In [47]:

```
import os
import sys, pprint
import numpy as np
import tensorflow as tf
from sklearn.neighbors import NearestNeighbors
from src import *
from src.CV_IO_utils import read_imgs_dir
from src.autoencoder import AutoEncoder
from src.CV_plot_utils import plot_query_retrieval, plot_tsne, p
lot_reconstructions
from src.CV_plot_utils import plot_query_retrieval, plot_tsne, p
lot_reconstructions
from src.CV_transform_utils import apply_transformer
from src.CV_transform_utils import resize_img, normalize_img
```

In [48]:

```
# Run mode: (autoencoder -> simpleAE, convAE) or (transfer learn
ing -> vgg19)
modelName = "vgg19"  # try: "simpleAE", "convAE", "vgg19"
trainModel = True
parallel = True  # use multicore processing
```

In [49]:

```
# Make paths
dataTrainDir = os.path.join(os.getcwd(), "data", "train")
dataTestDir = os.path.join(os.getcwd(), "data", "test")
outDir = os.path.join(os.getcwd(), "output", modelName)
if not os.path.exists(outDir):
    os.makedirs(outDir)
```

In [50]:

```
# Read images
extensions = [".jpg", ".jpeg"]
print("Reading train images from '{}'...".format(dataTrainDir))
imgs_train = read_imgs_dir(dataTrainDir, extensions, parallel=pa
rallel)
print("Reading test images from '{}'...".format(dataTestDir))
imgs_test = read_imgs_dir(dataTestDir, extensions, parallel=para
llel)
shape_img = imgs_train[0].shape
print("Image shape = {}".format(shape_img))
```

```
Reading train images from '/Users/nandhinigunalan/Co
mp_photography/Final_Project/Similarity_detection/da
ta/train'...
Reading test images from '/Users/nandhinigunalan/Com
p_photography/Final_Project/Similarity_detection/dat
a/test'...
Image shape = (227, 227, 3)
```

In [51]:

```
# Build models
if modelName in ["simpleAE", "convAE"]:

    # Set up autoencoder
    info = {
        "shape_img": shape_img,
        "autoencoderFile": os.path.join(outDir, "{}_autoecoder.h
5".format(modelName)),
        "encoderFile": os.path.join(outDir, "{}_encoder.h5".form
at(modelName)),
        "decoderFile": os.path.join(outDir, "{}_decoder.h5".form
at(modelName)),
    }
    model = AutoEncoder(modelName, info)
    model.set_arch()

    if modelName == "simpleAE":
        shape_img_resize = shape_img
        input_shape_model = (model.encoder.input.shape[1],)
        output_shape_model = (model.encoder.output.shape[1],)
        n_epochs = 300
    elif modelName == "convAE":
```

```
        shape_img_resize = shape_img
        input_shape_model = tuple([int(x) for x in model.encoder
.input.shape[1:]])
        output_shape_model = tuple([int(x) for x in model.encode
r.output.shape[1:]])
        n_epochs = 500
    else:
        raise Exception("Invalid modelName!")

elif modelName in ["vgg19"]:

    # Load pre-trained VGG19 model + higher level layers
    print("Loading VGG19 pre-trained model...")
    model = tf.keras.applications.VGG19(weights='imagenet', incl
ude_top=False,
                                        input_shape=shape_img)
    model.summary()

    shape_img_resize = tuple([int(x) for x in model.input.shape[
1:]])
    input_shape_model = tuple([int(x) for x in model.input.shape
[1:]])
    output_shape_model = tuple([int(x) for x in model.output.sha
pe[1:]])
    n_epochs = None

else:
    raise Exception("Invalid modelName!")
```

```
Loading VGG19 pre-trained model...
_________________________________________________________________
Layer (type)                 Output Shape              
Param #   
=================================================================
input_6 (InputLayer)         (None, 227, 227, 3)       
0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 227, 227, 64)      
1792      
_________________________________________________________________
```

```
block1_conv2 (Conv2D)        (None, 227, 227, 64)
36928
_________________________________________________________________
_____________
block1_pool (MaxPooling2D)   (None, 113, 113, 64)
0
_________________________________________________________________
_____________
block2_conv1 (Conv2D)        (None, 113, 113, 128)
73856
_________________________________________________________________
_____________
block2_conv2 (Conv2D)        (None, 113, 113, 128)
147584
_________________________________________________________________
_____________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)
0
_________________________________________________________________
_____________
block3_conv1 (Conv2D)        (None, 56, 56, 256)
295168
_________________________________________________________________
_____________
block3_conv2 (Conv2D)        (None, 56, 56, 256)
590080
_________________________________________________________________
_____________
block3_conv3 (Conv2D)        (None, 56, 56, 256)
590080
_________________________________________________________________
_____________
block3_conv4 (Conv2D)        (None, 56, 56, 256)
590080
_________________________________________________________________
_____________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)
0
_________________________________________________________________
_____________
block4_conv1 (Conv2D)        (None, 28, 28, 512)
1180160
_________________________________________________________________
_____________
block4_conv2 (Conv2D)        (None, 28, 28, 512)
```

```
2359808
_________________________________________________________________
_____________
block4_conv3 (Conv2D)        (None, 28, 28, 512)
2359808
_________________________________________________________________
_____________
block4_conv4 (Conv2D)        (None, 28, 28, 512)
2359808
_________________________________________________________________
_____________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)
0
_________________________________________________________________
_____________
block5_conv1 (Conv2D)        (None, 14, 14, 512)
2359808
_________________________________________________________________
_____________
block5_conv2 (Conv2D)        (None, 14, 14, 512)
2359808
_________________________________________________________________
_____________
block5_conv3 (Conv2D)        (None, 14, 14, 512)
2359808
_________________________________________________________________
_____________
block5_conv4 (Conv2D)        (None, 14, 14, 512)
2359808
_________________________________________________________________
_____________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)
0
=================================================
=============
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
_________________________________________________________________
_____________
```

In [52]:

```
# Print some model info
print("input_shape_model = {}".format(input_shape_model))
print("output_shape_model = {}".format(output_shape_model))

# Apply transformations to all images
class ImageTransformer(object):

    def __init__(self, shape_resize):
        self.shape_resize = shape_resize

    def __call__(self, img):
        img_transformed = resize_img(img, self.shape_resize)
        img_transformed = normalize_img(img_transformed)
        return img_transformed

transformer = ImageTransformer(shape_img_resize)
print("Applying image transformer to training images...")
imgs_train_transformed = apply_transformer(imgs_train, transform
er, parallel=parallel)
print("Applying image transformer to test images...")
imgs_test_transformed = apply_transformer(imgs_test, transformer
, parallel=parallel)
```

```
input_shape_model = (227, 227, 3)
output_shape_model = (7, 7, 512)
Applying image transformer to training images...
Applying image transformer to test images...
```

In [53]:

```
# Convert images to numpy array
X_train = np.array(imgs_train_transformed).reshape((-1,) + input
_shape_model)
X_test = np.array(imgs_test_transformed).reshape((-1,) + input_s
hape_model)
print(" -> X_train.shape = {}".format(X_train.shape))
print(" -> X_test.shape = {}".format(X_test.shape))

# Train (if necessary)
if modelName in ["simpleAE", "convAE"]:
    if trainModel:
        model.compile(loss="binary_crossentropy", optimizer="ada
m")
        model.fit(X_train, n_epochs=n_epochs, batch_size=256)
        model.save_models()
    else:
        model.load_models(loss="binary_crossentropy", optimizer=
"adam")
```

```
 -> X_train.shape = (24, 227, 227, 3)
 -> X_test.shape = (3, 227, 227, 3)
```

In [54]:

```
# Create embeddings using model
print("Inferencing embeddings using pre-trained model...")
E_train = model.predict(X_train)
E_train_flatten = E_train.reshape((-1, np.prod(output_shape_mode
l)))
E_test = model.predict(X_test)
E_test_flatten = E_test.reshape((-1, np.prod(output_shape_model)
))
print(" -> E_train.shape = {}".format(E_train.shape))
print(" -> E_test.shape = {}".format(E_test.shape))
print(" -> E_train_flatten.shape = {}".format(E_train_flatten.sh
ape))
print(" -> E_test_flatten.shape = {}".format(E_test_flatten.shap
e))
```

```
Inferencing embeddings using pre-trained model...
 -> E_train.shape = (24, 7, 7, 512)
 -> E_test.shape = (3, 7, 7, 512)
 -> E_train_flatten.shape = (24, 25088)
 -> E_test_flatten.shape = (3, 25088)
```

In [55]:

```
# Make reconstruction visualizations
if modelName in ["simpleAE", "convAE"]:
    print("Visualizing database image reconstructions...")
    imgs_train_reconstruct = model.decoder.predict(E_train)
    if modelName == "simpleAE":
        imgs_train_reconstruct = imgs_train_reconstruct.reshape(
(-1,) + shape_img_resize)
    plot_reconstructions(imgs_train, imgs_train_reconstruct,
                         os.path.join(outDir, "{}_reconstruct.pn
g".format(modelName)),
                         range_imgs=[0, 255],
                         range_imgs_reconstruct=[0, 1])
```

In [56]:

```
# Fit kNN model on training images
print("Fitting k-nearest-neighbour model on training images...")
knn = NearestNeighbors(n_neighbors=5, metric="cosine")
knn.fit(E_train_flatten)
```

```
Fitting k-nearest-neighbour model on training images
...
```

Out[56]:

```
NearestNeighbors(algorithm='auto', leaf_size=30, met
ric='cosine',
                 metric_params=None, n_jobs=None, n_
neighbors=5, p=2,
                 radius=1.0)
```

In [57]:

```
# Perform image retrieval on test images
print("Performing image retrieval on test images...")
for i, emb_flatten in enumerate(E_test_flatten):
    _, indices = knn.kneighbors([emb_flatten]) # find k nearest
train neighbours
    img_query = imgs_test[i] # query image
    imgs_retrieval = [imgs_train[idx] for idx in indices.flatten
()] # retrieval images
    outFile = os.path.join(outDir, "{}_retrieval_{}.png".format(
modelName, i))
    plot_query_retrieval(img_query, imgs_retrieval, outFile)
```

```
Performing image retrieval on test images...
```

In [58]:

```
# Plot t-SNE visualization
print("Visualizing t-SNE on training images...")
outFile = os.path.join(outDir, "{}_tsne.png".format(modelName))
plot_tsne(E_train_flatten, imgs_train, outFile)
```

```
Visualizing t-SNE on training images...
```

In [59]:

```
print("Images similar to test image found!")
```

```
Images similar to test image found!
```

In [ ]:

```

```