Deep Learning for Computer Vision

PyConDE 2017 Talk

Alex Conway alex@numberboost.com

Setup

imports

load vgg

In [1]:
```
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential
from keras.layers import Dropout, Flatten, Dense
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.callbacks import ModelCheckpoint
```
executed in 13.7s, finished 00:14:39 2017-10-25

```
Using TensorFlow backend.
```

In [2]:
```
from keras import backend as K
K.set_image_dim_ordering('th')
```
executed in 4ms, finished 00:14:39 2017-10-25

for plotting and misc

In [4]:
```
# setup matplotlib to display plots in the notebook
%matplotlib inline

# third party imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# setup display options
pd.options.display.max_rows = 200
pd.options.display.float_format = '{:,.5g}'.format
np.set_printoptions(precision=5, suppress=False)

# setup seaborn to use matplotlib defaults & styles
sns.set()
sns.set(font_scale=1.2)
sns.set_style("whitegrid", {'axes.grid' : False})

import os
import sys
import time
```

executed in 318ms, finished 00:14:40 2017-10-25

for PCA

In [160]:
```
from sklearn.decomposition import PCA
```

executed in 250ms, finished 01:57:45 2017-10-25

turn off jupyter notebook keras warnings

In [5]:
```
import warnings
warnings.filterwarnings(action='ignore')
```

executed in 3ms, finished 00:14:40 2017-10-25

paths

In [6]:
```
pwd = '/mnt/data/pycon/'
```

executed in 5ms, finished 00:14:40 2017-10-25

In [7]:
```
path_models = pwd + 'models/'
# for previewing data augmentation output:
# path_preview = path + 'data/preview/'
```

executed in 6ms, finished 00:14:40 2017-10-25

In [8]:
```
path_data = pwd + 'data/'
```

executed in 5ms, finished 00:14:40 2017-10-25

In [9]:
```
path_data_train = path_data +'train/'
path_data_valid = path_data +'valid/'
path_data_test = path_data +'test/'
# output
path_data_train, path_data_valid, path_data_test
```

executed in 8ms, finished 00:14:40 2017-10-25

Out[9]:
```
('/mnt/data/pycon/data/train/',
 '/mnt/data/pycon/data/valid/',
 '/mnt/data/pycon/data/test/')
```

In [10]:
```
!tree -d /mnt/data/pycon/
```

executed in 124ms, finished 00:14:40 2017-10-25

```
/mnt/data/pycon/
├── data
│   ├── test
│   │   └── unknown
│   ├── train
│   │   ├── accessories
│   │   ├── jackets
│   │   ├── jeans
│   │   ├── knitwear
│   │   ├── shirts
│   │   ├── shoes
│   │   ├── shorts
│   │   └── tees
│   └── valid
│       ├── accessories
│       ├── jackets
│       ├── jeans
│       ├── knitwear
│       ├── shirts
│       ├── shoes
│       ├── shorts
│       └── tees
└── models

22 directories
```

View some of the data

In [11]:
```
def plotimg(imgpath):
    img= plt.imread(imgpath)
    imgplot = plt.imshow(img)
```

executed in 3ms, finished 00:14:40 2017-10-25

In [12]:

```
def plot_pic_grid(path, filenames, add_title=True):
    # set figsize
    fig = plt.figure()
    fig.set_size_inches((16,8))

    plotted = 0

    for c, r in enumerate(filenames):

        # get path to image file
        img_path_on_disk = path + r

        if len(img_path_on_disk) > 0:

            if plotted < 10:

                plotted+=1

                # plotting 10 images
                a = fig.add_subplot(2, 5, (plotted))
                img= plt.imread(img_path_on_disk)
                imgplot = a.imshow(img)
                if add_title:
                    a.set_title(r)

                #print user_id, 'image plotted'
        else:
            # print user_id, 'no pic available'
            pass
```

executed in 13ms, finished 00:14:40 2017-10-25

In [13]:
```
plot_pic_grid(path_data +'train/jeans/', os.listdir(path_data +'train/jeans/')[:10])
```

executed in 1.98s, finished 00:14:42 2017-10-25

In [14]:
```
plot_pic_grid(path_data +'train/jackets/', os.listdir(path_data +'train/jackets/')[:10])
```

executed in 1.98s, finished 00:14:54 2017-10-25

Prepare Data [...]

Fit Model

We don't have a lot of data so we'll use a pre-trained convolutional.

This pre-trained network was built to predict which of 1000 ImageNet classes a particular image belongs to so the weights and convolutional filters are able to detect a wide variety of shapes and patterns.

We'll then chop off the final dense layers and keep only the pre-trained convolutional layers ("bottleneck features").

Next, we'll create our own final layers and train just these final layers for our task...

Setup image dimensions and number of samples

In [16]:
```
# dimension our images will be rescaled to (default size for VGG)
img_width, img_height = 224, 224
```

executed in 3ms, finished 00:15:44 2017-10-25

Instantiate pre-trained VGGNet and load weights

https://gist.github.com/fchollet/f35fbc80e066a49d65f1688a7e99f069 (https://gist.github.com/fchollet/f35fbc80e066a49d65f1688a7e99f069)

In [17]:

```
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Convolution2D, MaxPooling2D, ZeroPadding2D
from keras.layers import Activation, Dropout, Flatten, Dense
```

executed in 4ms, finished 00:15:45 2017-10-25

In [171]:

```
# https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3

model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(3,224,224)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1000, activation='softmax'))
```

executed in 327ms, finished 02:12:10 2017-10-25

Download pre-trained weights

In [172]:
```
# run the command line function below to download pre-trained VGG weights:
# !wget https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_weights_th_dim_ordering_th_kernels.h5
```
executed in 2ms, finished 02:12:11 2017-10-25

In [173]:
```
path_pretrained_vgg = '/mnt/data/pretrained_weights/vgg16_weights_th_dim_ordering_th_kernels.h5'
```
executed in 3ms, finished 02:12:11 2017-10-25

load pretrained weights

In [174]:
```
model.load_weights(path_pretrained_vgg)
```
executed in 656ms, finished 02:12:13 2017-10-25

In [175]:
```
# pop layers until just have the bottleneck max pooling 512,7,7 layer
for i in range(0,6):
    model.layers.pop()

model.outputs = [model.layers[-1].output]
model.layers[-1].outbound_nodes = []
```
executed in 6ms, finished 02:12:13 2017-10-25

In [176]:
```
model.compile(optimizer='rmsprop', loss='binary_crossentropy')
```
executed in 29ms, finished 02:12:14 2017-10-25

In [177]:
```
model.summary()
```

executed in 17ms, finished 02:12:16 2017-10-25

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
zero_padding2d_14 (ZeroPaddi (None, 3, 226, 226)       0         
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 64, 224, 224)      1792      
_________________________________________________________________
zero_padding2d_15 (ZeroPaddi (None, 64, 226, 226)      0         
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 64, 224, 224)      36928     
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 64, 112, 112)      0         
_________________________________________________________________
zero_padding2d_16 (ZeroPaddi (None, 64, 114, 114)      0         
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 128, 112, 112)     73856     
_________________________________________________________________
zero_padding2d_17 (ZeroPaddi (None, 128, 114, 114)     0         
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 128, 112, 112)     147584    
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 128, 56, 56)       0         
_________________________________________________________________
zero_padding2d_18 (ZeroPaddi (None, 128, 58, 58)       0         
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 256, 56, 56)       295168    
_________________________________________________________________
zero_padding2d_19 (ZeroPaddi (None, 256, 58, 58)       0         
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 256, 56, 56)       590080    
_________________________________________________________________
zero_padding2d_20 (ZeroPaddi (None, 256, 58, 58)       0         
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 256, 56, 56)       590080    
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 256, 28, 28)       0         
_________________________________________________________________
zero_padding2d_21 (ZeroPaddi (None, 256, 30, 30)       0         
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 512, 28, 28)       1180160   
_________________________________________________________________
zero_padding2d_22 (ZeroPaddi (None, 512, 30, 30)       0         
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 512, 28, 28)       2359808   
_________________________________________________________________
zero_padding2d_23 (ZeroPaddi (None, 512, 30, 30)       0         
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 512, 28, 28)       2359808   
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 512, 14, 14)       0         
_________________________________________________________________
zero_padding2d_24 (ZeroPaddi (None, 512, 16, 16)       0         
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 512, 14, 14)       2359808   
_________________________________________________________________
zero_padding2d_25 (ZeroPaddi (None, 512, 16, 16)       0         
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 512, 14, 14)       2359808   
```

```
_________________________________________________________________
zero_padding2d_26 (ZeroPaddi (None, 512, 16, 16)       0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 512, 14, 14)       2359808
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 512, 7, 7)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

> notice now the final layer of the network is the 512 x 7 x 7 bottleneck layer

Precompute bottleneck outputs for train and validation data

> We precompute the bottleneck features since that's where most of the computation time is - once we've precomputed them, we'll just learn the weights on final layer(s) added on top of the bottleneck features but if we didn't precompute them, we'd have to compute them each time we pass an image through the entire network even though we're only learning weights on the final layer.

create generators

In [26]:
```
# used to rescale the pixel values from [0, 255] to [0, 1] interval
datagen = ImageDataGenerator(rescale=1./255)
```

executed in 3ms, finished 00:16:11 2017-10-25

In [27]:
```
train_generator_bottleneck = datagen.flow_from_directory(
        path_data_train,
        target_size=(img_width, img_height),
        batch_size=1,
        class_mode=None,
        shuffle=False)
```

executed in 210ms, finished 00:16:12 2017-10-25

```
Found 3467 images belonging to 8 classes.
```

In [28]:
```
validation_generator_bottleneck = datagen.flow_from_directory(
        path_data_valid,
        target_size=(img_width, img_height),
        batch_size=1,
        class_mode=None,
        shuffle=False)
```

executed in 108ms, finished 00:16:12 2017-10-25

```
Found 382 images belonging to 8 classes.
```

In [29]:
```
train_generator_bottleneck.class_indices
```
executed in 4ms, finished 00:16:12 2017-10-25

Out[29]:
```
{'accessories': 0,
 'jackets': 1,
 'jeans': 2,
 'knitwear': 3,
 'shirts': 4,
 'shoes': 5,
 'shorts': 6,
 'tees': 7}
```

In [30]:
```
validation_generator_bottleneck.class_indices
```
executed in 5ms, finished 00:16:13 2017-10-25

Out[30]:
```
{'accessories': 0,
 'jackets': 1,
 'jeans': 2,
 'knitwear': 3,
 'shirts': 4,
 'shoes': 5,
 'shorts': 6,
 'tees': 7}
```

precompute bottleneck features

In [31]:
```
bottleneck_features_train = model.predict_generator(train_generator_bottleneck, train_generator_bottleneck.n, verbose = 1)
np.save(open(path_models + 'bottleneck_features_train.npy', 'wb'), bottleneck_features_train)
```
executed in 1m 45.6s, finished 00:17:59 2017-10-25

```
3467/3467 [==============================] - 102s
```

In [32]:
```
bottleneck_features_validation = model.predict_generator(validation_generator_bottleneck, validation_generator_bottleneck.n, verbose = 1)
np.save(open(path_models + 'bottleneck_features_validation.npy', 'wb'), bottleneck_features_validation)
```
executed in 11.6s, finished 00:18:11 2017-10-25

```
381/382 [============================>.] - ETA: 0s
```

In [33]:
```
train_data = np.load(open(path_models + 'bottleneck_features_train.npy', 'rb'))
train_labels = train_generator_bottleneck.classes

validation_data = np.load(open(path_models + 'bottleneck_features_validation.npy', 'rb'))
validation_labels = validation_generator_bottleneck.classes
```
executed in 200ms, finished 00:18:11 2017-10-25

Add new final layer(s)

add fully connected layers (on top of bottleneck convolutional layers)

> Can play around with the number and size of dense layers we add here...

In [59]:
```
model_top = Sequential()
model_top.add(Flatten(input_shape=train_data.shape[1:]))
model_top.add(Dense(256, activation='relu'))
model_top.add(Dropout(0.5))
model_top.add(Dense(len(np.unique(train_labels)), activation='softmax'))
```
executed in 43ms, finished 01:28:22 2017-10-25

I tried a bunch of optimizers and got the highest val_acc on VGG with adadelta, near runner up was adam

In [60]:
```
model_top.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```
executed in 34ms, finished 01:28:23 2017-10-25

In [164]:
```
model_top.summary()
```
executed in 7ms, finished 02:00:38 2017-10-25

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_4 (Flatten)          (None, 25088)             0         
_________________________________________________________________
dense_8 (Dense)              (None, 256)               6422784   
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_9 (Dense)              (None, 8)                 2056      
=================================================================
Total params: 6,424,840
Trainable params: 6,424,840
Non-trainable params: 0
_________________________________________________________________
```

Learn weights on new final layer(s)

In [61]:
```
model_name = 'VGG_finetuned'
```
executed in 3ms, finished 01:28:24 2017-10-25

In [62]:
```
# define keras model checkpointer to save best model (highets val_acc)
checkpointer = ModelCheckpoint(path_models + model_name + '_BEST_.hdf5',
                               monitor='val_acc',
                               save_best_only=True,
                               save_weights_only=True)
```
executed in 3ms, finished 01:28:25 2017-10-25

In [63]:
```
nb_epoch = 100
```
executed in 3ms, finished 01:28:30 2017-10-25

In [159]:
```
# fit model, storing history in variable to plot learning curves
history = model_top.fit(train_data,
                train_labels,
                nb_epoch=nb_epoch,
                batch_size=32,
                validation_data=(validation_data, validation_labels),
                callbacks=[checkpointer])
```

executed in 2m 14s, finished 01:57:45 2017-10-25

```
Train on 3467 samples, validate on 382 samples
Epoch 1/100
3467/3467 [==============================] - 1s - loss: 0.1449 - acc: 0.9420 - val_loss: 0.1940 - val_acc: 0.9424
Epoch 2/100
3467/3467 [==============================] - 1s - loss: 0.1214 - acc: 0.9533 - val_loss: 0.1938 - val_acc: 0.9581
Epoch 3/100
3467/3467 [==============================] - 1s - loss: 0.1203 - acc: 0.9518 - val_loss: 0.1730 - val_acc: 0.9555
Epoch 4/100
3467/3467 [==============================] - 1s - loss: 0.1081 - acc: 0.9596 - val_loss: 0.1859 - val_acc: 0.9529
Epoch 5/100
3467/3467 [==============================] - 1s - loss: 0.1029 - acc: 0.9570 - val_loss: 0.2690 - val_acc: 0.9346
Epoch 6/100
3467/3467 [==============================] - 1s - loss: 0.1063 - acc: 0.9639 - val_loss: 0.2070 - val_acc: 0.9607
Epoch 7/100
3467/3467 [==============================] - 1s - loss: 0.0764 - acc: 0.9680 - val_loss: 0.2278 - val_acc: 0.9424
Epoch 8/100
3467/3467 [==============================] - 1s - loss: 0.0733 - acc: 0.9738 - val_loss: 0.2151 - val_acc: 0.9581
Epoch 9/100
3467/3467 [==============================] - 1s - loss: 0.0823 - acc: 0.9663 - val_loss: 0.1987 - val_acc: 0.9503
Epoch 10/100
3467/3467 [==============================] - 1s - loss: 0.0739 - acc: 0.9729 - val_loss: 0.2022 - val_acc: 0.9503
Epoch 11/100
3467/3467 [==============================] - 1s - loss: 0.0804 - acc: 0.9674 - val_loss: 0.2306 - val_acc: 0.9346
Epoch 12/100
3467/3467 [==============================] - 1s - loss: 0.0860 - acc: 0.9663 - val_loss: 0.1879 - val_acc: 0.9529
Epoch 13/100
3467/3467 [==============================] - 1s - loss: 0.0868 - acc: 0.9680 - val_loss: 0.1691 - val_acc: 0.9607
Epoch 14/100
3467/3467 [==============================] - 1s - loss: 0.0767 - acc: 0.9714 - val_loss: 0.1896 - val_acc: 0.9529
Epoch 15/100
3467/3467 [==============================] - 1s - loss: 0.0736 - acc: 0.9732 - val_loss: 0.1922 - val_acc: 0.9634
Epoch 16/100
3467/3467 [==============================] - 1s - loss: 0.0687 - acc: 0.9740 - val_loss: 0.2486 - val_acc: 0.9476
Epoch 17/100
3467/3467 [==============================] - 1s - loss: 0.0678 - acc: 0.9758 - val_loss: 0.2305 - val_acc: 0.9529
Epoch 18/100
3467/3467 [==============================] - 1s - loss: 0.0579 - acc: 0.9755 - val_loss: 0.2360 - val_acc: 0.9555
Epoch 19/100
3467/3467 [==============================] - 1s - loss: 0.0627 - acc: 0.9755 - val_loss: 0.2376 - val_acc: 0.9503
Epoch 20/100
3467/3467 [==============================] - 1s - loss: 0.0666 - acc: 0.9726 - val_loss: 0.2136 - val_acc: 0.9450
Epoch 21/100
3467/3467 [==============================] - 1s - loss: 0.0826 - acc: 0.9688 - val_loss: 0.1764 - val_acc: 0.9503
Epoch 22/100
3467/3467 [==============================] - 1s - loss: 0.0758 - acc: 0.9714 - val_loss: 0.2140 - val_acc: 0.9529
Epoch 23/100
3467/3467 [==============================] - 1s - loss: 0.0735 - acc: 0.9738 - val_loss: 0.2245 - val_acc: 0.9529
Epoch 24/100
3467/3467 [==============================] - 1s - loss: 0.0665 - acc: 0.9763 - val_loss: 0.2509 - val_acc: 0.9503
Epoch 25/100
3467/3467 [==============================] - 1s - loss: 0.0708 - acc: 0.9735 - val_loss: 0.2146 - val_acc: 0.9529
Epoch 26/100
```

```
3467/3467 [==============================] - 1s - loss: 0.0630 - acc: 0.9729 - val_loss: 0.2614 - val_acc: 0.9450
Epoch 27/100
3467/3467 [==============================] - 1s - loss: 0.0838 - acc: 0.9709 - val_loss: 0.2186 - val_acc: 0.9581
Epoch 28/100
3467/3467 [==============================] - 1s - loss: 0.0772 - acc: 0.9761 - val_loss: 0.2717 - val_acc: 0.9503
Epoch 29/100
3467/3467 [==============================] - 1s - loss: 0.0727 - acc: 0.9723 - val_loss: 0.2041 - val_acc: 0.9555
Epoch 30/100
3467/3467 [==============================] - 1s - loss: 0.0620 - acc: 0.9775 - val_loss: 0.3022 - val_acc: 0.9450
Epoch 31/100
3467/3467 [==============================] - 1s - loss: 0.0672 - acc: 0.9775 - val_loss: 0.1985 - val_acc: 0.9503
Epoch 32/100
3467/3467 [==============================] - 1s - loss: 0.0547 - acc: 0.9789 - val_loss: 0.2255 - val_acc: 0.9555
Epoch 33/100
3467/3467 [==============================] - 1s - loss: 0.0717 - acc: 0.9706 - val_loss: 0.2039 - val_acc: 0.9607
Epoch 34/100
3467/3467 [==============================] - 1s - loss: 0.0681 - acc: 0.9743 - val_loss: 0.2135 - val_acc: 0.9503
Epoch 35/100
3467/3467 [==============================] - 1s - loss: 0.0802 - acc: 0.9717 - val_loss: 0.3110 - val_acc: 0.9398
Epoch 36/100
3467/3467 [==============================] - 1s - loss: 0.0507 - acc: 0.9818 - val_loss: 0.2284 - val_acc: 0.9607
Epoch 37/100
3467/3467 [==============================] - 1s - loss: 0.0741 - acc: 0.9729 - val_loss: 0.2351 - val_acc: 0.9581
Epoch 38/100
3467/3467 [==============================] - 1s - loss: 0.0831 - acc: 0.9683 - val_loss: 0.1611 - val_acc: 0.9634
Epoch 39/100
3467/3467 [==============================] - 1s - loss: 0.0675 - acc: 0.9726 - val_loss: 0.2264 - val_acc: 0.9555
Epoch 40/100
3467/3467 [==============================] - 1s - loss: 0.0697 - acc: 0.9729 - val_loss: 0.2987 - val_acc: 0.9476
Epoch 41/100
3467/3467 [==============================] - 1s - loss: 0.0643 - acc: 0.9781 - val_loss: 0.2458 - val_acc: 0.9555
Epoch 42/100
3467/3467 [==============================] - 1s - loss: 0.0629 - acc: 0.9789 - val_loss: 0.2109 - val_acc: 0.9529
Epoch 43/100
3467/3467 [==============================] - 1s - loss: 0.0657 - acc: 0.9755 - val_loss: 0.2710 - val_acc: 0.9503
Epoch 44/100
3467/3467 [==============================] - 1s - loss: 0.0699 - acc: 0.9735 - val_loss: 0.1590 - val_acc: 0.9581
Epoch 45/100
3467/3467 [==============================] - 1s - loss: 0.0495 - acc: 0.9807 - val_loss: 0.2738 - val_acc: 0.9529
Epoch 46/100
3467/3467 [==============================] - 1s - loss: 0.0546 - acc: 0.9781 - val_loss: 0.2963 - val_acc: 0.9476
Epoch 47/100
3467/3467 [==============================] - 1s - loss: 0.0392 - acc: 0.9876 - val_loss: 0.2942 - val_acc: 0.9529
Epoch 48/100
3467/3467 [==============================] - 1s - loss: 0.0530 - acc: 0.9841 - val_loss: 0.2829 - val_acc: 0.9450
Epoch 49/100
3467/3467 [==============================] - 1s - loss: 0.0531 - acc: 0.9821 - val_loss: 0.2756 - val_acc: 0.9529
Epoch 50/100
3467/3467 [==============================] - 1s - loss: 0.0434 - acc: 0.9844 - val_loss: 0.2575 - val_acc: 0.9529
Epoch 51/100
3467/3467 [==============================] - 1s - loss: 0.0657 - acc: 0.9752 - val_loss: 0.2314 - val_acc: 0.9503
Epoch 52/100
3467/3467 [==============================] - 1s - loss: 0.0572 - acc: 0.9781 - val_loss: 0.2720 - val_acc: 0.9503
Epoch 53/100
3467/3467 [==============================] - 1s - loss: 0.0559 - acc: 0.9813 - val_loss: 0.2926 - val_acc: 0.9529
Epoch 54/100
3467/3467 [==============================] - 1s - loss: 0.0565 - acc: 0.9789 - val_loss: 0.3165 - val_acc: 0.9372
Epoch 55/100
3467/3467 [==============================] - 1s - loss: 0.0632 - acc: 0.9761 - val_loss: 0.2729 - val_acc: 0.9476
Epoch 56/100
3467/3467 [==============================] - 1s - loss: 0.0600 - acc: 0.9752 - val_loss: 0.2973 - val_acc: 0.9476
```

```
Epoch 57/100
3467/3467 [==============================] - 1s - loss: 0.0759 - acc: 0.9726 - val_loss: 0.2025 - val_acc: 0.9529
Epoch 58/100
3467/3467 [==============================] - 1s - loss: 0.0607 - acc: 0.9798 - val_loss: 0.2774 - val_acc: 0.9503
Epoch 59/100
3467/3467 [==============================] - 1s - loss: 0.0512 - acc: 0.9787 - val_loss: 0.2634 - val_acc: 0.9503
Epoch 60/100
3467/3467 [==============================] - 1s - loss: 0.0803 - acc: 0.9703 - val_loss: 0.2163 - val_acc: 0.9529
Epoch 61/100
3467/3467 [==============================] - 1s - loss: 0.0496 - acc: 0.9824 - val_loss: 0.2223 - val_acc: 0.9634
Epoch 62/100
3467/3467 [==============================] - 1s - loss: 0.0501 - acc: 0.9813 - val_loss: 0.2854 - val_acc: 0.9529
Epoch 63/100
3467/3467 [==============================] - 1s - loss: 0.0678 - acc: 0.9789 - val_loss: 0.2848 - val_acc: 0.9424
Epoch 64/100
3467/3467 [==============================] - 1s - loss: 0.0686 - acc: 0.9743 - val_loss: 0.2747 - val_acc: 0.9555
Epoch 65/100

3467/3467 [==============================] - 1s - loss: 0.0647 - acc: 0.9769 - val_loss: 0.2303 - val_acc: 0.9503
Epoch 66/100
3467/3467 [==============================] - 1s - loss: 0.0631 - acc: 0.9778 - val_loss: 0.2134 - val_acc: 0.9555
Epoch 67/100
3467/3467 [==============================] - 1s - loss: 0.0631 - acc: 0.9763 - val_loss: 0.2390 - val_acc: 0.9529
Epoch 68/100
3467/3467 [==============================] - 1s - loss: 0.0555 - acc: 0.9784 - val_loss: 0.2996 - val_acc: 0.9476
Epoch 69/100
3467/3467 [==============================] - 1s - loss: 0.0472 - acc: 0.9856 - val_loss: 0.2797 - val_acc: 0.9555
Epoch 70/100
3467/3467 [==============================] - 1s - loss: 0.0602 - acc: 0.9784 - val_loss: 0.2944 - val_acc: 0.9529
Epoch 71/100
3467/3467 [==============================] - 1s - loss: 0.0815 - acc: 0.9712 - val_loss: 0.2510 - val_acc: 0.9555
Epoch 72/100
3467/3467 [==============================] - 1s - loss: 0.0591 - acc: 0.9752 - val_loss: 0.3294 - val_acc: 0.9319
Epoch 73/100
3467/3467 [==============================] - 1s - loss: 0.0655 - acc: 0.9749 - val_loss: 0.3002 - val_acc: 0.9529
Epoch 74/100
3467/3467 [==============================] - 1s - loss: 0.0573 - acc: 0.9813 - val_loss: 0.2823 - val_acc: 0.9476
Epoch 75/100
3467/3467 [==============================] - 1s - loss: 0.0492 - acc: 0.9833 - val_loss: 0.3431 - val_acc: 0.9372
Epoch 76/100
3467/3467 [==============================] - 1s - loss: 0.0535 - acc: 0.9827 - val_loss: 0.3645 - val_acc: 0.9450
Epoch 77/100
3467/3467 [==============================] - 1s - loss: 0.0618 - acc: 0.9778 - val_loss: 0.2768 - val_acc: 0.9581
Epoch 78/100
3467/3467 [==============================] - 1s - loss: 0.0388 - acc: 0.9859 - val_loss: 0.2818 - val_acc: 0.9476
Epoch 79/100
3467/3467 [==============================] - 1s - loss: 0.0414 - acc: 0.9867 - val_loss: 0.2449 - val_acc: 0.9555
Epoch 80/100
3467/3467 [==============================] - 1s - loss: 0.0594 - acc: 0.9821 - val_loss: 0.3400 - val_acc: 0.9398
Epoch 81/100
3467/3467 [==============================] - 1s - loss: 0.0492 - acc: 0.9827 - val_loss: 0.2631 - val_acc: 0.9529
Epoch 82/100
3467/3467 [==============================] - 1s - loss: 0.0496 - acc: 0.9821 - val_loss: 0.3296 - val_acc: 0.9450
Epoch 83/100
3467/3467 [==============================] - 1s - loss: 0.0664 - acc: 0.9766 - val_loss: 0.3010 - val_acc: 0.9555
Epoch 84/100
3467/3467 [==============================] - 1s - loss: 0.0550 - acc: 0.9807 - val_loss: 0.2712 - val_acc: 0.9424
Epoch 85/100
3467/3467 [==============================] - 1s - loss: 0.0545 - acc: 0.9838 - val_loss: 0.3102 - val_acc: 0.9555
Epoch 86/100
3467/3467 [==============================] - 1s - loss: 0.0487 - acc: 0.9838 - val_loss: 0.3341 - val_acc: 0.9503
```

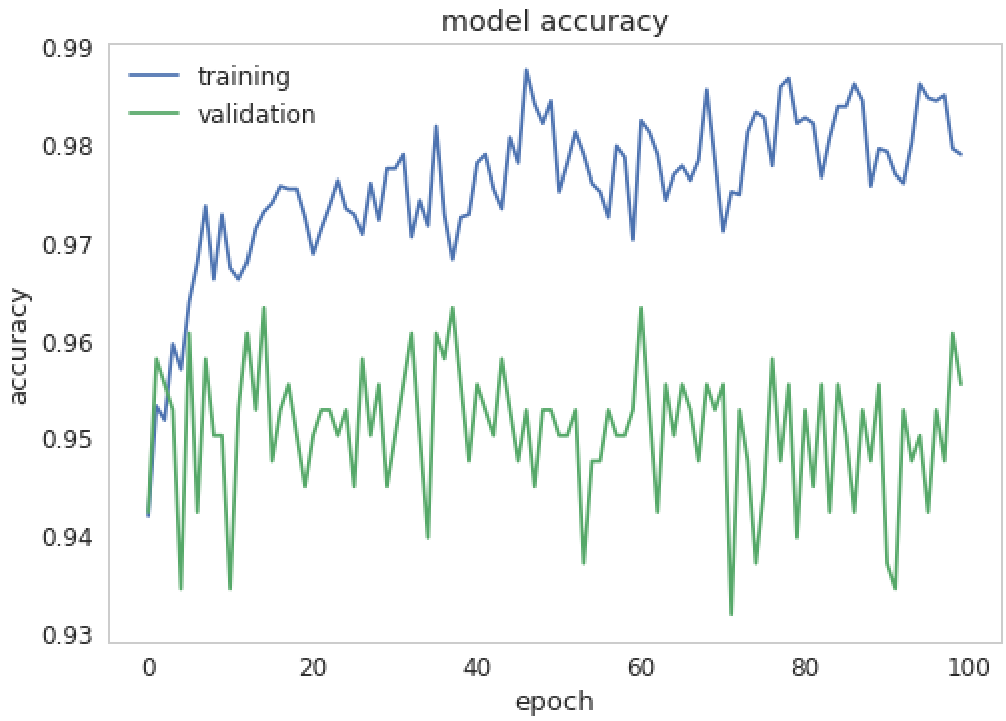
```
Epoch 87/100
3467/3467 [==============================] - 1s - loss: 0.0440 - acc: 0.9862 - val_loss: 0.3427 - val_acc: 0.9424
Epoch 88/100
3467/3467 [==============================] - 1s - loss: 0.0444 - acc: 0.9844 - val_loss: 0.3061 - val_acc: 0.9529
Epoch 89/100
3467/3467 [==============================] - 1s - loss: 0.0634 - acc: 0.9758 - val_loss: 0.2709 - val_acc: 0.9476
Epoch 90/100
3467/3467 [==============================] - 1s - loss: 0.0529 - acc: 0.9795 - val_loss: 0.3436 - val_acc: 0.9555
Epoch 91/100
3467/3467 [==============================] - 1s - loss: 0.0578 - acc: 0.9792 - val_loss: 0.4559 - val_acc: 0.9372
Epoch 92/100
3467/3467 [==============================] - 1s - loss: 0.0628 - acc: 0.9769 - val_loss: 0.3600 - val_acc: 0.9346
Epoch 93/100
3467/3467 [==============================] - 1s - loss: 0.0673 - acc: 0.9761 - val_loss: 0.2970 - val_acc: 0.9529
Epoch 94/100
3467/3467 [==============================] - 1s - loss: 0.0595 - acc: 0.9801 - val_loss: 0.3057 - val_acc: 0.9476
Epoch 95/100
3467/3467 [==============================] - 1s - loss: 0.0437 - acc: 0.9862 - val_loss: 0.3246 - val_acc: 0.9503
Epoch 96/100
3467/3467 [==============================] - 1s - loss: 0.0492 - acc: 0.9847 - val_loss: 0.3493 - val_acc: 0.9424
Epoch 97/100
3467/3467 [==============================] - 1s - loss: 0.0403 - acc: 0.9844 - val_loss: 0.2687 - val_acc: 0.9529
Epoch 98/100
3467/3467 [==============================] - 1s - loss: 0.0462 - acc: 0.9850 - val_loss: 0.3214 - val_acc: 0.9476
Epoch 99/100
3467/3467 [==============================] - 1s - loss: 0.0577 - acc: 0.9795 - val_loss: 0.2081 - val_acc: 0.9607
Epoch 100/100
3467/3467 [==============================] - 1s - loss: 0.0632 - acc: 0.9789 - val_loss: 0.2861 - val_acc: 0.9555
```

Plot learning curves

In [161]:

```
# plot accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='upper left')
plt.show()

# plot loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='upper left')
plt.show()
```

executed in 753ms, finished 01:57:46 2017-10-25



In [162]:
```
# load weights from best epoch (saved by checkpointer)
model_top.load_weights(path_models + model_name + '_BEST_.hdf5')
```
executed in 21ms, finished 01:57:46 2017-10-25

In [163]:
```
# check metrics for this best model
print (model_top.metrics_names)
model_top.evaluate(validation_data, validation_labels, verbose = 0)
```
executed in 45ms, finished 01:57:46 2017-10-25

```
['loss', 'acc']
```

Out[163]:
```
[0.16105182613130337, 0.96335079282990299]
```

View predictions and some examples where the model is going wrong

Define function to do prediction

In [70]:
```
# Need to get bottleneck features from model
# then feed these to model_top to do actual prediction
def get_prediction(img_path, return_probabilities = False):
    img = image.load_img(img_path, target_size=(224, 224))
    x = image.img_to_array(img) / 255
    x = np.expand_dims(x, axis=0)
    x = model.predict(x)
    x = model_top.predict(x)
    if return_probabilities:
        return x
    else:
        return np.argmax(x)
```
executed in 7ms, finished 01:32:58 2017-10-25

In [75]:
```
# fit model, storing history in variable to plot learning curves
predictions = model_top.predict(validation_data,
                                batch_size=32)
```
executed in 42ms, finished 01:34:16 2017-10-25

In [85]:
```
# get argmax of each row in predictions output (which is a probability distribution)
# there's definitely a way to do this with numpy but I'm being lazy and looping...
predictions_labels = []

for c, row in enumerate(predictions):
    predictions_labels.append(np.argmax(row))
```
executed in 7ms, finished 01:36:37 2017-10-25

In [132]:
```
# build dataframe
df = pd.DataFrame({"filename":validation_generator_bottleneck.filenames, "prediction": predictions_labels, "truth": validation_labels})
df['correct'] = (df['prediction'] == df['truth']).astype(int)
df['correct'] = df['correct'].astype(int)
df['prediction'] = df['prediction'].astype(int)
```
executed in 8ms, finished 01:44:48 2017-10-25

In [133]:
```
# join class names
df_names = pd.DataFrame.from_dict(validation_generator_bottleneck.class_indices, orient = 'index')
df_names.columns = ['idx']
df_names['name'] = df_names.index
#
df = pd.merge(df,df_names,left_on = 'prediction', right_on='idx', how='left')
del df['idx']
```
executed in 11ms, finished 01:44:52 2017-10-25

In [135]:
```
df.columns = ['filename','prediction','truth','correct','prediction_name']
```
executed in 4ms, finished 01:45:00 2017-10-25

In [136]:
```
df = pd.merge(df,df_names,left_on = 'truth', right_on='idx', how='left')
del df['idx']
```
executed in 7ms, finished 01:45:06 2017-10-25

In [137]:
```
df.columns = ['filename','prediction','truth','correct','prediction_name','truth_name']
```
executed in 5ms, finished 01:45:10 2017-10-25

In [138]:
```
df.tail()
```
executed in 13ms, finished 01:45:11 2017-10-25

Out[138]:

	filename	prediction	truth	correct	prediction_name	truth_name
377	tees/productimg_774.jpg	7	7	1	tees	tees
378	tees/productimg_3815.jpg	7	7	1	tees	tees
379	tees/productimg_4073.jpg	7	7	1	tees	tees
380	tees/productimg_2408.jpg	7	7	1	tees	tees
381	tees/productimg_1355.jpg	3	7	0	knitwear	tees

errors:

In [154]:
```
df[df['correct'] == 0]
```

executed in 15ms, finished 01:50:54 2017-10-25

Out[154]:

	filename	prediction	truth	correct	prediction_name	truth_name
29	jackets/productimg_2592.jpg	3	1	0	knitwear	jackets
96	knitwear/productimg_1369.jpg	4	3	0	shirts	knitwear
101	knitwear/productimg_728.jpg	1	3	0	jackets	knitwear
104	knitwear/productimg_1143.jpg	1	3	0	jackets	knitwear
106	knitwear/productimg_2052.jpg	7	3	0	tees	knitwear
115	shirts/productimg_737.jpg	3	4	0	knitwear	shirts
118	shirts/productimg_167.jpg	7	4	0	tees	shirts
123	shirts/productimg_840.jpg	3	4	0	knitwear	shirts
131	shirts/productimg_274.jpg	7	4	0	tees	shirts
288	tees/productimg_1499.jpg	4	7	0	shirts	tees
294	tees/productimg_1457.jpg	3	7	0	knitwear	tees
307	tees/productimg_3208.jpg	3	7	0	knitwear	tees
323	tees/productimg_2736.jpg	3	7	0	knitwear	tees
370	tees/productimg_2776.jpg	3	7	0	knitwear	tees
381	tees/productimg_1355.jpg	3	7	0	knitwear	tees

Let's plot some of the errors

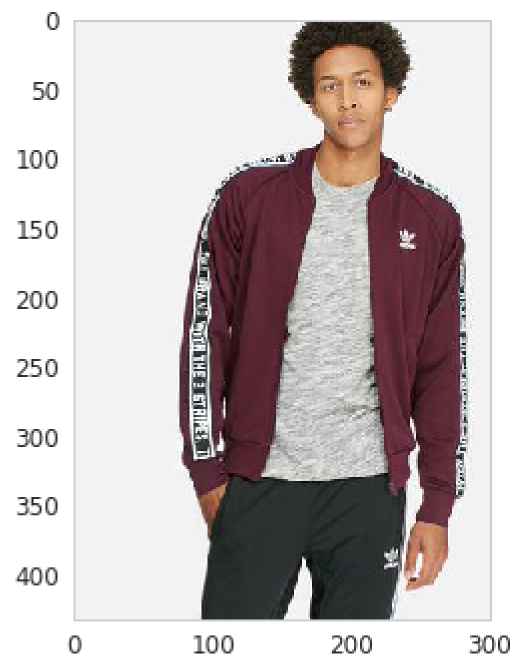
Prediction = shirt, truth = tee ... can see why the model predicted shirt lol

In [145]:
```
plotimg(path_data_valid + "tees/productimg_1499.jpg")
```

executed in 328ms, finished 01:47:48 2017-10-25

```
Model prediction = knitwear but truth = jacket
```

In [151]:
```
plotimg(path_data_valid + "jackets/productimg_2592.jpg")
```
executed in 340ms, finished 01:49:55 2017-10-25



Get activation vectors

Pop off final softmax layer and only keep activations from model top

In [185]:
```
model_top.layers.pop()
model_top.layers.pop()

model_top.outputs = [model_top.layers[-1].output]
model_top.layers[-1].outbound_nodes = []

model_top.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```
executed in 6ms, finished 02:13:24 2017-10-25

Out[185]:
```
<keras.layers.core.Dense at 0x7f3b906c89b0>
```

In [209]:
```
model_top.summary()
```

executed in 6ms, finished 02:15:54 2017-10-25

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_4 (Flatten)          (None, 25088)             0         
_________________________________________________________________
dense_8 (Dense)              (None, 256)               6422784   
=================================================================
Total params: 6,422,784
Trainable params: 6,422,784
Non-trainable params: 0
_________________________________________________________________
```

define function to get activations

In [230]:
```
# Need to get bottleneck features from model
# then feed these to model_top to get activations from fine-tuned new dense layer
def get_activation(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    x = image.img_to_array(img) / 255
    x = np.expand_dims(x, axis=0)
    x = model.predict(x)
    x = model_top.predict(x)
    return x[0]
```

executed in 6ms, finished 02:21:24 2017-10-25

get activations for each validation image

In [304]:
```
activations_list = []

for image_path in train_generator_bottleneck.filenames:
    #print (path_data_valid + image_path)
    activations_list.append(get_activation(path_data_train + image_path))
```

executed in 1m 51.3s, finished 04:01:01 2017-10-25

In [305]:
```
len(activations_list)
```

executed in 5ms, finished 04:01:01 2017-10-25

Out[305]: 3467

In [306]:
```
type(activations_list[0])
```

executed in 4ms, finished 04:14:43 2017-10-25

Out[306]: numpy.ndarray

In [307]:
```
# cast activations to numpy matrix
activations = np.asmatrix(activations_list)
```

executed in 6ms, finished 04:14:43 2017-10-25

In [308]:
```
# cast activations to dataframe
adf = pd.DataFrame(activations)
```

executed in 3ms, finished 04:14:44 2017-10-25

In [347]:
```
adf['filename'] = train_generator_bottleneck.filenames
```
executed in 3ms, finished 04:22:56 2017-10-25

In [348]:
```
adf.head()
```
executed in 26ms, finished 04:22:57 2017-10-25

Out[348]:

	0	1	2	3	4	5	6	7	8	9	...	247	248	249	250	251	252	253	254	255	filename
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	accessories/productimg_1610.jpg
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	accessories/productimg_1312.jpg
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	accessories/productimg_1.jpg
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	accessories/productimg_1247.jpg
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	accessories/productimg_1246.jpg

5 rows × 257 columns

Get Most Visually Similar Products

get activation for test image

In [349]:
```
os.listdir(path_data_test + 'unknown')
```
executed in 6ms, finished 04:22:59 2017-10-25

Out[349]:
```
['test6.jpg',
 'test3.jpg',
 'test7.jpg',
 'test1.jpg',
 'test8.jpg',
 'test5.jpg',
 'test4.jpg',
 'test2.jpg']
```

In [367]:
```
def get_most_similar(test_img_path):
    test_img_vec = get_activation(test_img_path)

    # do dot prod
    test_img_vec = test_img_vec.reshape(len(test_img_vec),1)
    a = np.dot(activations, test_img_vec)

    # transform scores
    results = pd.DataFrame(a)
    results['filenames'] = adf['filename']
    results.columns = ['scores', 'filenames']
    results.sort_values('scores', ascending = False, inplace = True)
    results.head()

    # get matches
    matches = results['filenames'].values[:10]
    match_scores = results['scores'].values[:10]

    return matches
```
executed in 11ms, finished 04:26:18 2017-10-25

In [362]:

```
def plot_pic_grid(filenames):
    # set figsize
    fig = plt.figure()
    fig.set_size_inches((16,8))

    plotted = 0

    for c, r in enumerate(filenames):

        # get path to image file
        img_path_on_disk = path_data_train + r

        if len(img_path_on_disk) > 0:

            if plotted < 10:

                plotted+=1

                # plotting 10 images
                a = fig.add_subplot(2, 5, (plotted))
                img= plt.imread(img_path_on_disk)
                imgplot = a.imshow(img)
                a.set_title(match_scores[c])

                #print user_id, 'image plotted'
        else:
            # print user_id, 'no pic available'
            pass
```

executed in 12ms, finished 04:24:11 2017-10-25

In [371]:

```
[path_data_test + 'unknown/' + f for f in os.listdir(path_data_test + 'unknown/')]
```

executed in 5ms, finished 04:27:32 2017-10-25

Out[371]:

```
['/mnt/data/pycon/data/test/unknown/test6.jpg',
 '/mnt/data/pycon/data/test/unknown/test3.jpg',
 '/mnt/data/pycon/data/test/unknown/test7.jpg',
 '/mnt/data/pycon/data/test/unknown/test1.jpg',
 '/mnt/data/pycon/data/test/unknown/test8.jpg',
 '/mnt/data/pycon/data/test/unknown/test5.jpg',
 '/mnt/data/pycon/data/test/unknown/test4.jpg',
 '/mnt/data/pycon/data/test/unknown/test2.jpg']
```

In [375]:

```
test_img_path = '/mnt/data/pycon/data/test/unknown/test6.jpg'
plotimg(test_img_path)
plot_pic_grid(get_most_similar(test_img_path))
```

executed in 2.55s, finished 04:28:07 2017-10-25

In [374]:
```
test_img_path = '/mnt/data/pycon/data/test/unknown/test5.jpg'
plotimg(test_img_path)
plot_pic_grid(get_most_similar(test_img_path))
```

executed in 2.31s, finished 04:27:59 2017-10-25

Do PCA on activations

In [353]:
```
vec
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-353-fd0966cb4bb1> in <module>()
----> 1 vec

NameError: name 'vec' is not defined
```

In [257]:
```
ff = adf.copy()
del ff['filename']
```

executed in 5ms, finished 02:28:39 2017-10-25

In [264]:
```
n_points = 100
```

executed in 3ms, finished 02:30:11 2017-10-25

In [356]:
```
matches
```

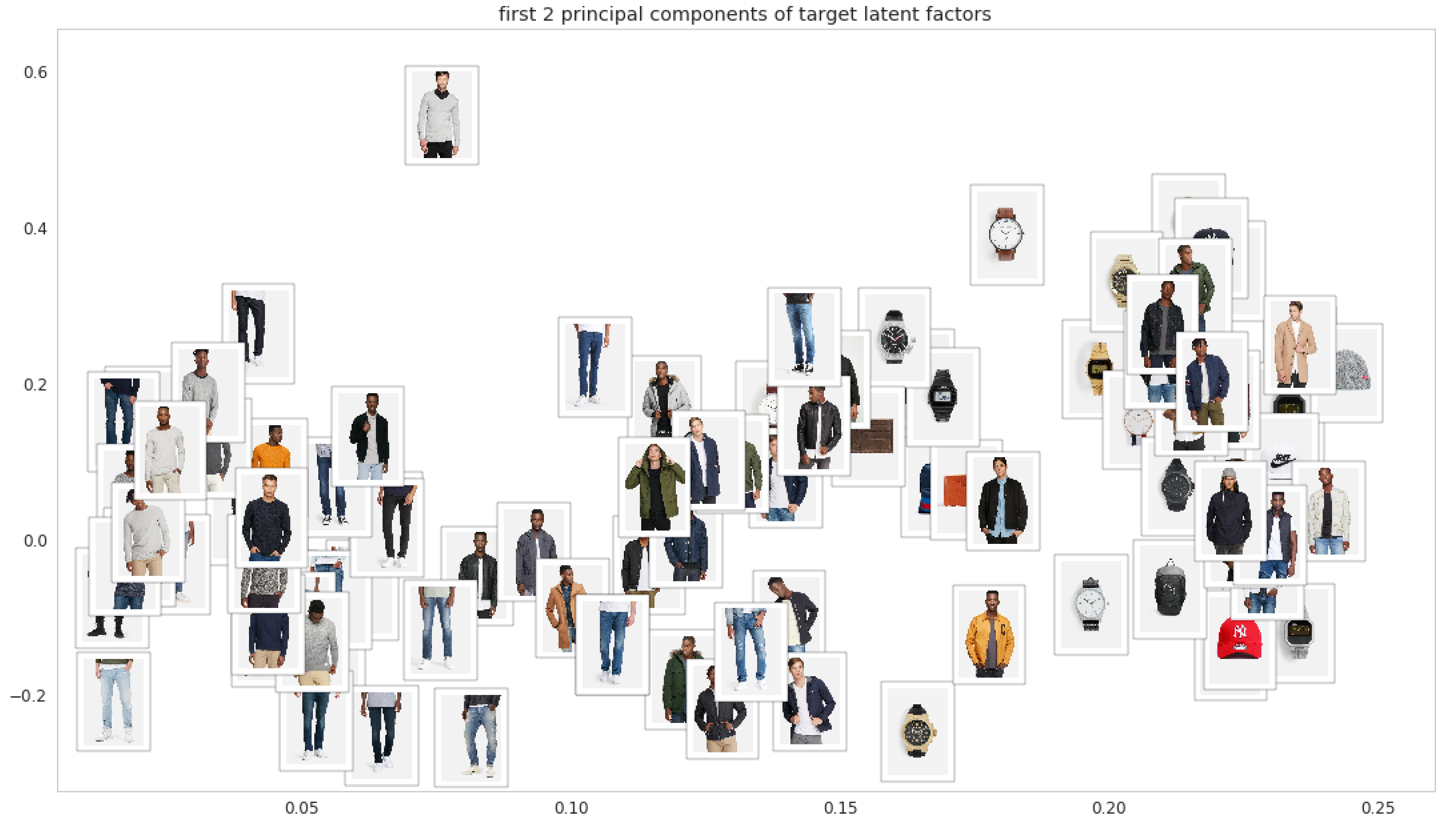
Out[356]:
```
array(['tees/productimg_3255.jpg', 'tees/productimg_4024.jpg',
       'tees/productimg_4242.jpg', 'tees/productimg_3588.jpg',
       'tees/productimg_3336.jpg', 'tees/productimg_3522.jpg',
       'tees/productimg_3213.jpg', 'tees/productimg_992.jpg',
       'tees/productimg_4155.jpg', 'tees/productimg_4238.jpg'], dtype=object)
```

In [293]:
```
#ff = f_targets[f_targets['country'] == 'BR']
#del ff['country']

from sklearn.decomposition import PCA
pca = PCA(n_components=8)
t_pcs = pca.fit(ff.values.T).components_
t_pcs.shape

# first 2 PCs
fac0 = t_pcs[0]
fac1 = t_pcs[1]

start=200
end = 350
X = fac0[start:end]
Y = fac1[start:end]

point_images = [path_data_valid + f for f in list(adf['filename'].values)]
```

executed in 36ms, finished 02:34:57 2017-10-25

In [294]:
```
# # https://stackoverflow.com/questions/4860417/placing-custom-images-in-a-plot-window-as-custom-data-markers-or-to-annotate-t
import matplotlib.pyplot as PLT
from matplotlib.offsetbox import AnnotationBbox, OffsetImage
from matplotlib._png import read_png

fig = PLT.gcf()
fig.clf()
ax = PLT.subplot(111)

fig.set_size_inches(18.5, 10.5)

# Plots an image at each x and y location.
def plotImage(x, y, im):

    if os.path.exists(im):

        img_content = plt.imread(im)
        imagebox = OffsetImage(img_content, zoom=.15)
        xy = [0.25, 0.45]

        ab = AnnotationBbox(imagebox, [x,y],
            xybox=(30., -30.),
            xycoords='data',
            boxcoords="offset points")
        ax.add_artist(ab)

for p in range(0,n_points):
    #print p
    plotImage(X[p],Y[p],point_images[p])


# # Set the x and y limits
ax.set_ylim(Y.min()*1.1,Y.max()*1.1)
ax.set_xlim(X.min()*1.1,X.max()*1.1)

plt.title('first 2 principal components of target latent factors')

plt.show()

#fig.savefig('pca.png', dpi=100)
```

executed in 1.77s, finished 02:34:59 2017-10-25



In [338]:

```
results.head()
```

Out[338]:

	scores	filenames	filename
2474	13,427	NaN	NaN
2788	13,379	NaN	NaN
3229	13,122	NaN	NaN
3126	13,009	NaN	NaN
3308	12,834	NaN	NaN

In [359]:

In [ ]:

```
def get_most_similar_products(test_img_idx):
    # plot test img
    vec_test.loc[test_img_idx]['filename']
    plotimg(pwd + '/data/test/' + vec_test.loc[test_img_idx]['filename'])

    # do dot prod
    test_img_vec = vec_test_num.loc[test_img_idx][:]
    test_img_vec = test_img_vec.reshape(len(test_img_vec),1)
    a = np.dot(vec_valid_num.ix[:], test_img_vec)

    # transform scores
    results = pd.DataFrame(a)
    results['filenames'] = vec_valid['filename']
    results.columns = ['scores', 'filenames']
    results.sort_values('scores', ascending = False, inplace = True)
    results.head()

    # get matches
    matches = results['filenames'].values[:10]
    match_scores = results['scores'].values[:10]

    plot_pic_grid(matches)
```

In [ ]: