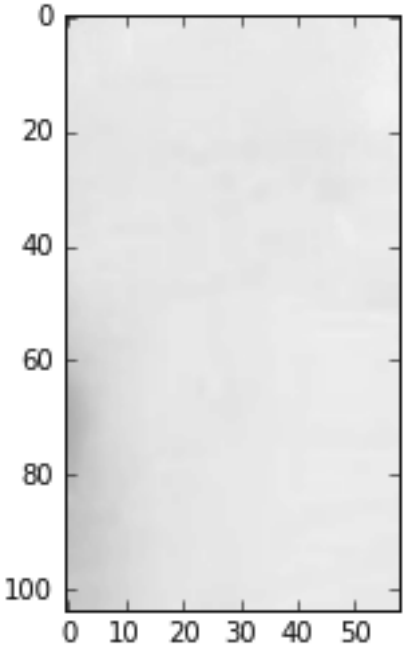
query



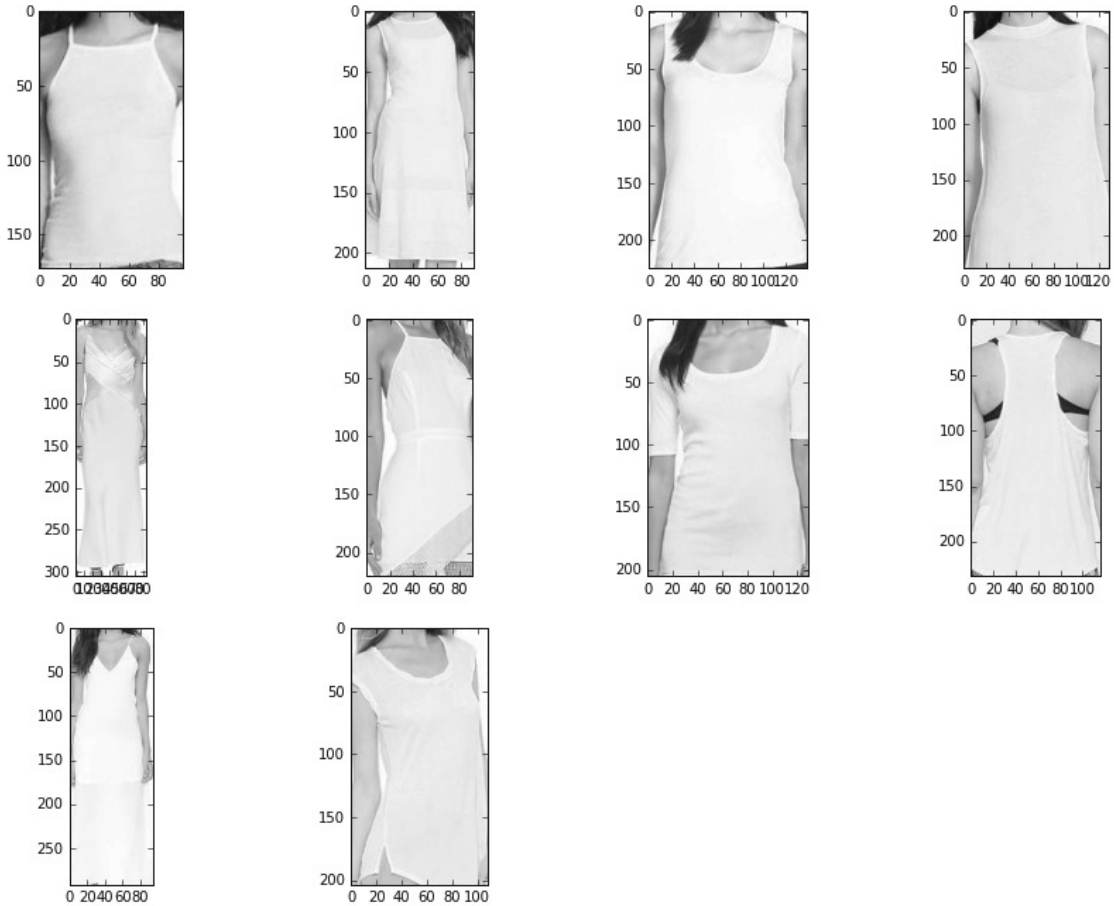
Similarity



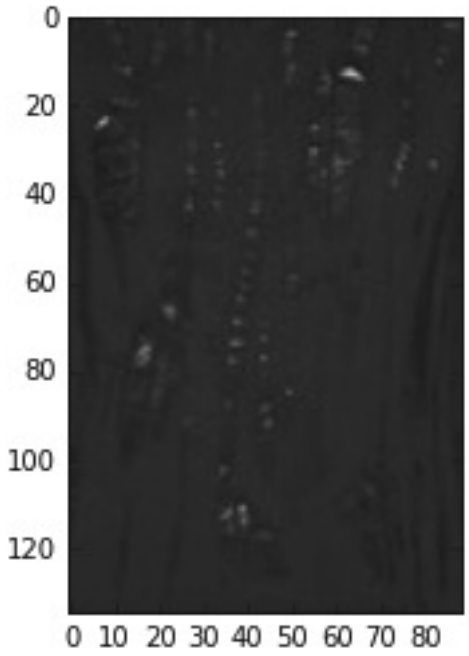
query



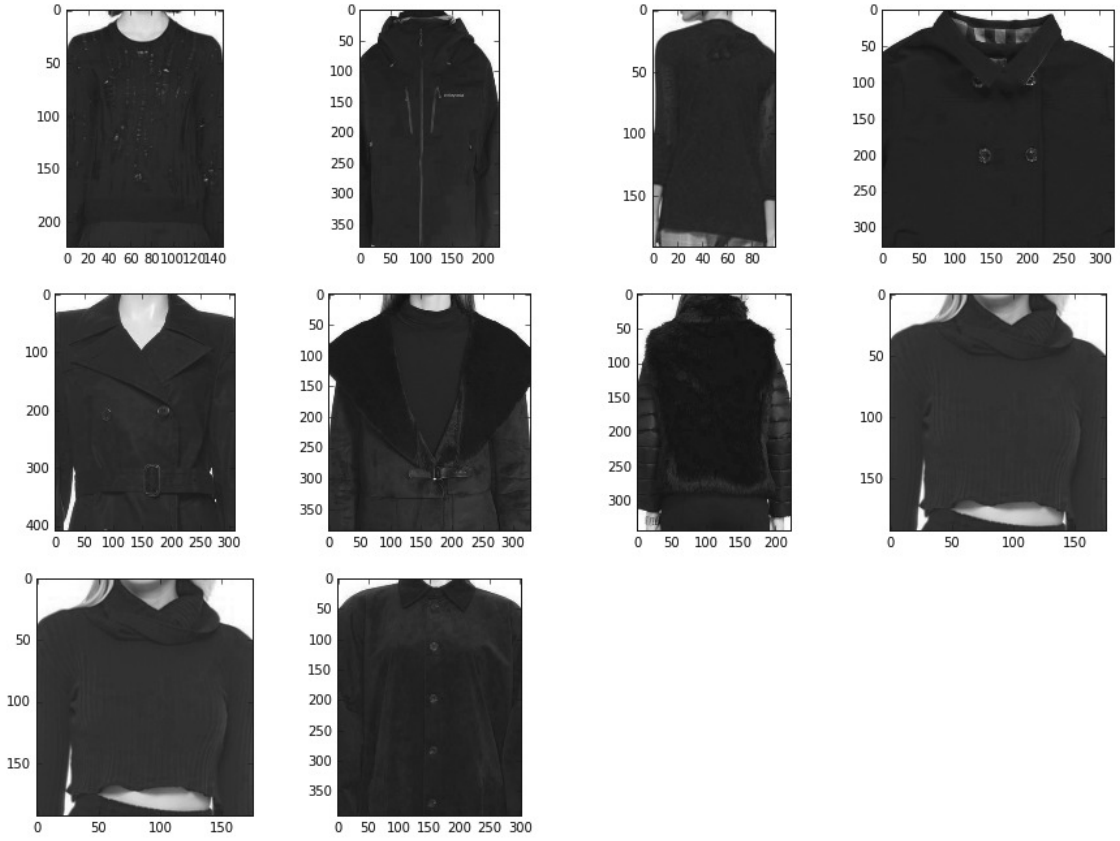

```
Similarity
```



query

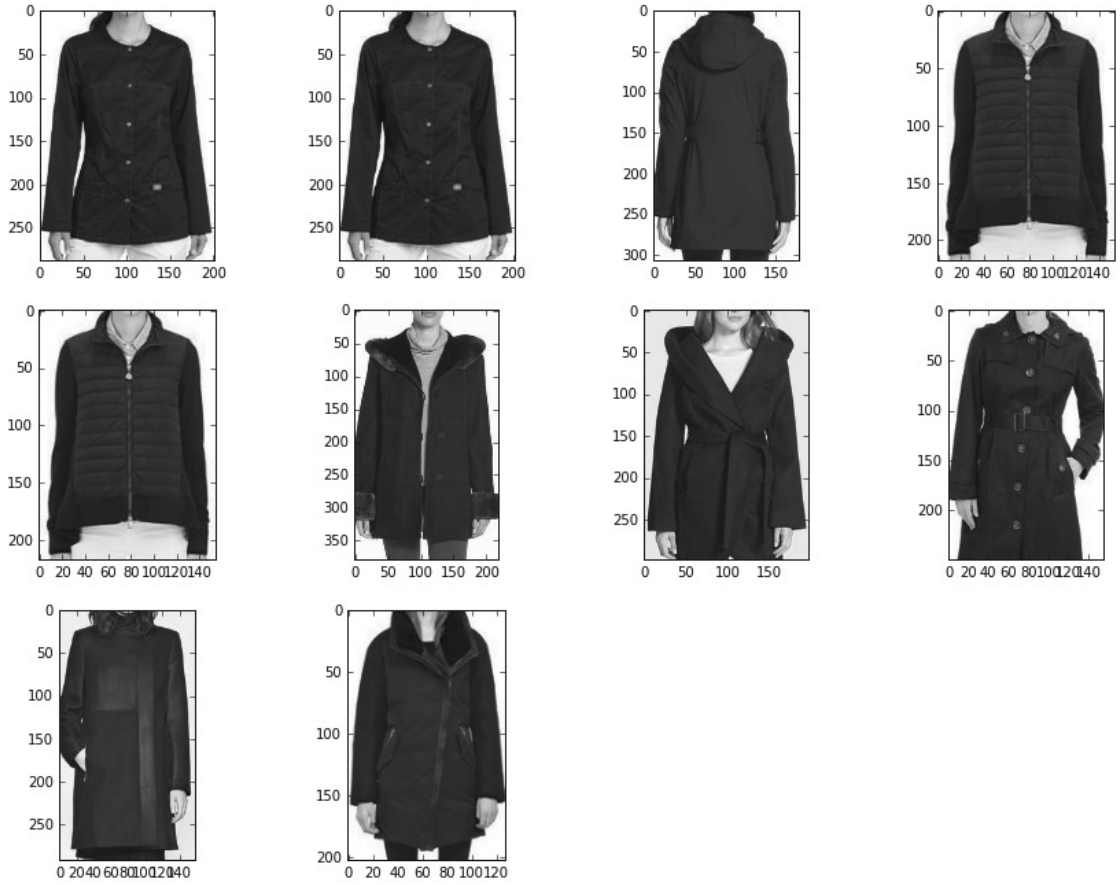


Similarity



query

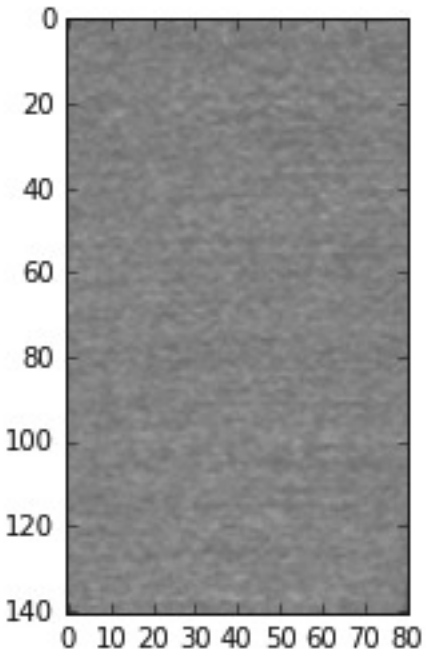
Similarity



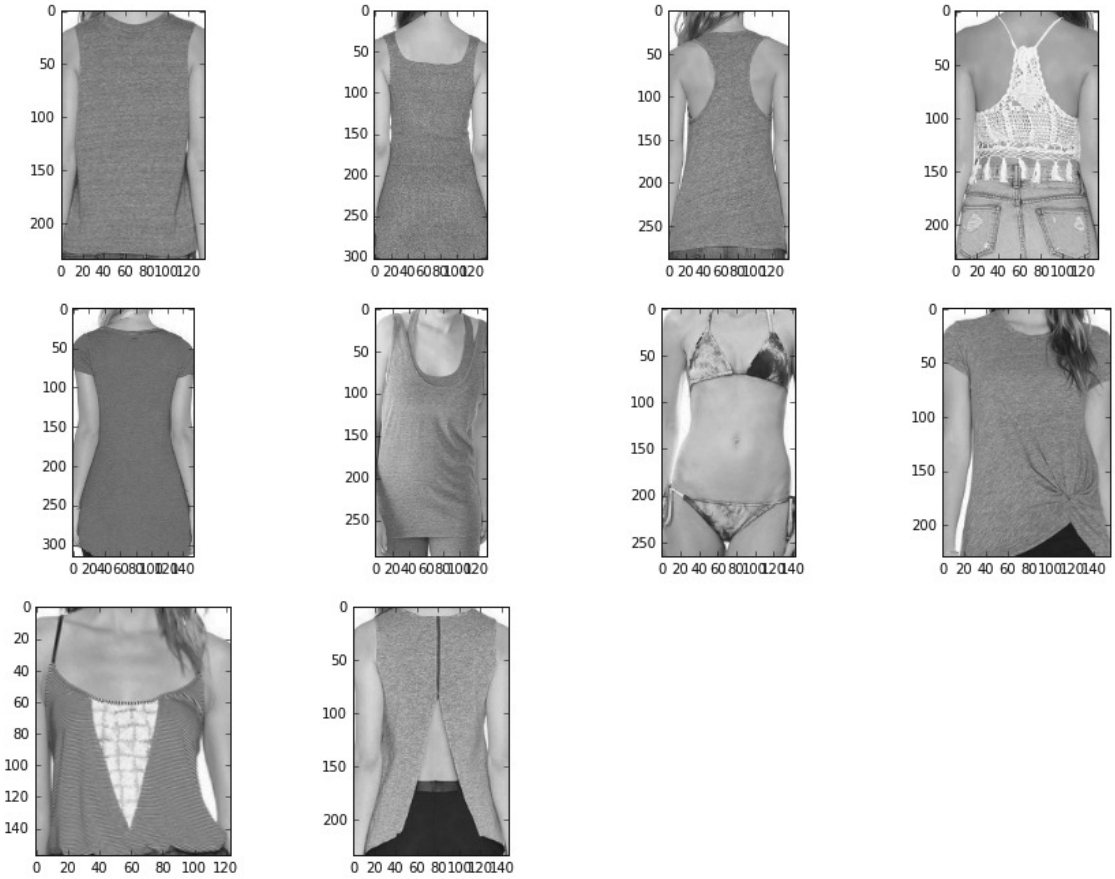
```
In [15]: index =  'index_crop_pool1.csv'

         cd = DeepDescriptor('pool1','VGG')
         # initialize the image descriptor


         for ilen in range(len(Query_list)):
           query = Query_list[ilen];
           image = cv2.imread(query);
           (h, w) = image.shape[:2];
           r = 0.8;
           rh = math.ceil(r*h);
           wh = math.ceil(r*w);
           img = image[math.ceil((1-r)*h):math.ceil(r*h), math.ceil((1-r)*w):math.ceil(r*w)];


           print "query"
         # display the query
           plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB));
           plt.show();
           features = cd.describe(query);

         # perform the search
```

```
  searcher = Searcher(index,'pool1')
  results_pool_1_crop = searcher.search(features)
#
  plt.figure(figsize=(15,15));
  lenf = len(results_pool_1_crop);
  lensq = math.ceil(math.sqrt(lenf));
  resultidx = 1;
  print "Similarity"
# loop over the results
  for (score, resultID) in results_pool_1_crop:
        # load the result image and display it
    result = cv2.imread(result_path + "/" + resultID);
    plt.subplot(lensq,lensq,resultidx);
    plt.imshow(cv2.cvtColor(result,cv2.COLOR_BGR2RGB));
    resultidx += 1;
  plt.show();
```

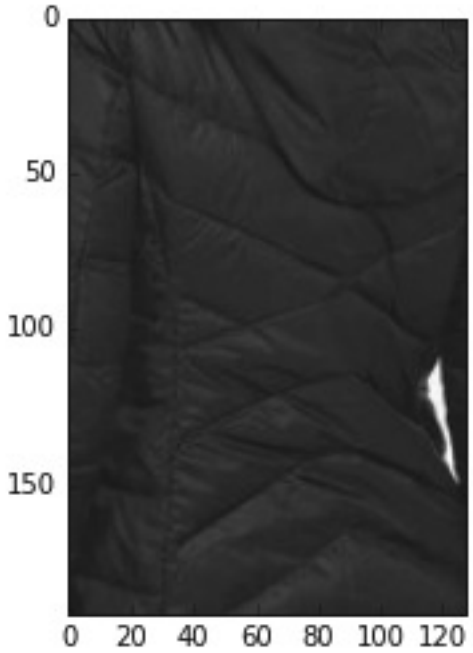
query



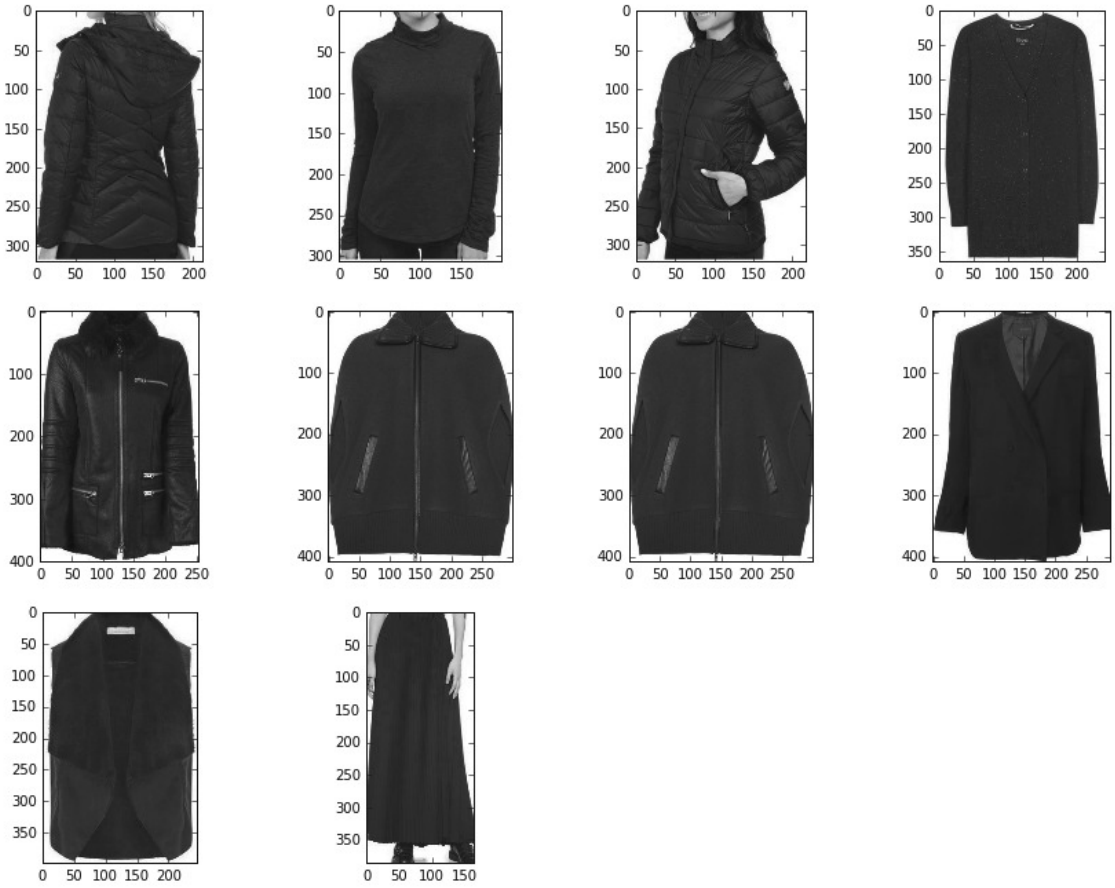
Similarity



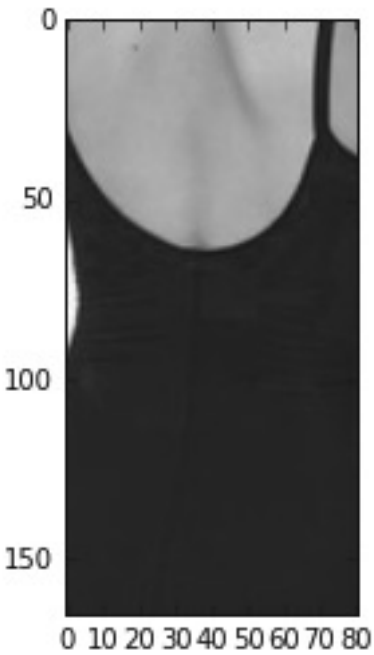
query



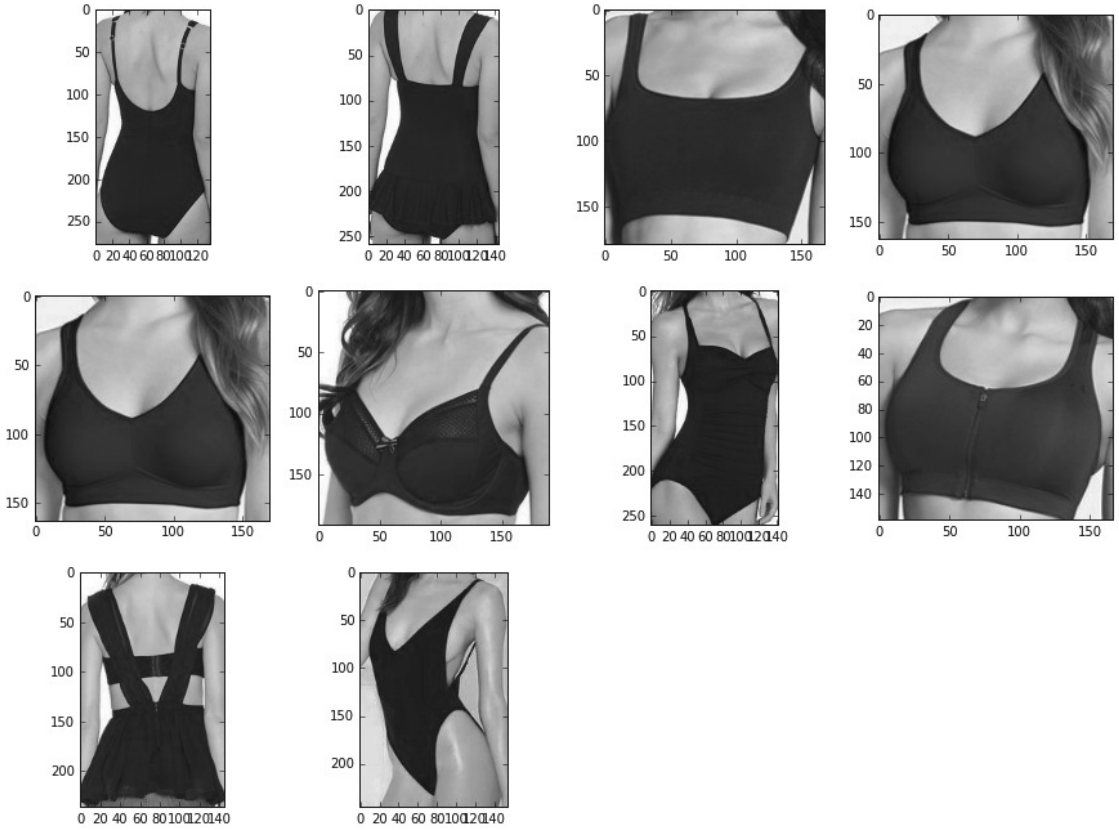
Similarity



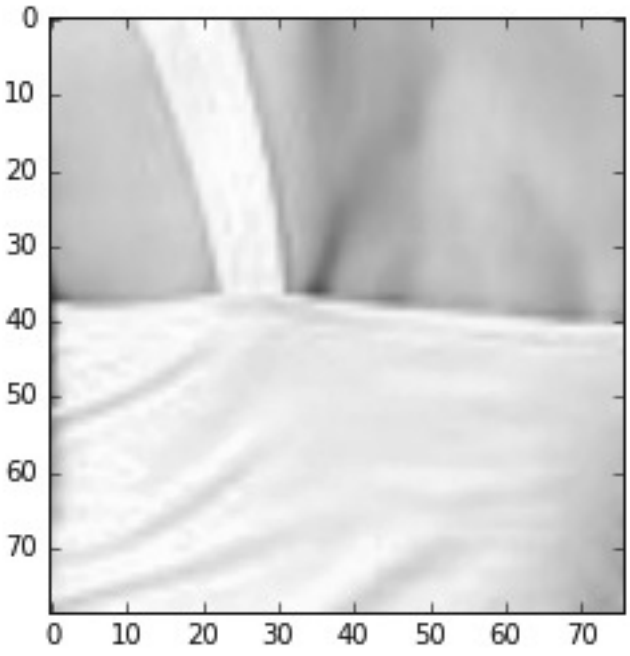
query



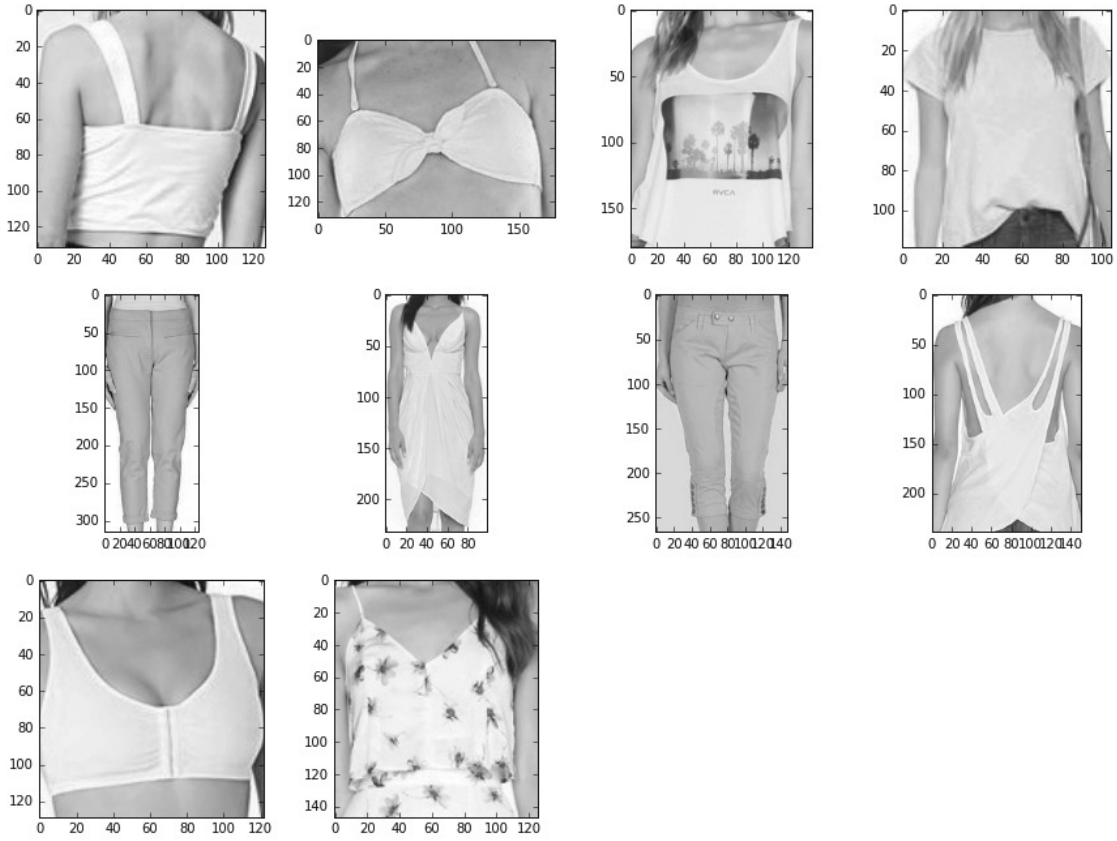
Similarity



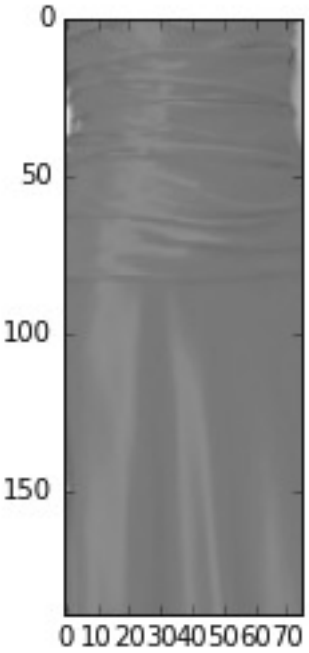
query



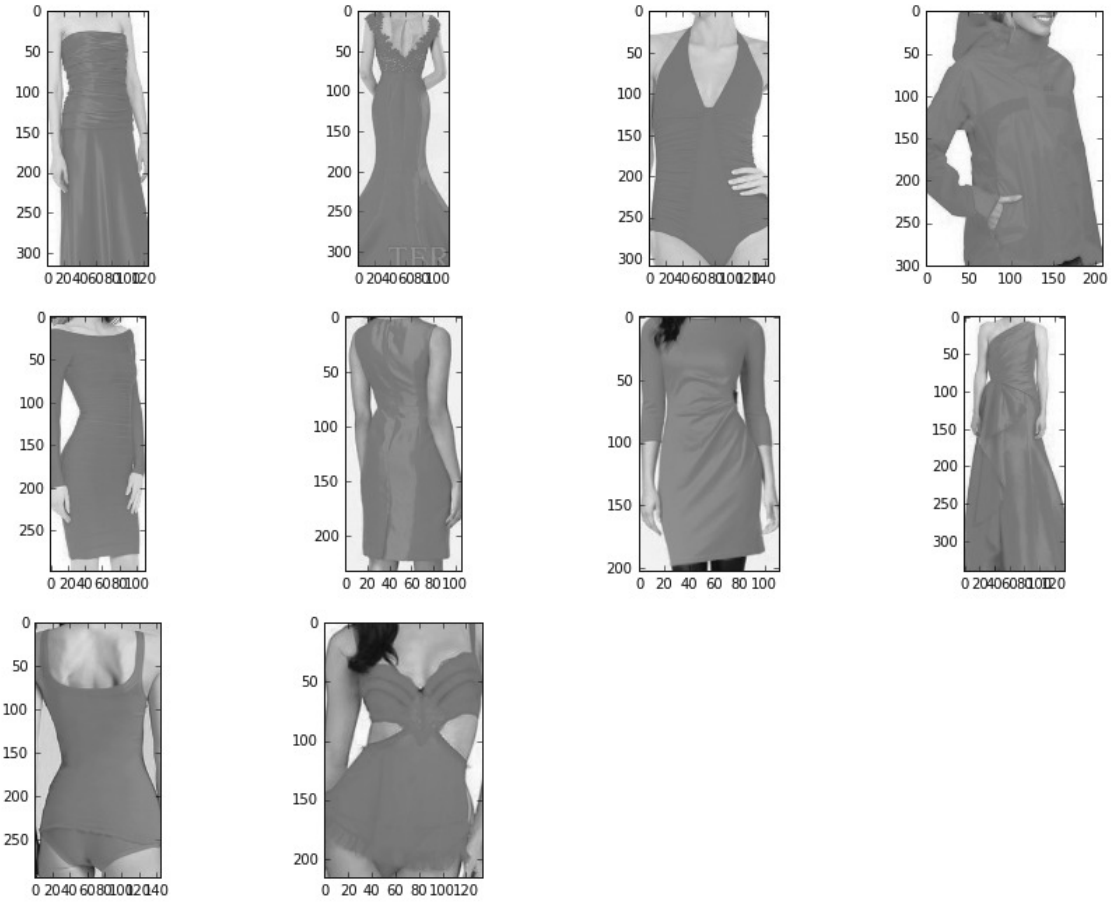
Similarity



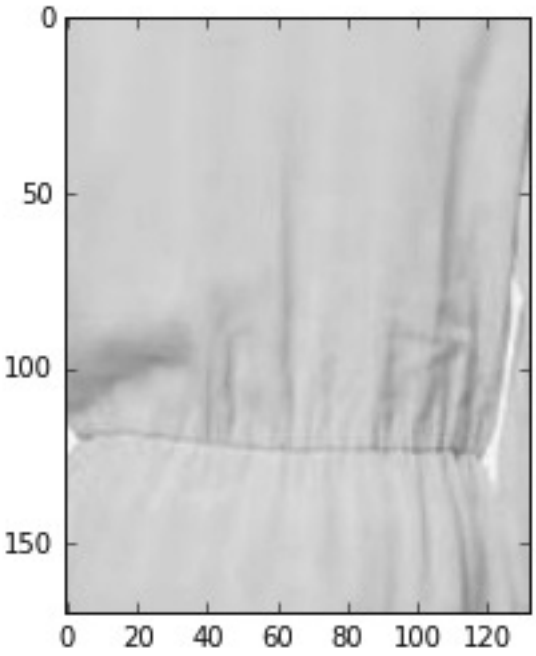
query



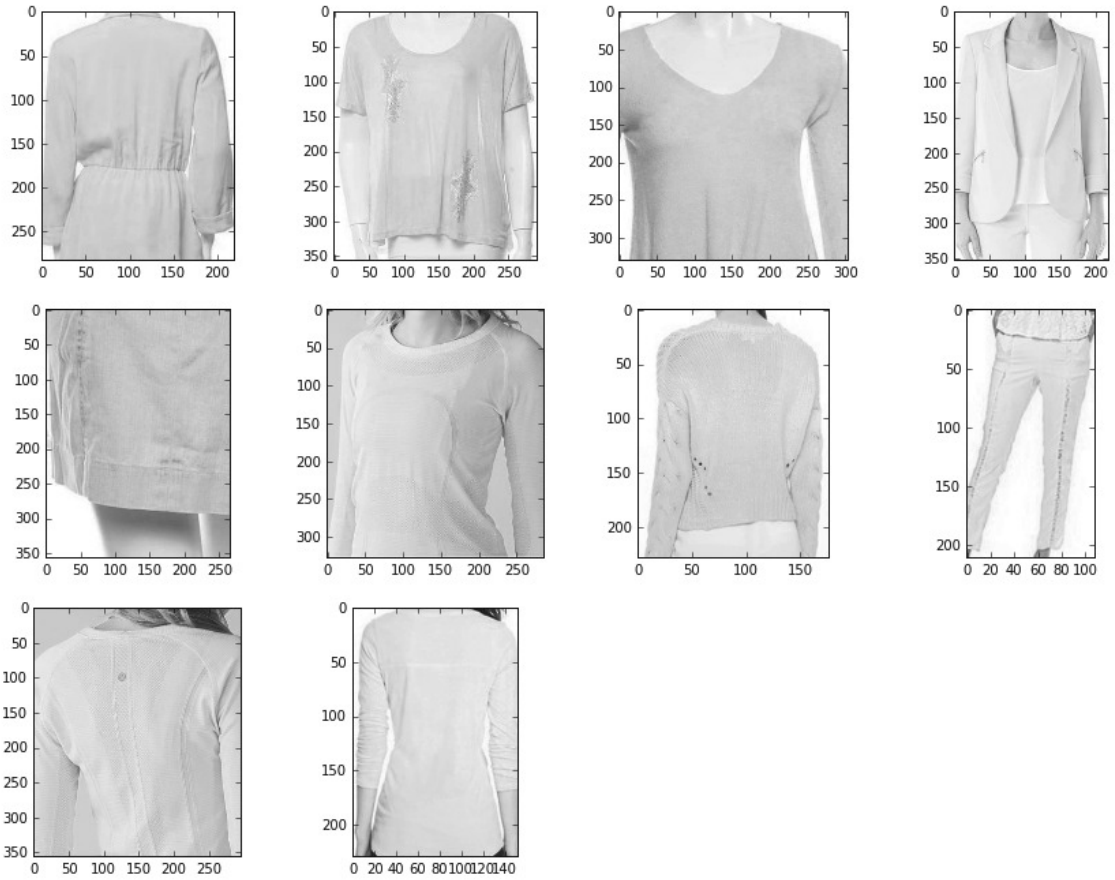
Similarity



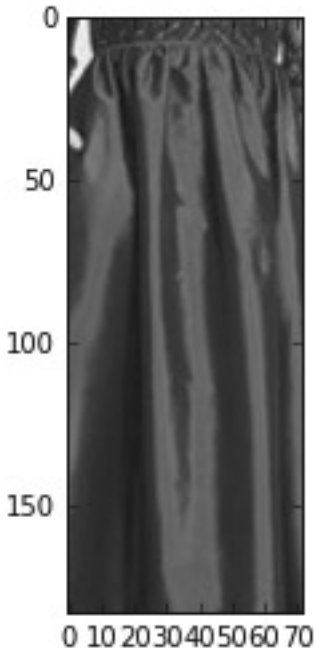
query



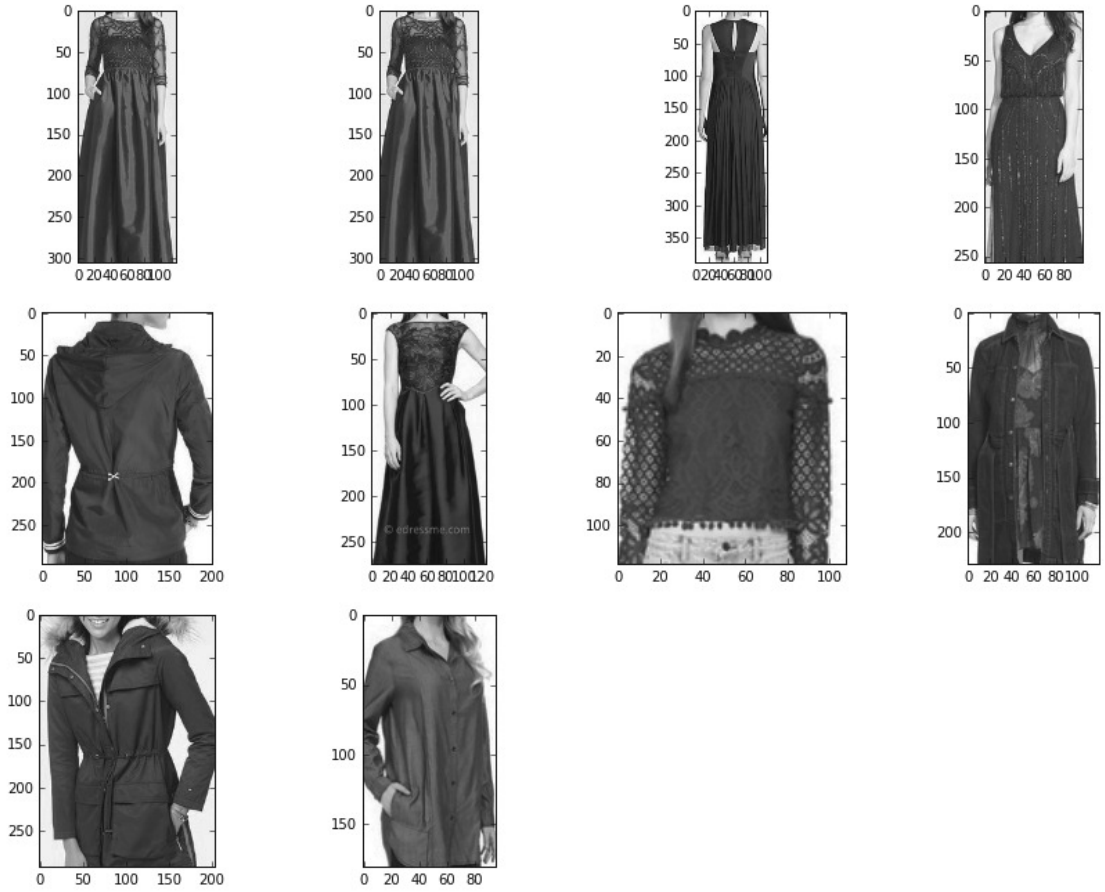
Similarity



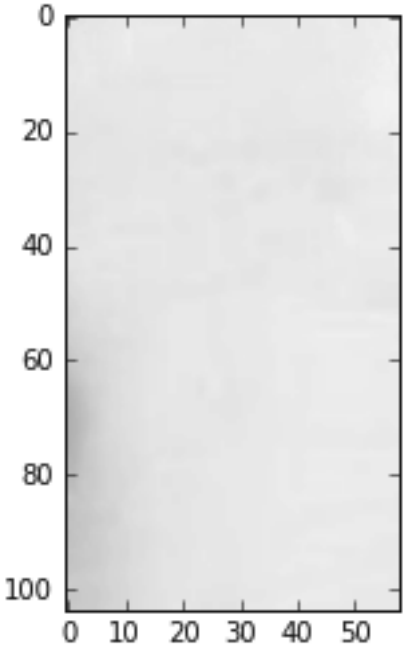
query



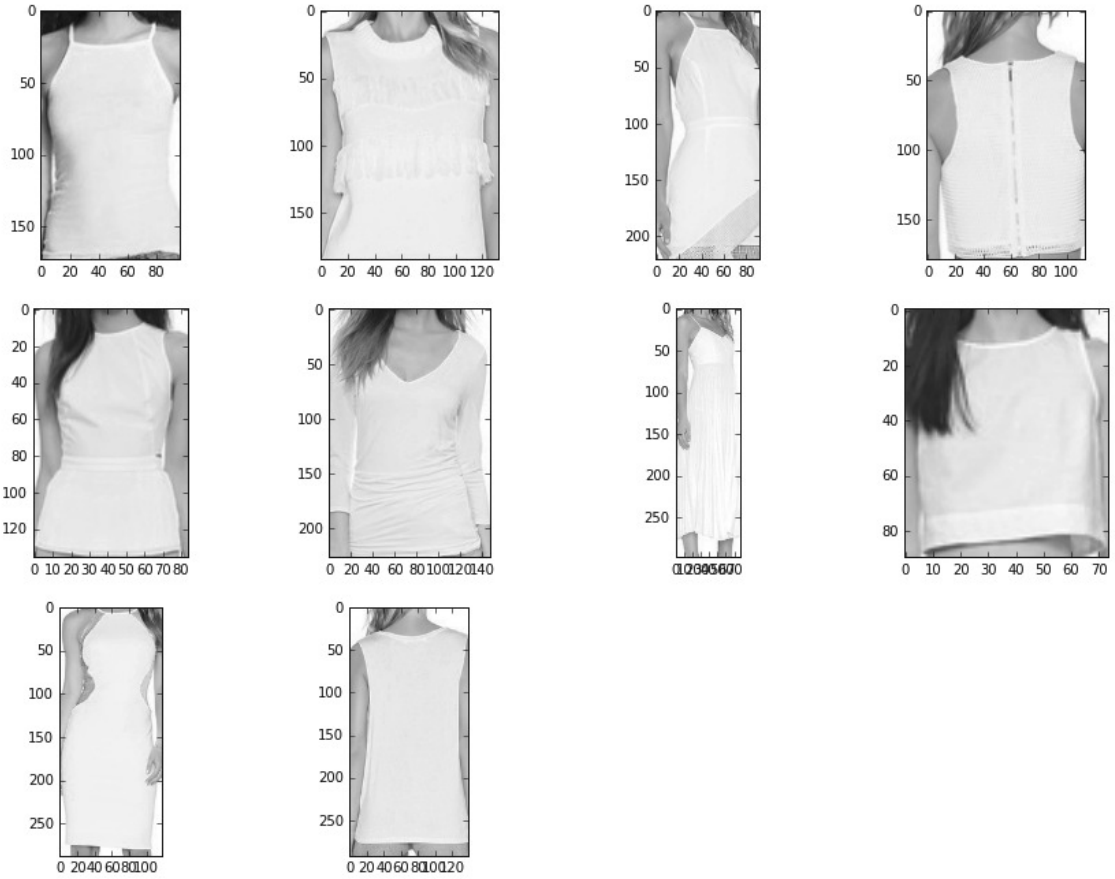
Similarity



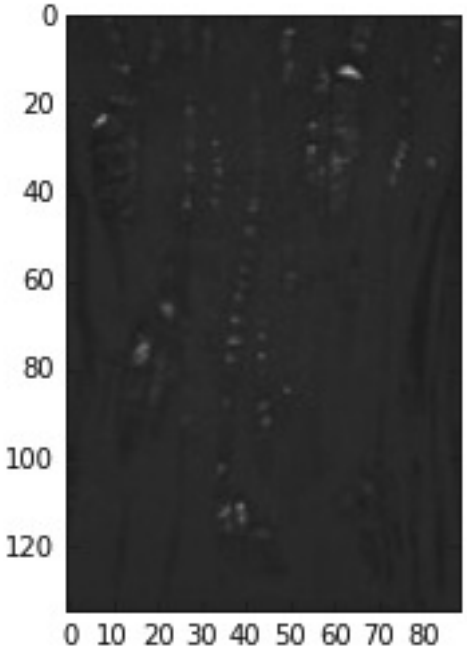
query



Similarity



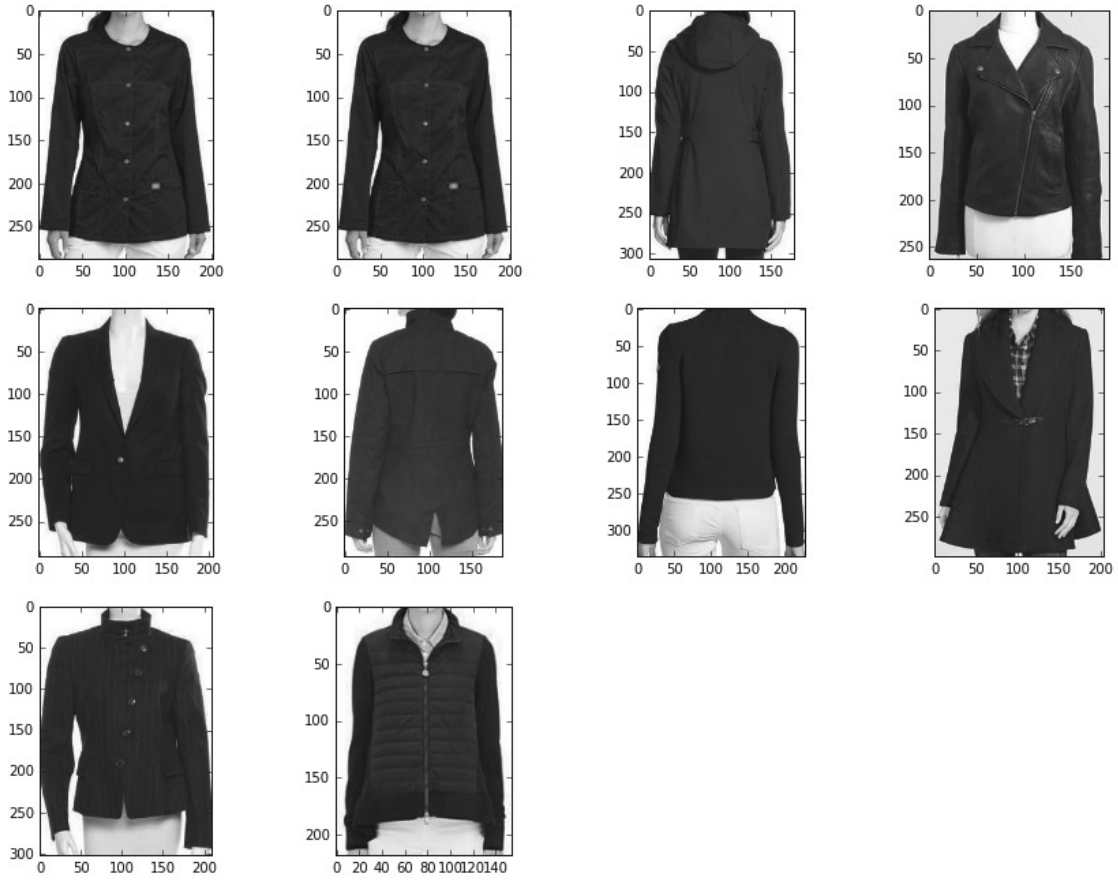
query



Similarity

query

Similarity

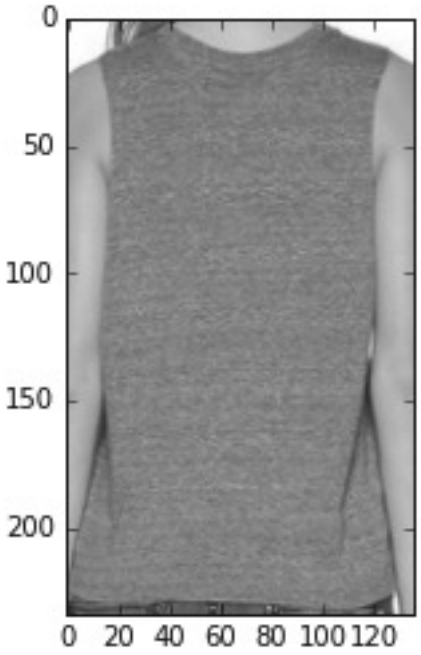


2.1 Using Pool 1 + Pool 2

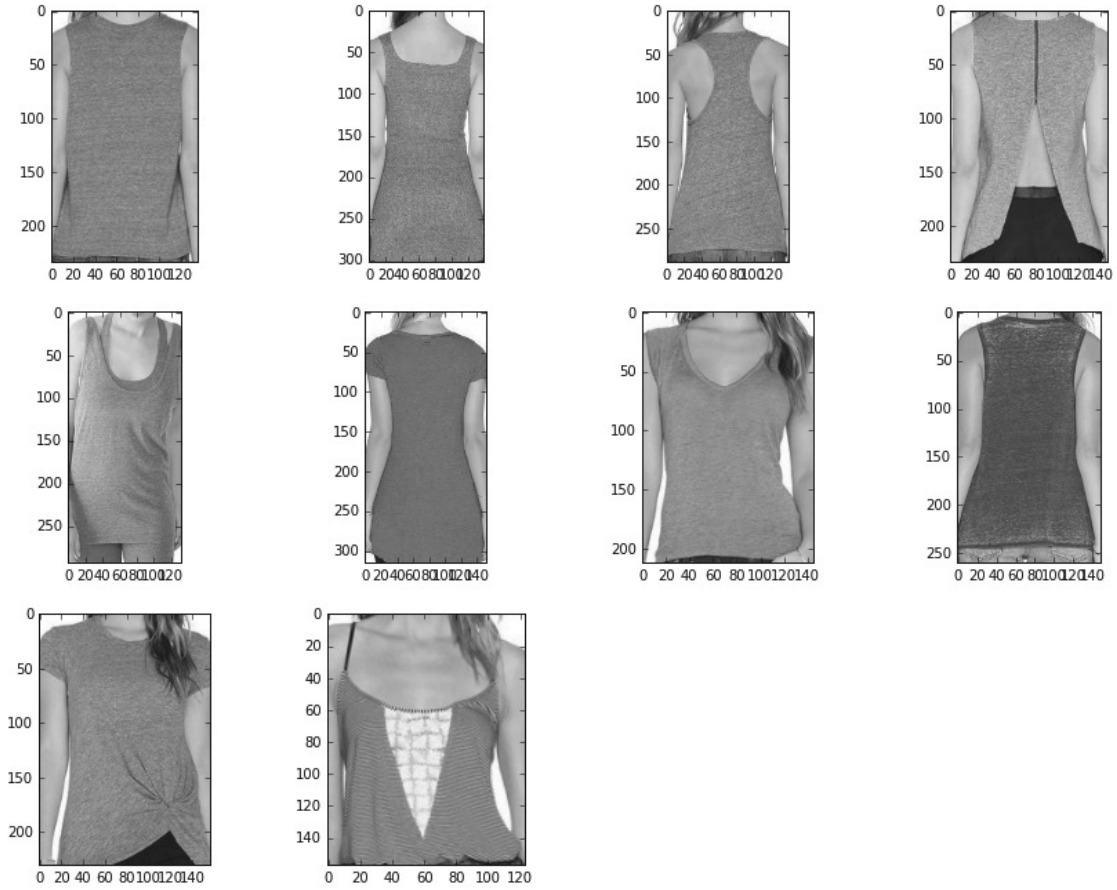
```
In [16]: index =  'index_pool_1_pool_2.csv'
         # initialize the image descriptor
         cd = DeepDescriptor('pool1+pool2','VGG')

         for ilen in range(len(Query_list)):
           query = Query_list[ilen];
           img = cv2.imread(query);
           print "Query"
         # display the query
           plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB));
           plt.show();
           features = cd.describe(query);


         # perform the search
           searcher = Searcher(index,'pool1+pool2');
           results_pool_1 = searcher.search(features)
         #
           plt.figure(figsize=(15,15));
           lenf = len(results_pool_1);
           lensq = math.ceil(math.sqrt(lenf));
```

```
  resultidx = 1;
  print "Similarity"
# loop over the results
  for (score, resultID) in results_pool_1:
        # load the result image and display it
    result = cv2.imread(result_path + "/" + resultID);
    plt.subplot(lensq,lensq,resultidx);
    plt.imshow(cv2.cvtColor(result,cv2.COLOR_BGR2RGB));
    resultidx += 1;
  plt.show();
#        cv2.waitKey(0)
```

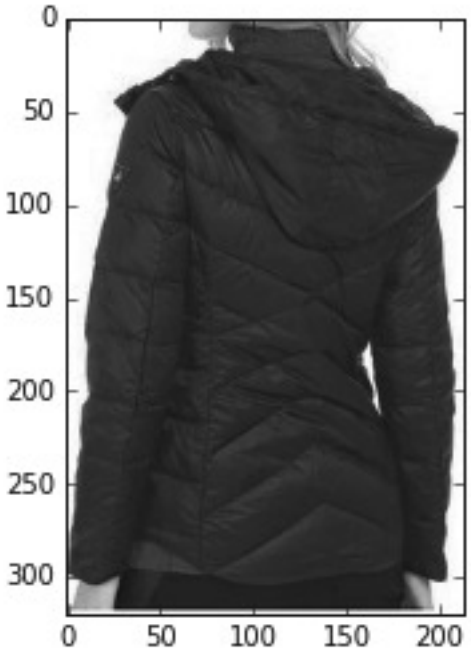
```
Query
```




```
(128, 128)
Similarity
```

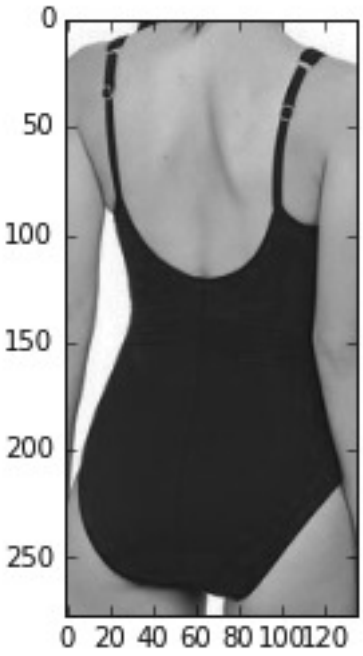


Query

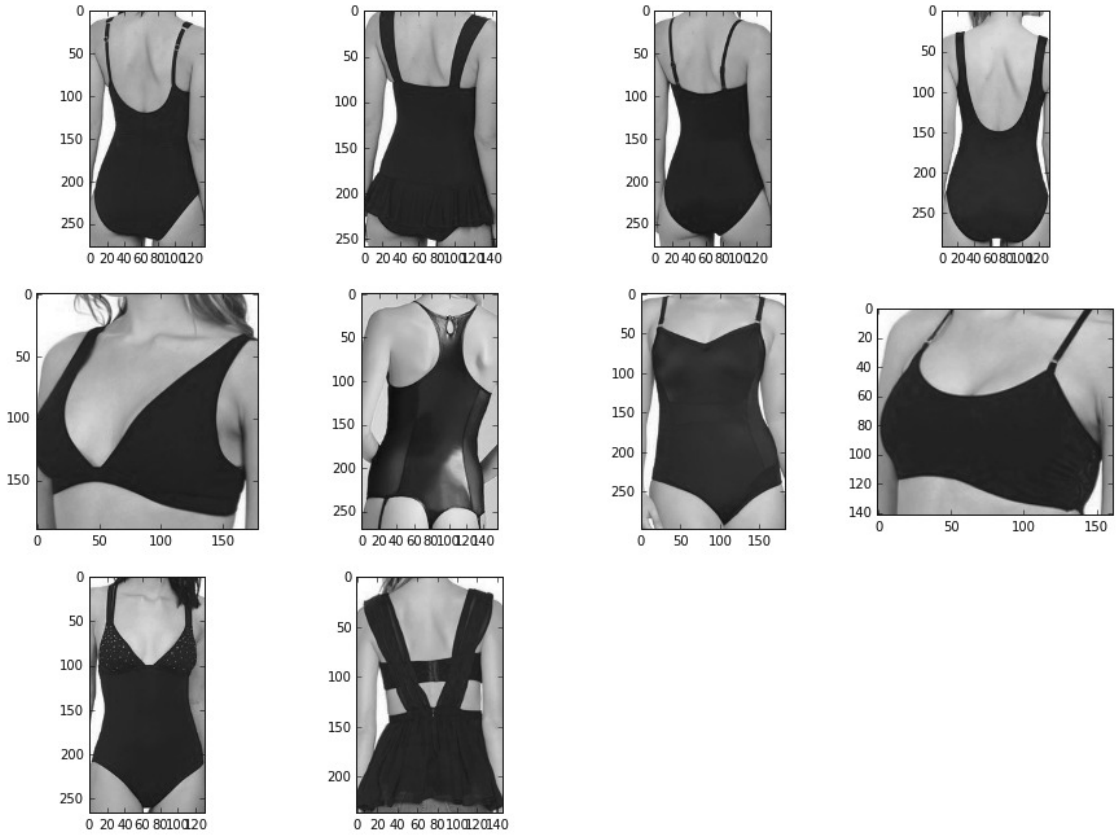


(128, 128)
Similarity

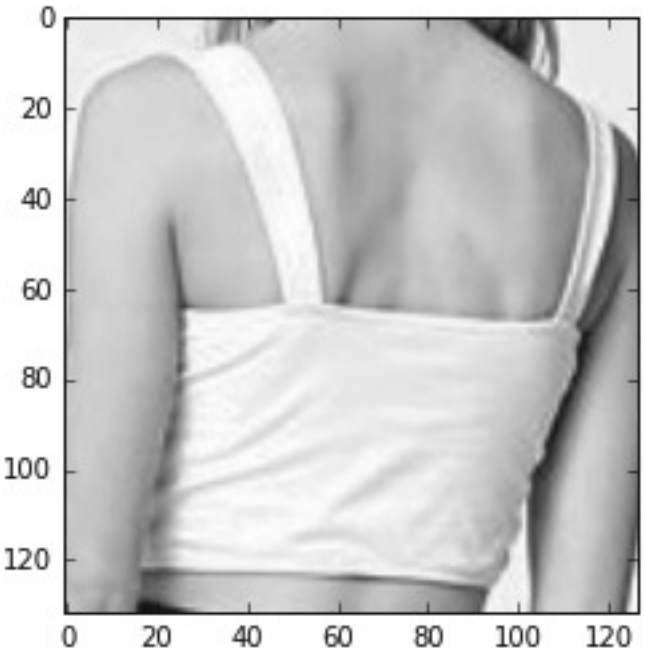
Query



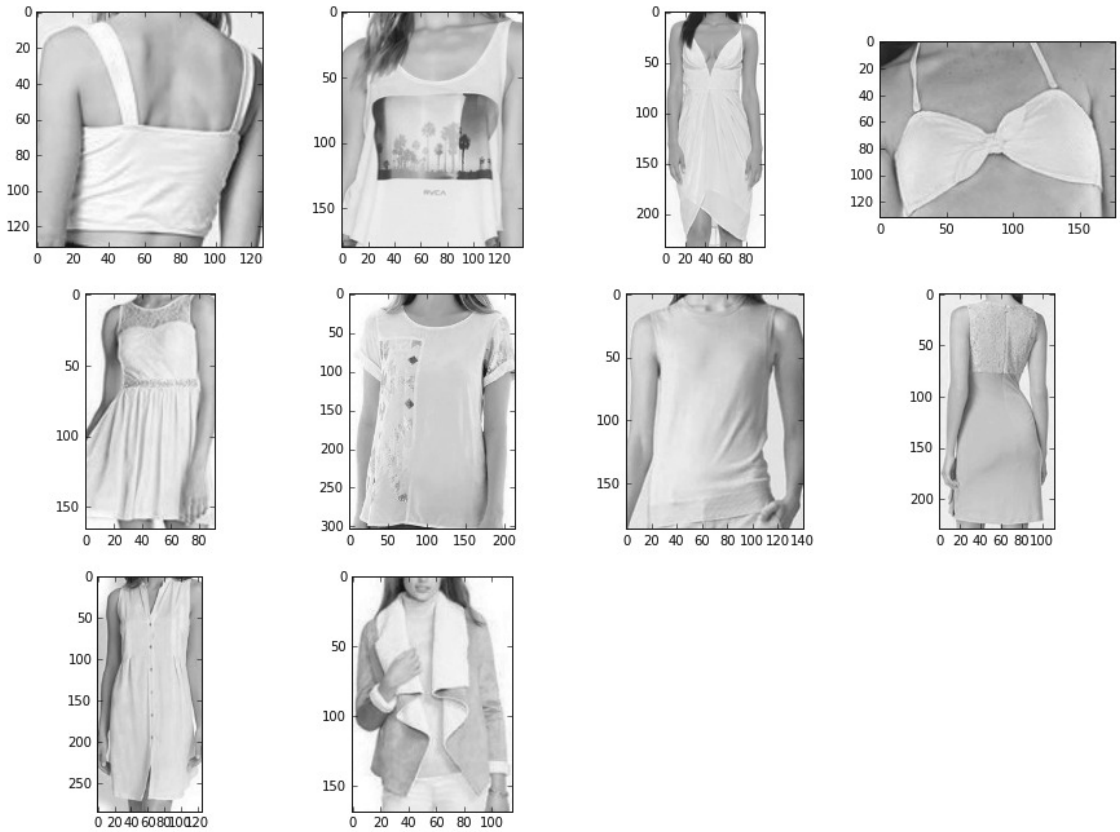
(128, 128)
Similarity



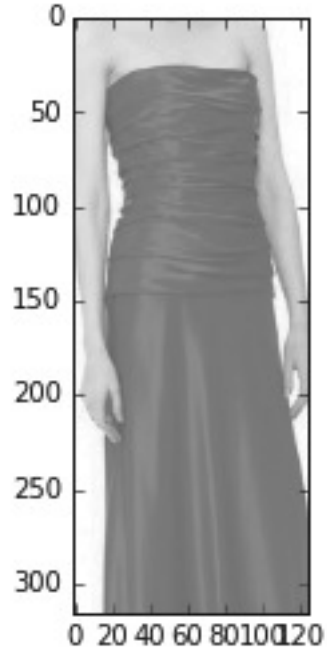
Query



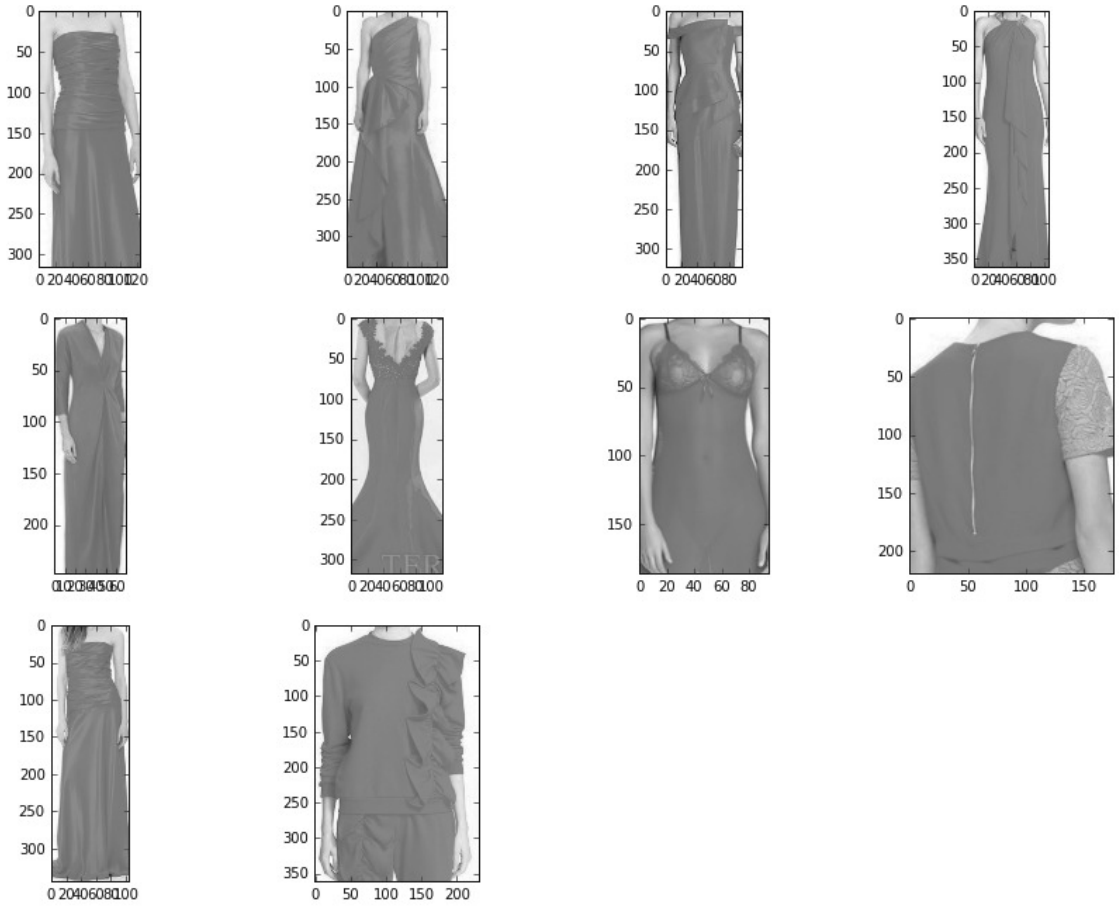
(128, 128)
Similarity



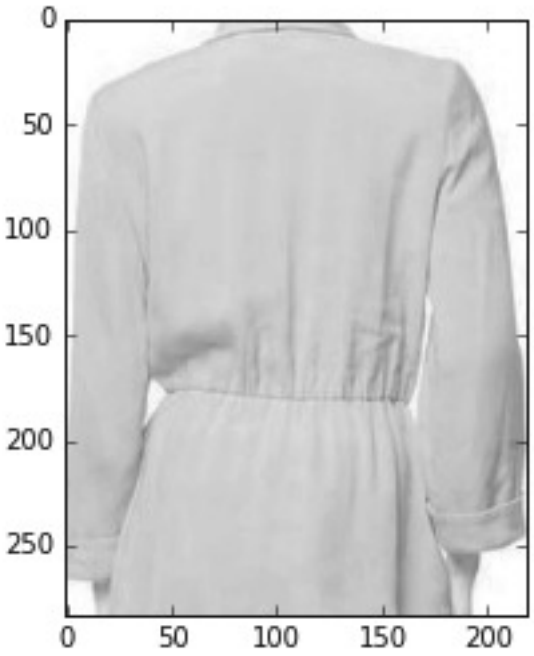
Query



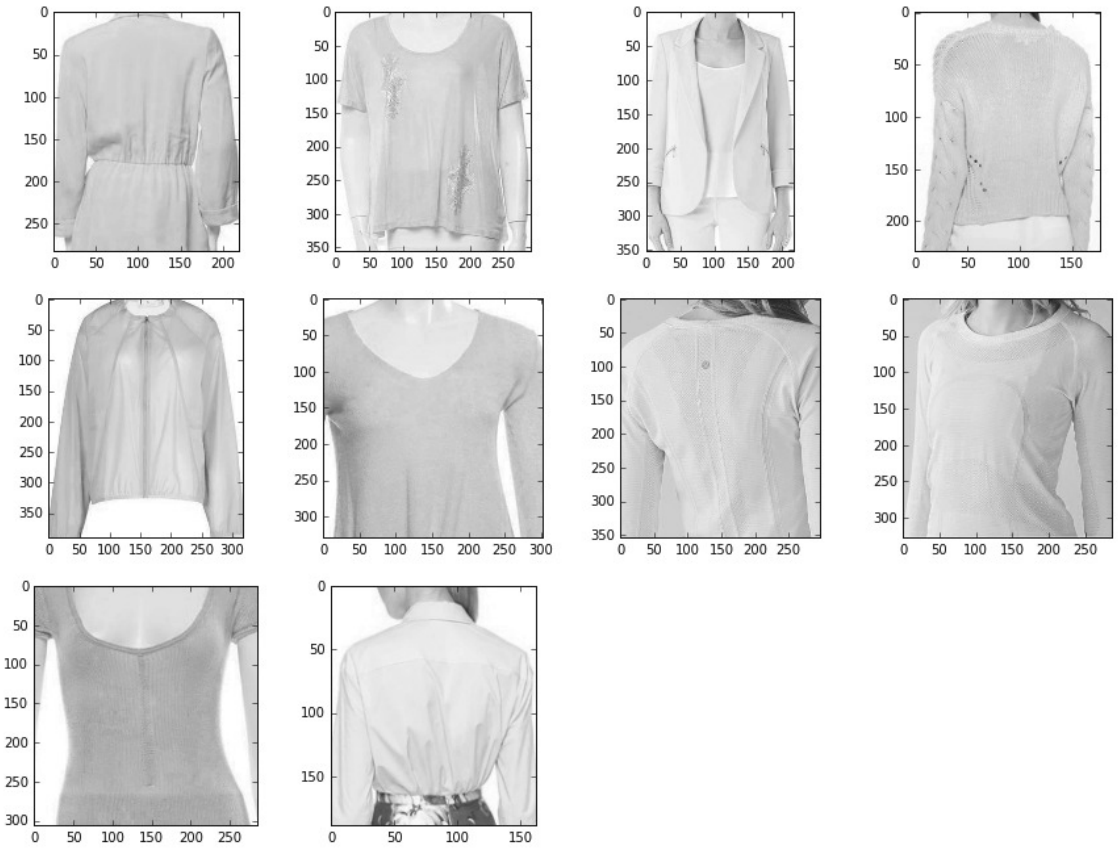
(128, 128)
Similarity



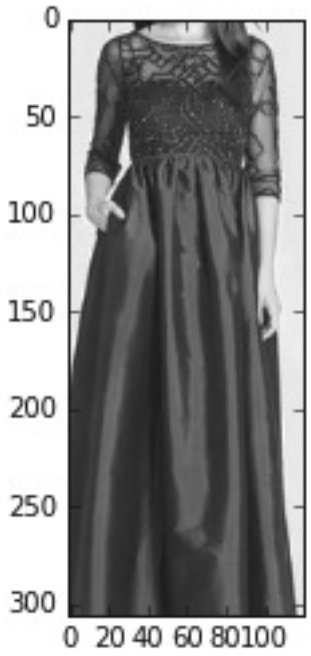
Query



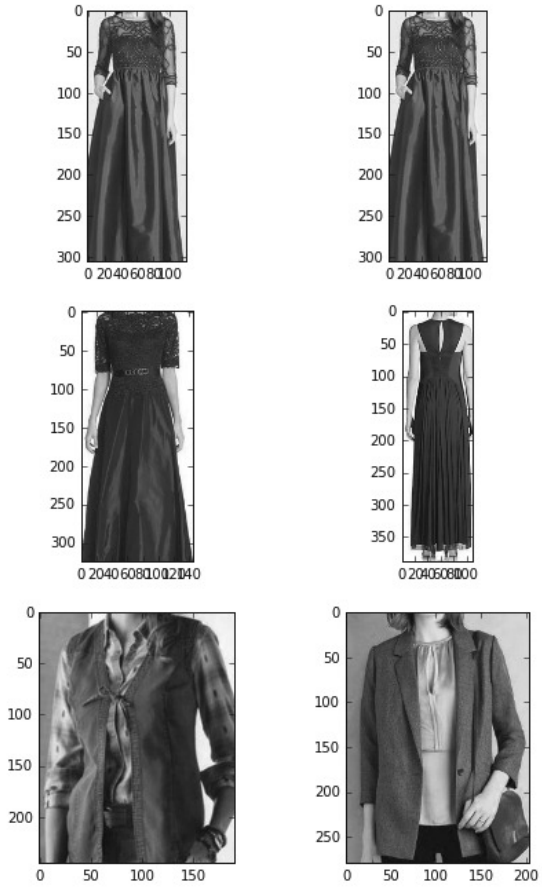
```
(128, 128)
Similarity
```



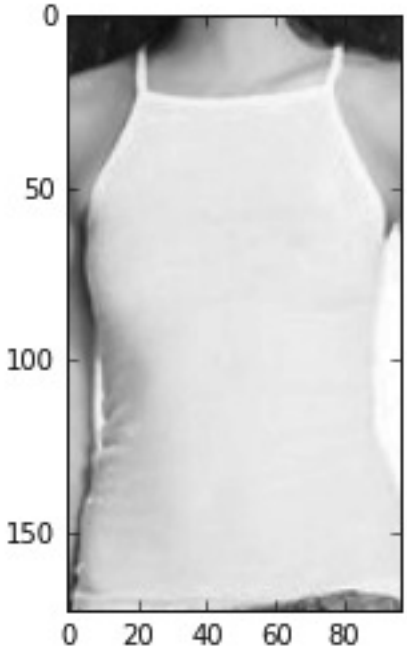
Query



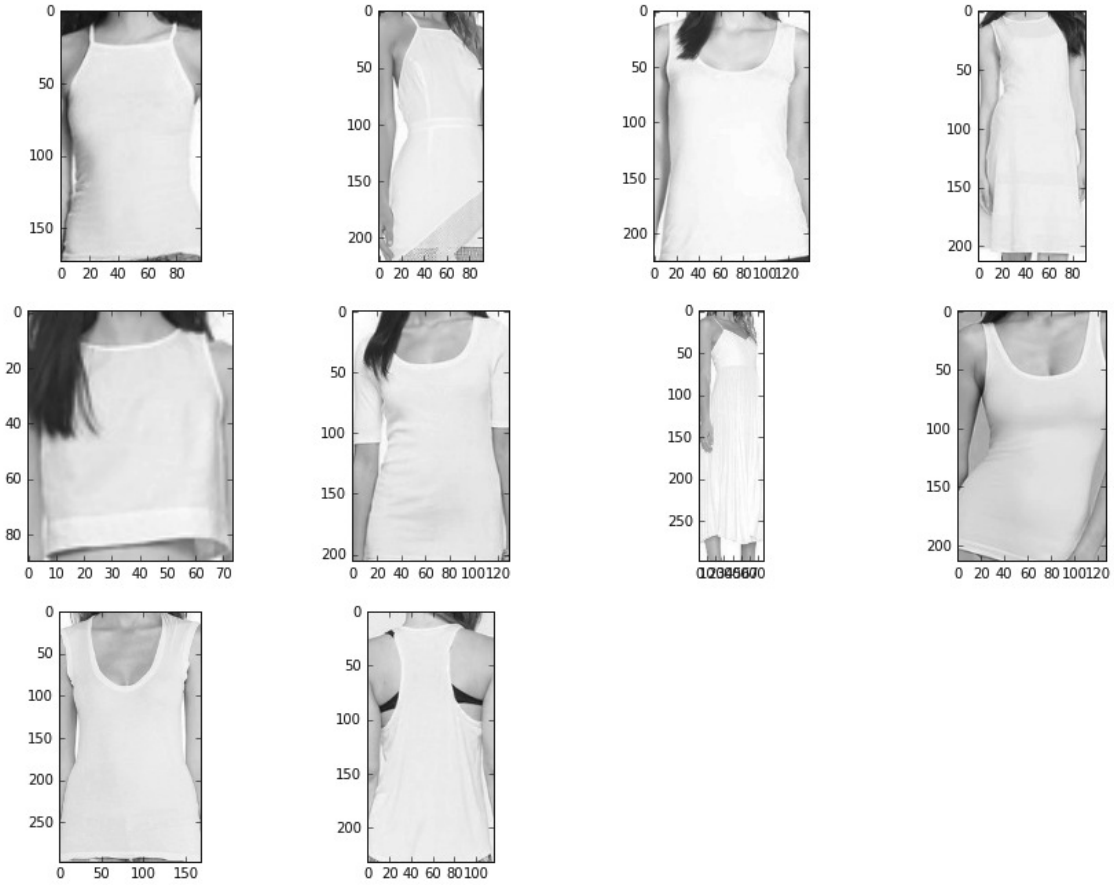
(128, 128)
Similarity



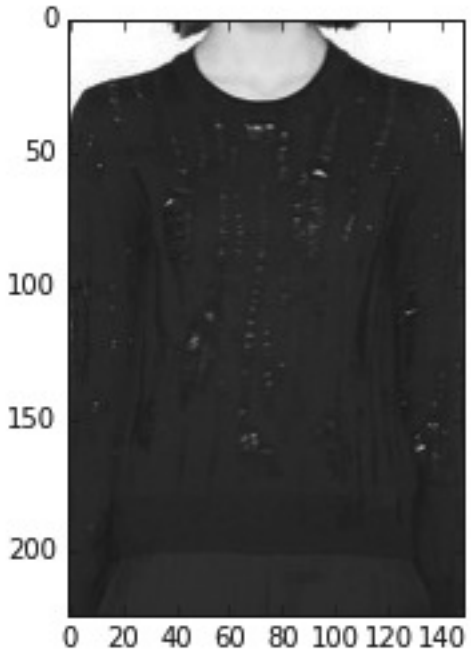
Query



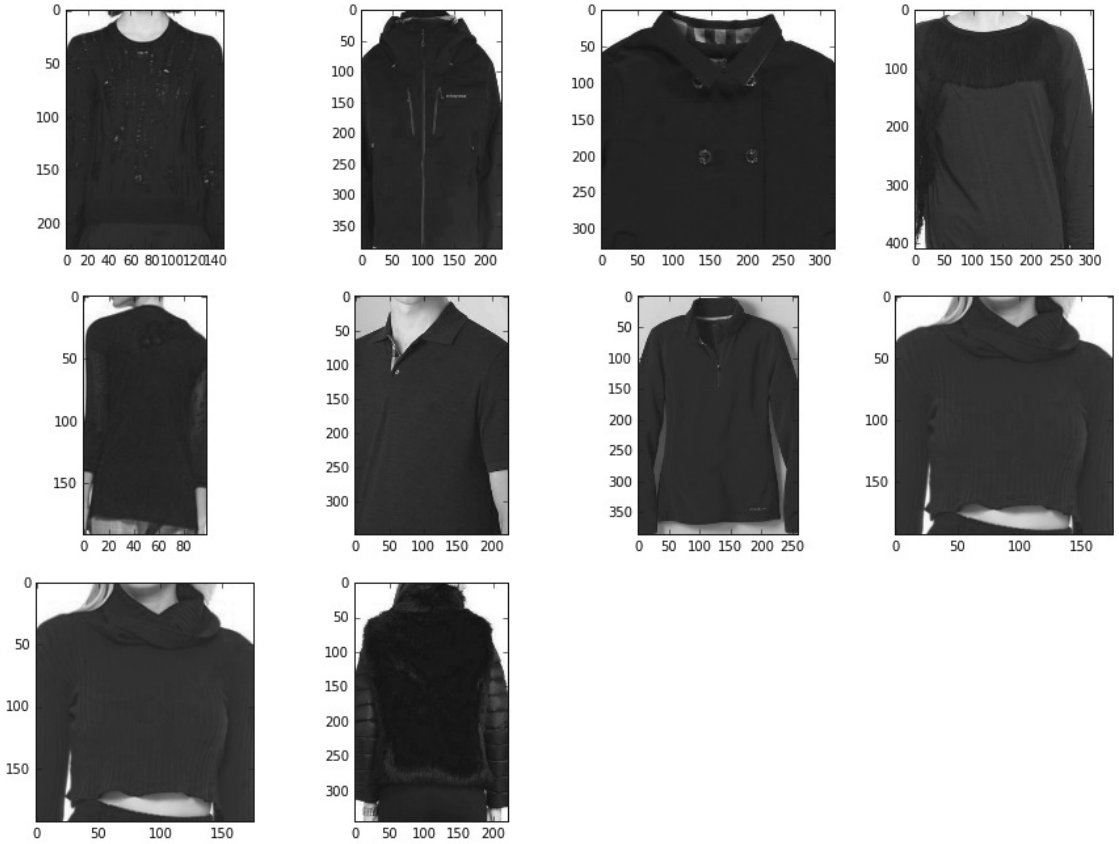
(128, 128)
Similarity



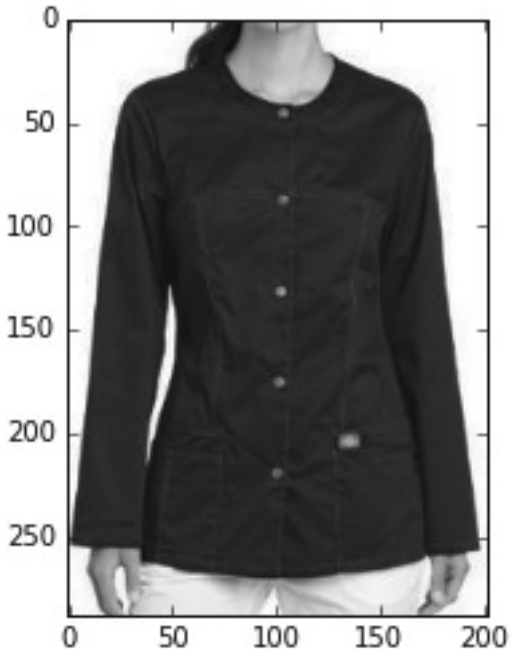
Query



(128, 128)
Similarity



Query



(128, 128)
Similarity

In [16]:

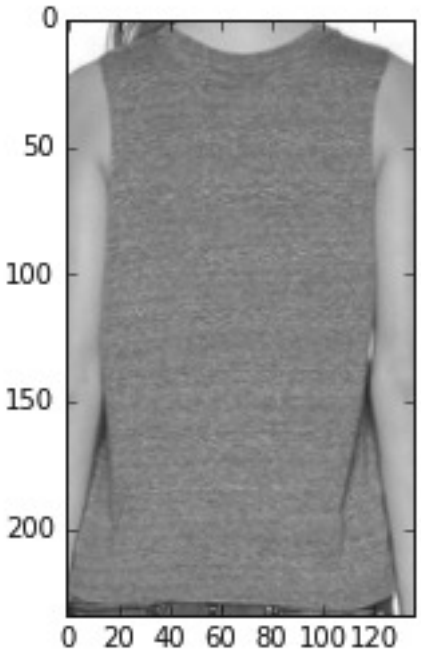
In [17]:
```
# import the necessary packages
from DeepDescriptor import DeepDescriptor
from searcher import Searcher
import argparse
import cv2
import os, random
import matplotlib.pyplot as plt
import math;

%matplotlib inline
# construct the argument parser and parse the arguments
result_path = '../../fashion_train';

index =  'inception_3a.csv'
 # initialize the image descriptor
cd = DeepDescriptor('inception_3a/output','GoogleNet')
```

GoogleNet

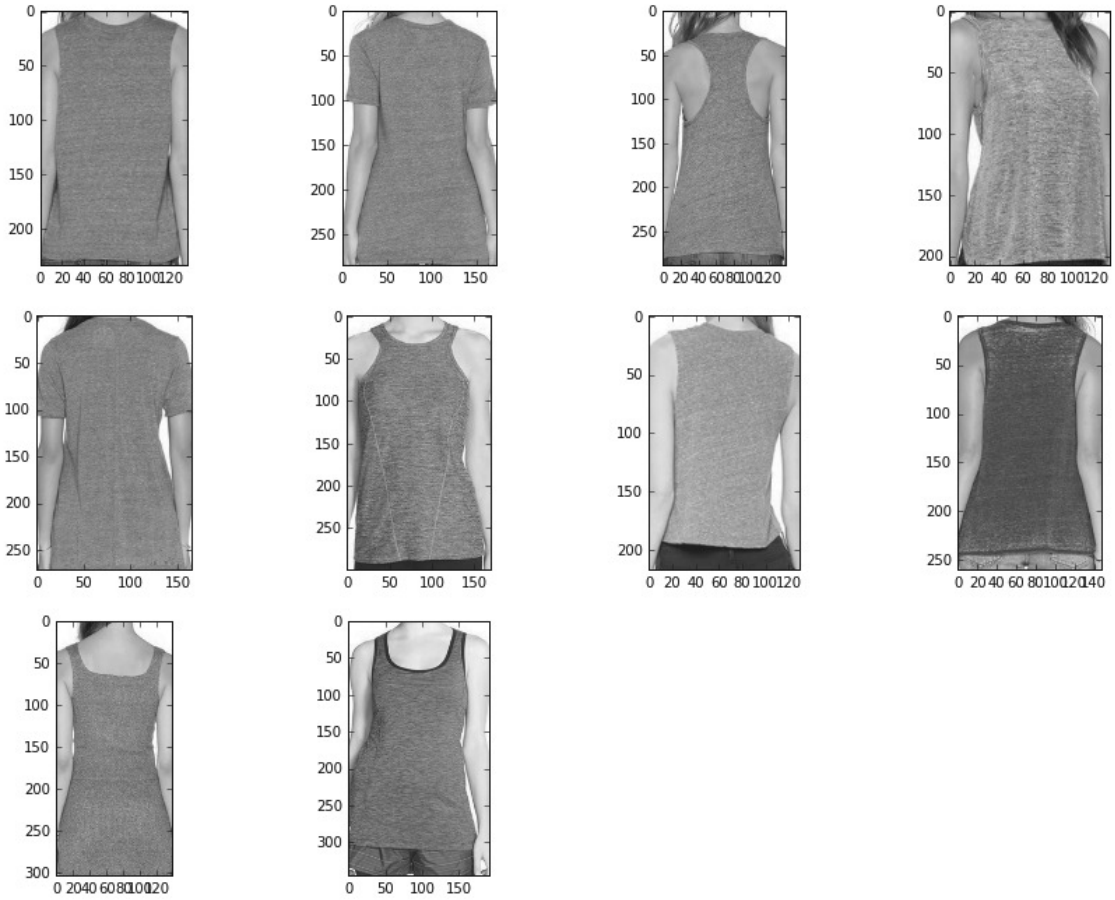
In [18]:
```
for ilen in range(len(Query_list)):
```

```
  query = Query_list[ilen];
  img = cv2.imread(query);
  print "Query"
# display the query
  plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB));
  plt.show();
  features = cd.describe(query);

# perform the search
  searcher = Searcher(index,"inception_3a/output")
  results_pool_2 = searcher.search(features)
#
  plt.figure(figsize=(15,15));
  lenf = len( results_pool_2);
  lensq = math.ceil(math.sqrt(lenf));
  resultidx = 1;
  #print  results_pool_2
  print "Similarity"
# loop over the results
  for (score, resultID) in results_pool_2:
        # load the result image and display it
    result = cv2.imread(result_path + "/" + resultID);
    plt.subplot(lensq,lensq,resultidx);
    plt.imshow(cv2.cvtColor(result,cv2.COLOR_BGR2RGB));
    resultidx += 1;
  plt.show();
#       cv2.waitKey(0)Query_list = Query_list[:10]
plt.figure(figsize=(15,15));
lensq = math.ceil(math.sqrt(nQuery_images));
qidx = 1;
for ilen in range(len(Query_list)):
  query = Query_list[ilen];
  img = cv2.imread(query);
 # print query;
  plt.subplot(lensq,lensq,qidx);
  plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB));
  qidx += 1;

plt.show();
```

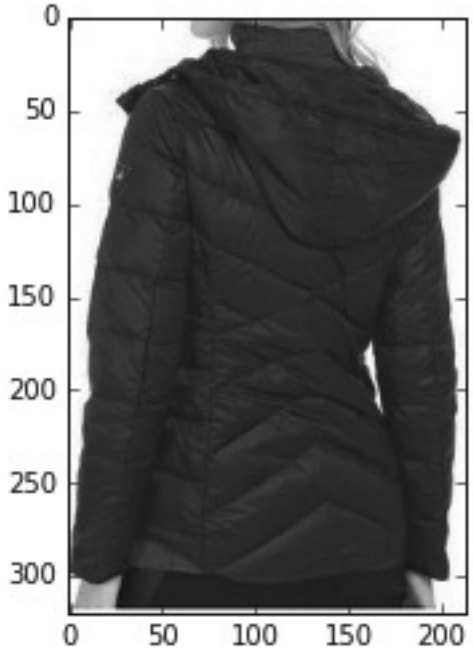
```
Query
```



Similarity

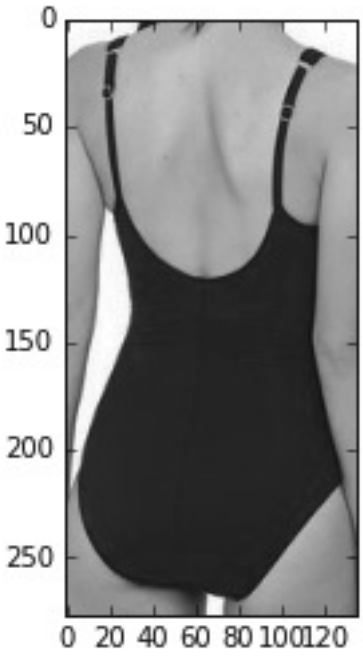


Query

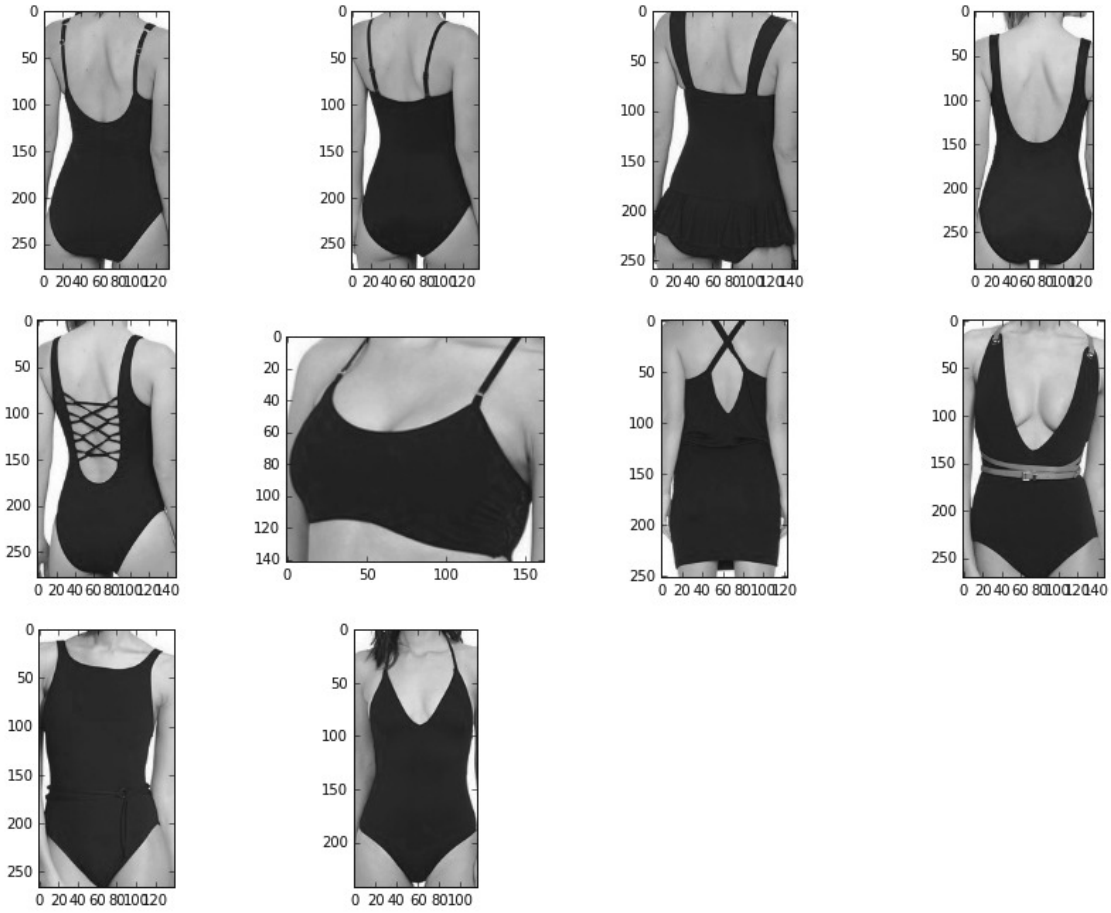


Similarity

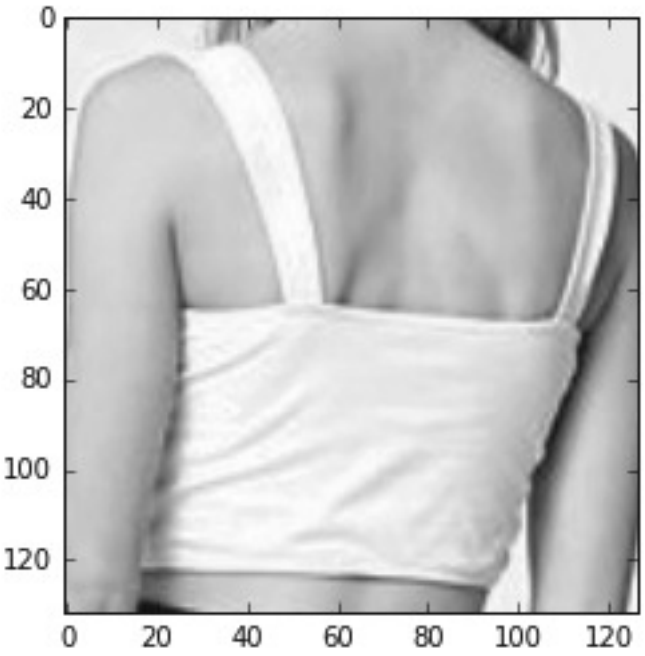
Query



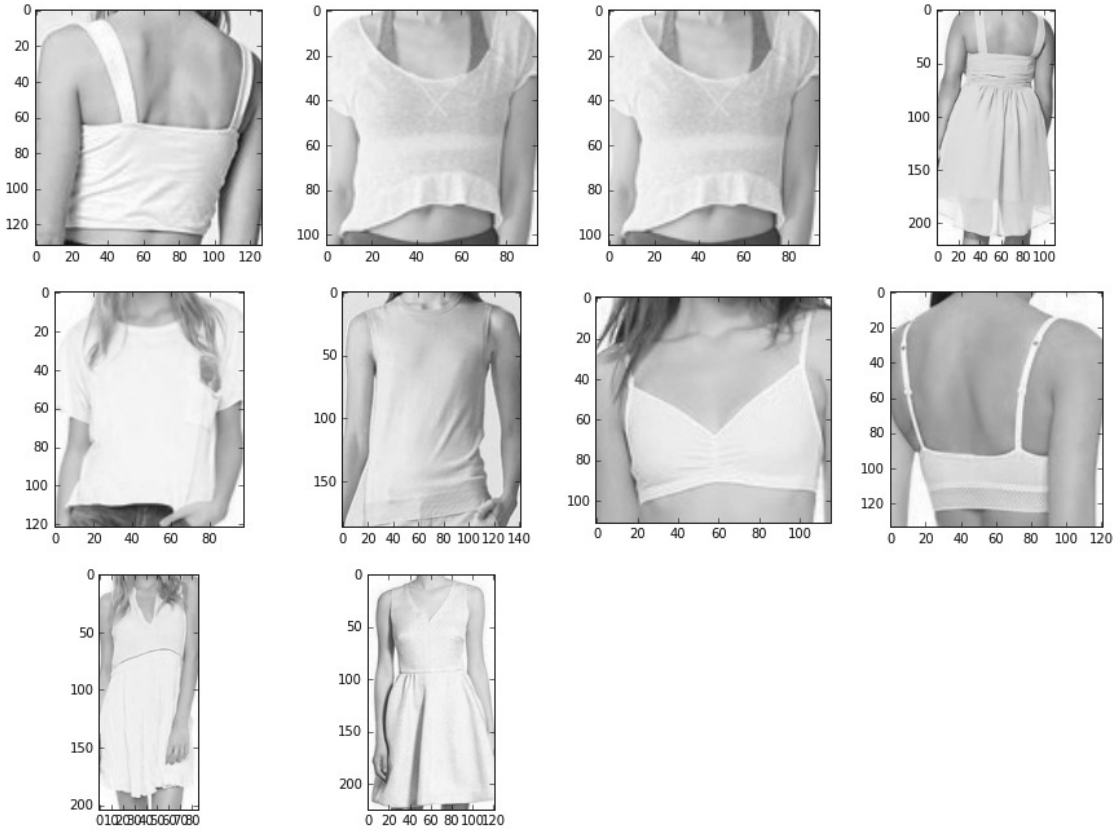
Similarity



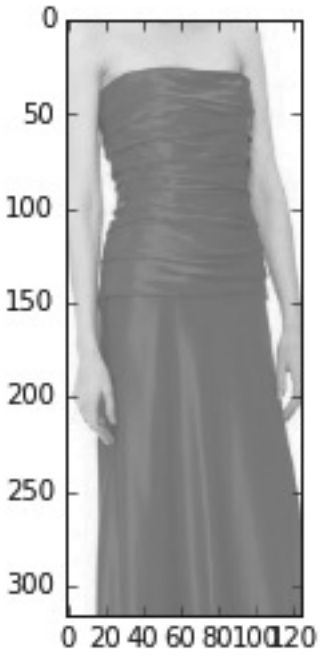
Query



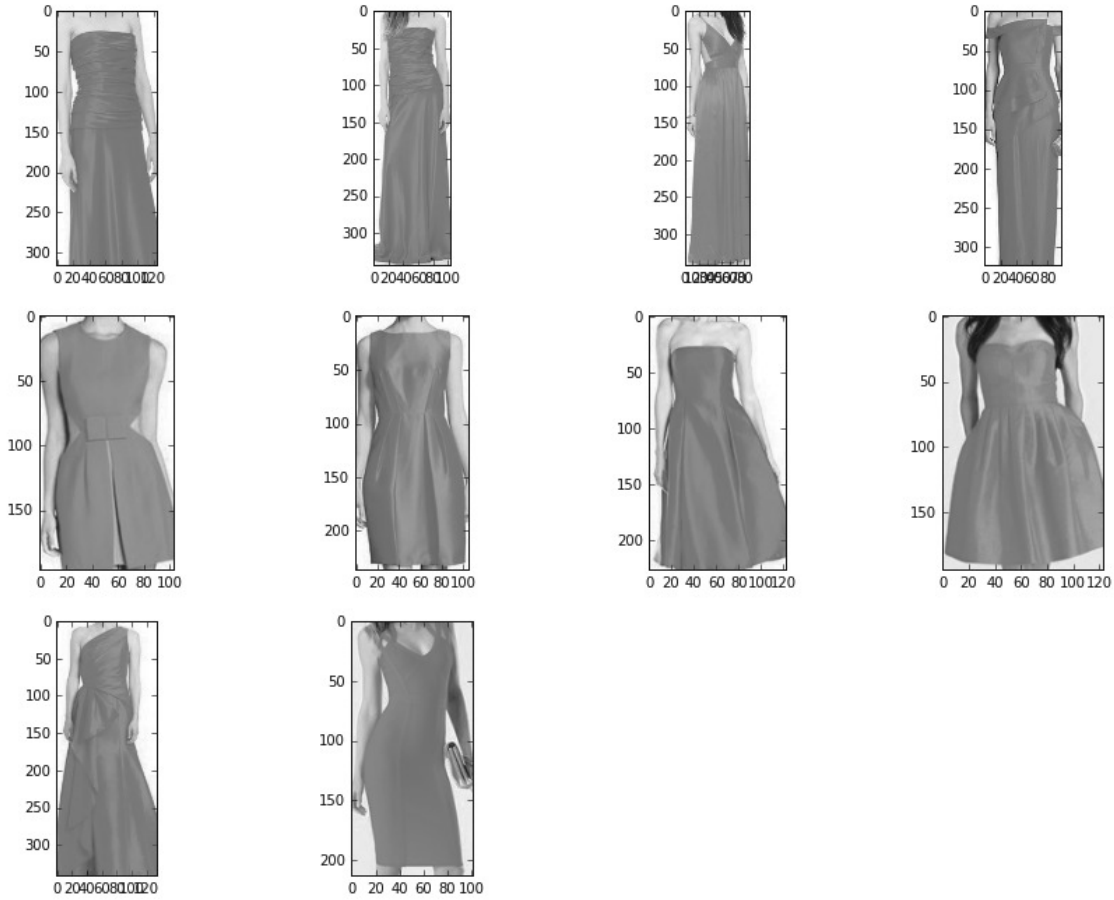
Similarity



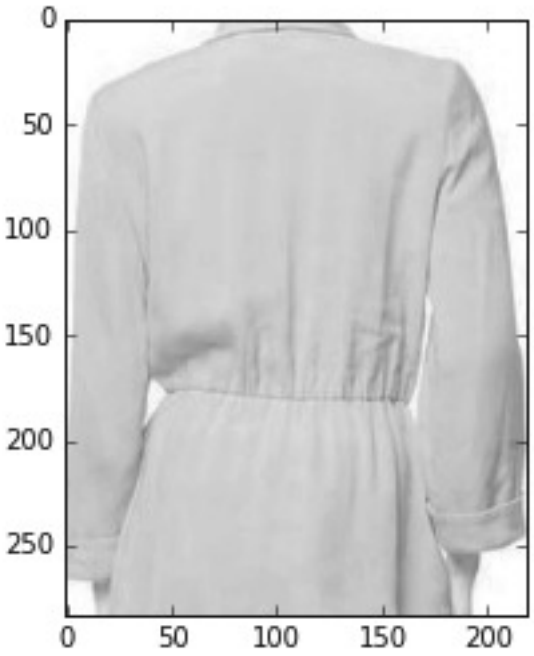
Query



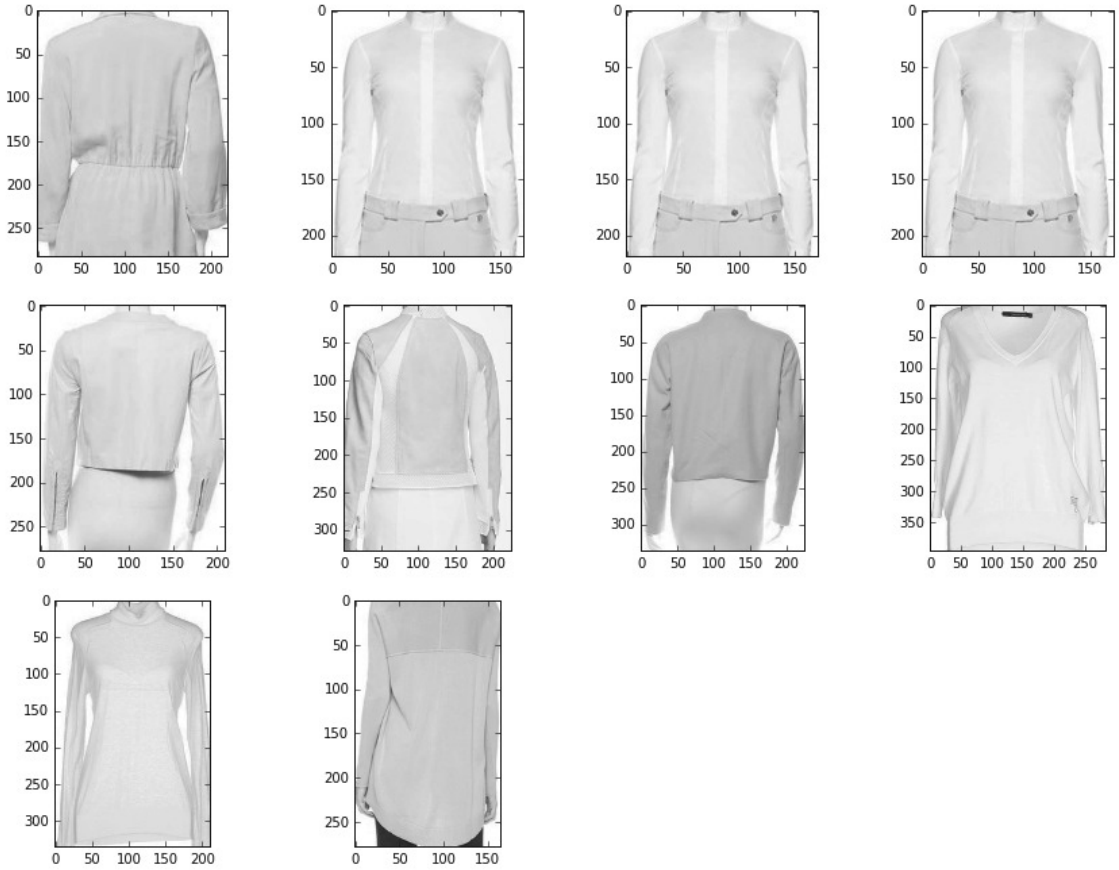
Similarity



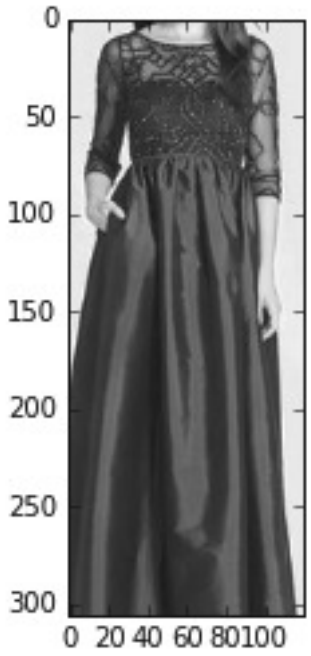
Query



Similarity

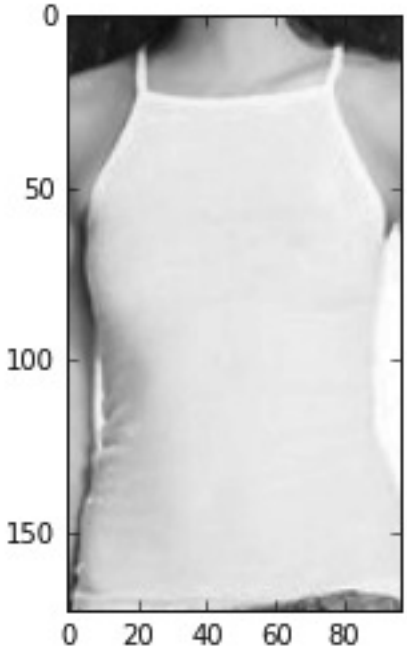


Query

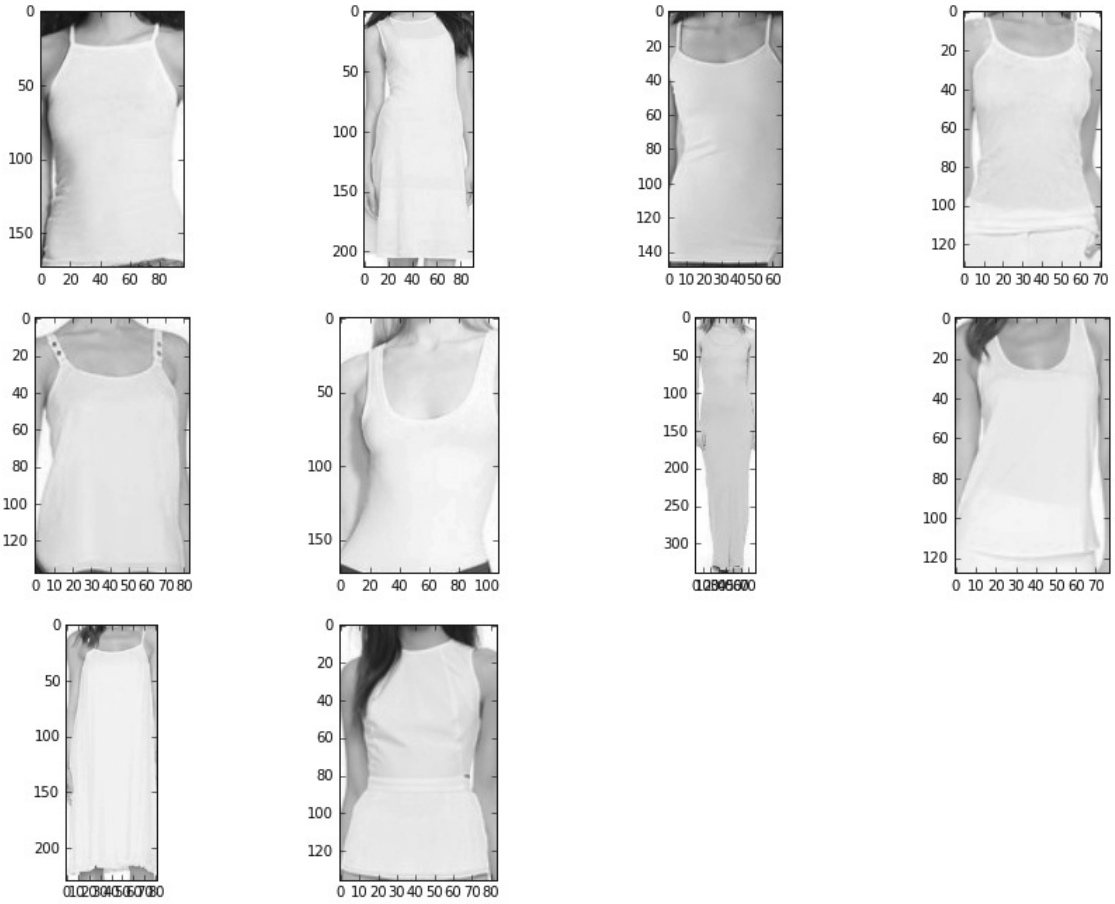


Similarity

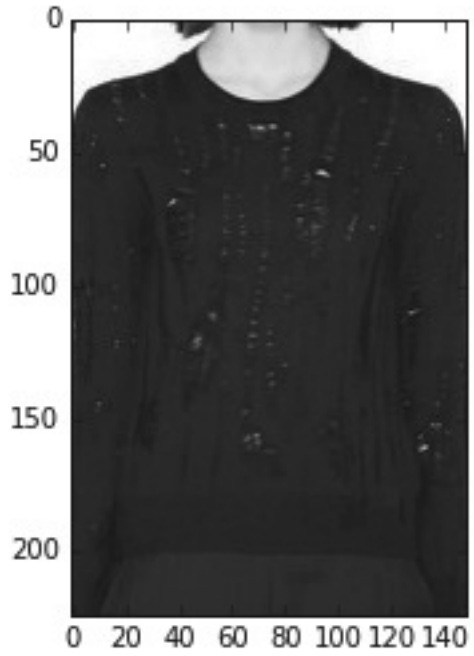
Query



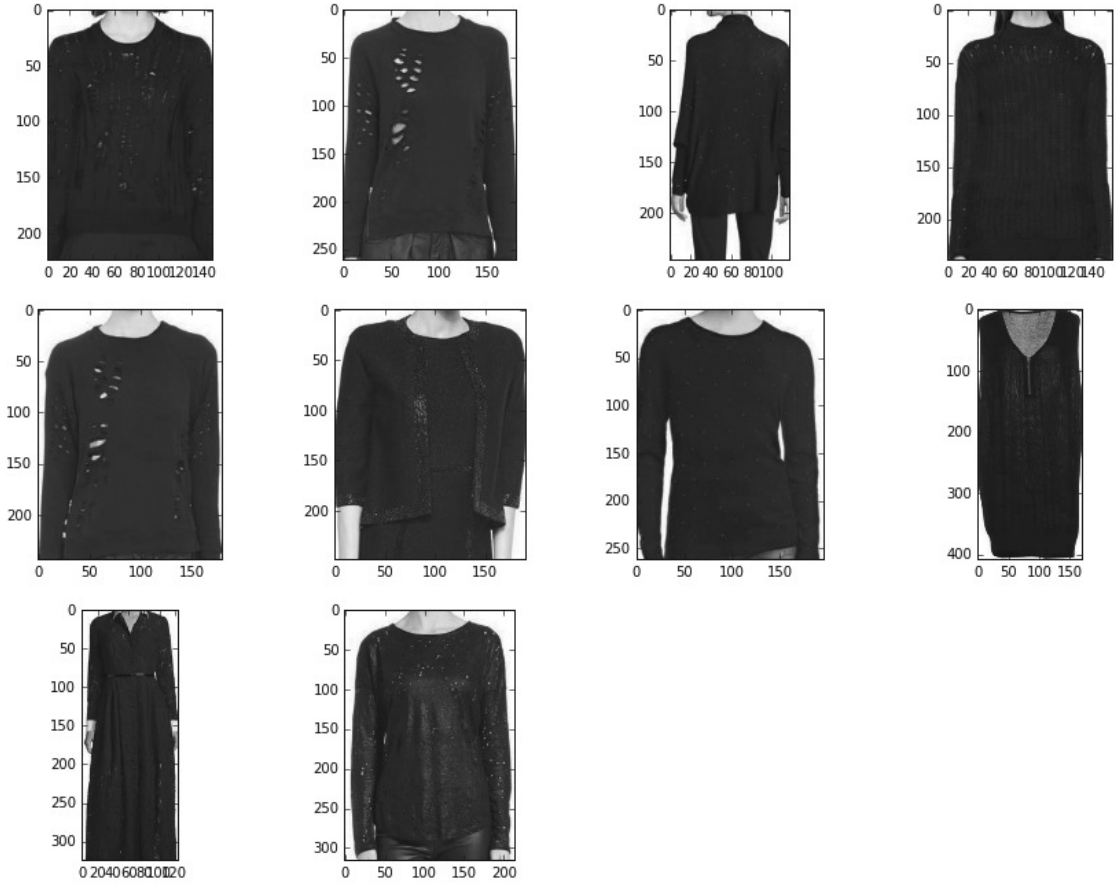

```
Similarity
```



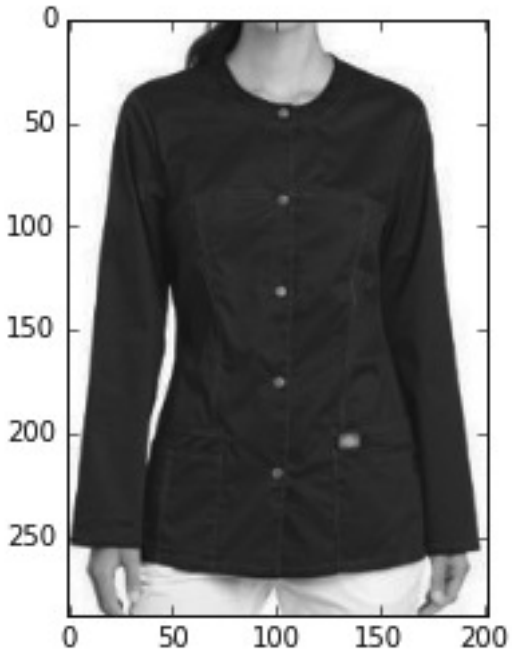
Query



Similarity



Query



Similarity

In [10]:

In [ ]: