Project Cardigan

KEYWORDS:
Deep Learning, Computer Vision, Visual Search, Image Retrieval, Distributed Systems, Recommender Systems, E-Commerce

Deep Learning Based Image Retrieval
2018 Mariusz Wołoszyn

Cardigan

Kardigan (ang. *cardigan*) – rodzaj swetra bez kołnierza, zapinanego z przodu na guziki lub zamek. Guziki w damskim kardiganie często są duże i stanowią element ozdobny.

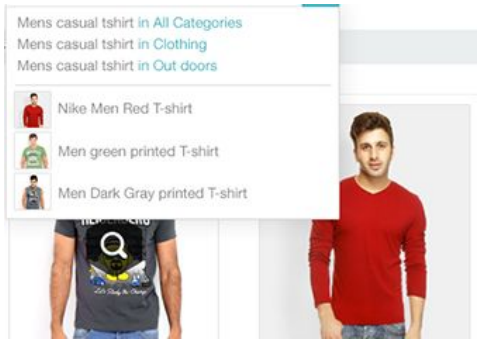
Nazwa pochodzi od tytułu angielskiego generała Jamesa Thomasa Brudenella (1797-1868) – *hrabiego Cardigan*, który podczas wojny krymskiej w bitwie pod Bałakławą poprowadził słynną szarżę lekkiej brygady kawalerii brytyjskiej.

Źródło: Wikipedia

Image Retrieval

Content-based image retrieval (**CBIR**), also known as **query by image content** (**QBIC**) and **content-based visual information retrieval** (**CBVIR**) is the application of computer vision techniques to the image retrieval problem, that is, the problem of searching for digital images in large databases. Content-based image retrieval is opposed to traditional **concept-based approaches** (see **Concept-based image indexing**).

Traditional e-commerce search



- Text based,
- Uses metadata (attributes, description)
- Works well for products with reach descriptions (e.g. electronics, books, movies, etc.)
- Fashion items lack detailed textual description
- Image-based techniques to the rescue:
  - Search
  - Visual recommendation

Challenges

(a) Similar catalog items with and without human model

(b) Concept based similarity across spooky printed t-shirts

(c) Detail based similarity via spacing and thickness of stripes

(d) Wild Image similarity across radically different poses

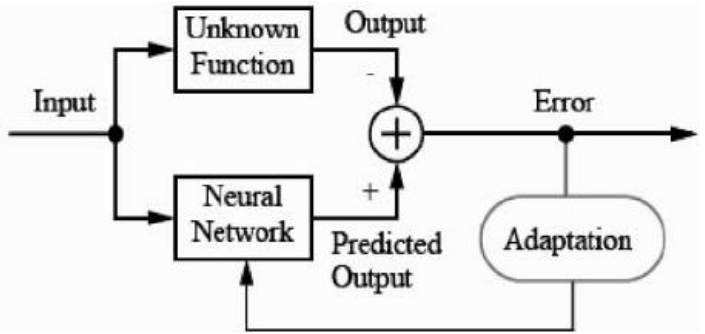
- Various notions of similarities,
- Different types of images (professional, wild/street),
- Different models or without a model,
- Different/poor lighting conditions,
- Different perspective,
- Different crops,
- Different postures, etc.

Źródło: Deep Learning based Large Scale Visual Recommendation and Search for E-Commerce

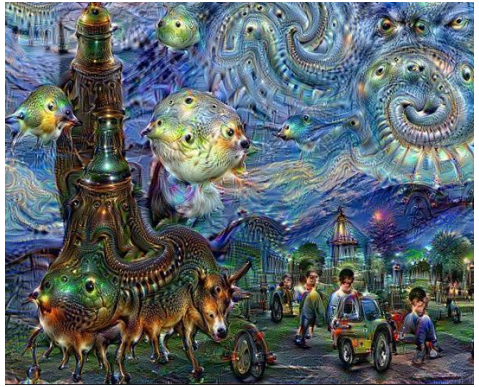
Neural networks to the rescue

The *universal approximation theorem* for neural networks states that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated arbitrarily closely by a multi-layer perceptron with just one hidden layer. This result holds only for restricted classes of activation functions, e.g. for the sigmoidal functions. (Wikipedia.org)

- Neural network as function approximator
- Deep NN can be used to “learn” practically any ineligible transformation:
  - image to vector (encoder)
  - vector to image (decoder)
  - image to text (description)
  - text to number (regression)
  - ...
- Can learn directly
- Can learn indirectly

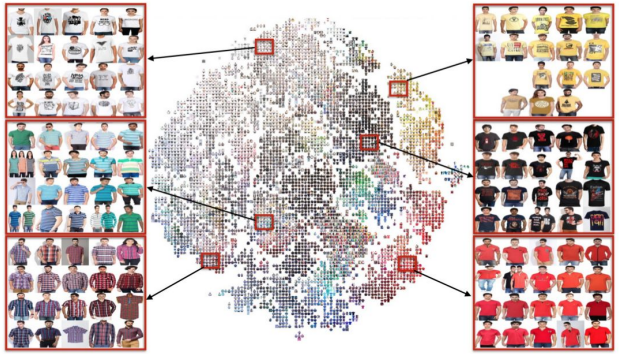


Let’s dream the solution

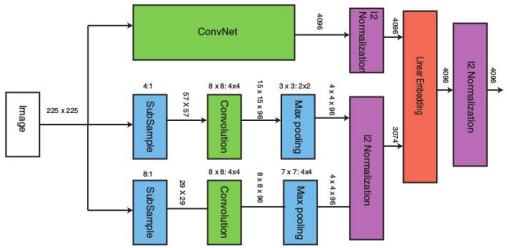


Let’s imagine a function with following properties:

- Converts image into vector (compression) that represents a point in vector space.
- Vectors representing similar object s are close to each other
- Vectors representing dissimilar objects are apart

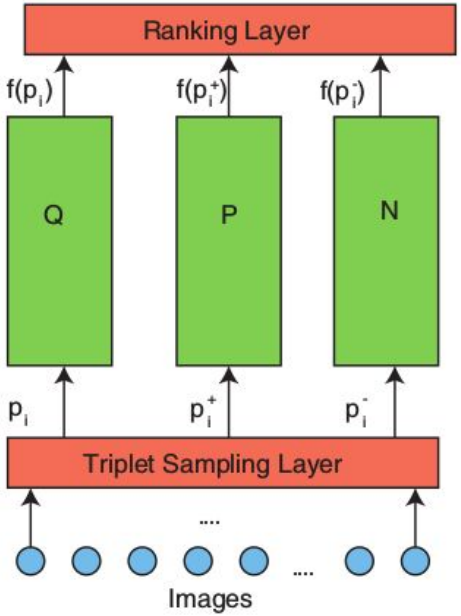


Architecture



- Inspired by siamese network architectures, proposed by Ailon et al.
- Described in 2014 in paper: Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. 2014. Learning Fine-Grained Image Similarity with Deep Ranking. In Proc. CVPR. 1386–1393.
- Triplet based supervised similarity information learning
- DCNN + Shallow path
- Hinge Loss to learn feature embeddings

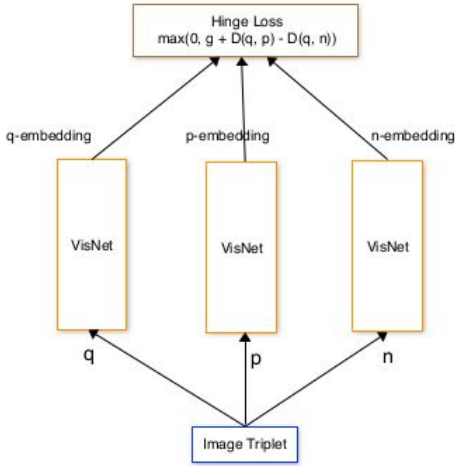
VisNet

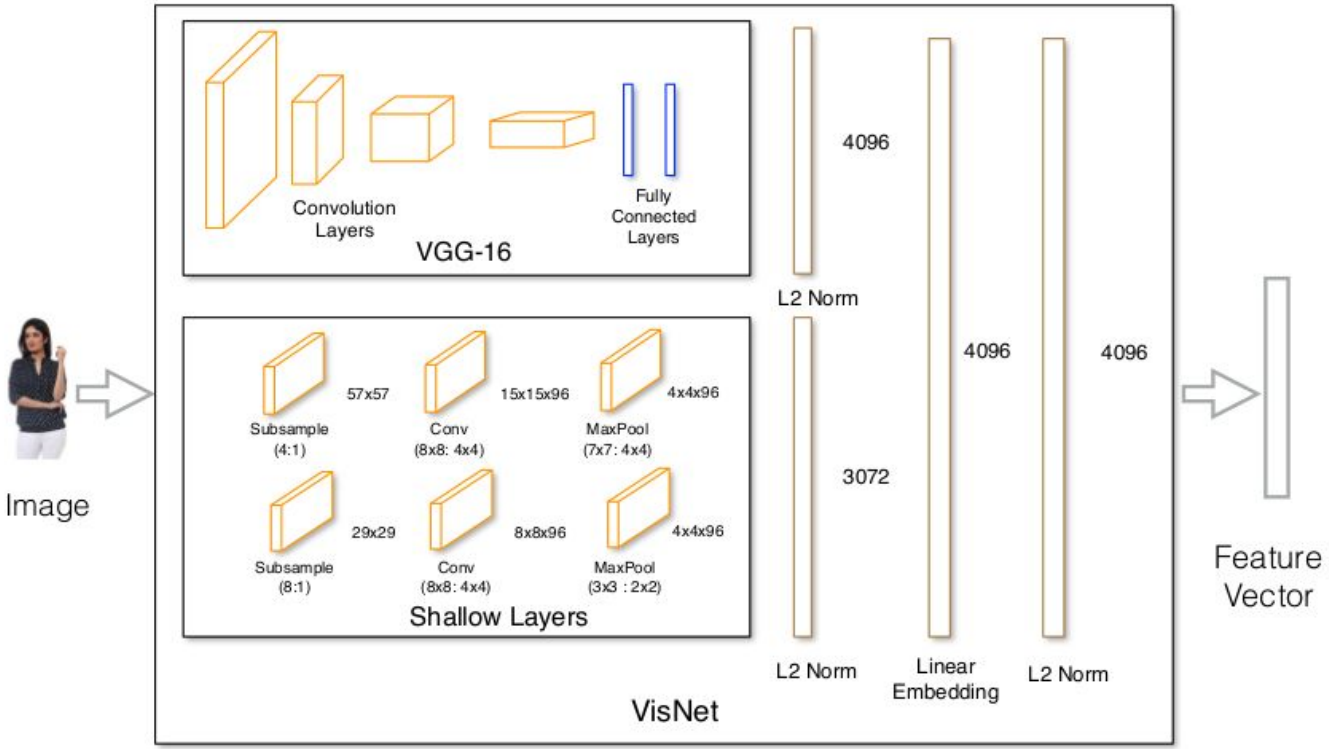


- Deep Learning solution Published in march 2017 by Fipkart, India's largest e-commerce company with over 100 million users making several thousand visits per second.
- Works on 50M products and supports 2K queries per second.
- With over 100K additions/deletions/modifications per hour.
- Generates 26% Conversion Rate (as opposed to an average of 8-10 % from the other modules).

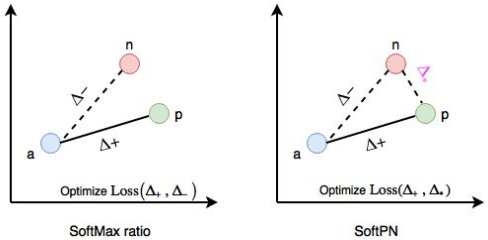
https://github.com/flipkart-incubator/fk-visual-search/

Architecture





Recap



1. Take 3 images:
   a. Q-query
   b. P-positive/similar
   c. N-negative/dissimilar
2. Compress them using our NN into feature vector of 512 length
3. Compute the euclidean distance among the vectors
4. Compute Hinge Loss
5. If negative image is closer in the feature space than the positive one then hinge loss is greater than 0 thus we can obtain a gradient from it and propagate back to the network.

There are other ways to express the loss function like SoftPN where both query and positive samples are as far as possible from negative.

Cardigan Prototype

- [Exact Street2Shop Dataset]
- Keras VGG16 based, simplified model
- Custom loss function
- Recurrent process of learning (close negatives selection)

Add our own fully connected layers on top of VGG. Instead of 4096x4096 use smaller 1024 classifier.

```
from keras.layers import (Input, Conv2D, MaxPool2D, Lambda, Flatten,
                          Concatenate, Reshape, BatchNormalization, Dense,
                          Dropout,
                         )
from keras.models import Model
from keras import regularizers
from keras import backend as K

# load model
model = VGG16(weights='imagenet')

input_ = model.layers[0].output # model.get_layer('input').output

flatten = model.get_layer('flatten').output

fc6=Dense(1024, activation='relu')(flatten)
drop6=Dropout(0.4)(fc6)

fc8=Dense(512, activation='relu')(fc6)

fc8_norm = BatchNormalization()(fc8)
```

Build complete ranking architecture

Once the VisNet like model is complete it's time to create complete Deep Ranking solution using 3 VisNets and Hinge Loss function

The Loss function is defined as follows:

$$L = max(0, g + D(\vec{q}, \vec{p}) - (\vec{q}, \vec{n}))$$

where $D(\vec{x}, \vec{y})$ denotes the Euclidean Distance between $\vec{x}$ and $\vec{y}$

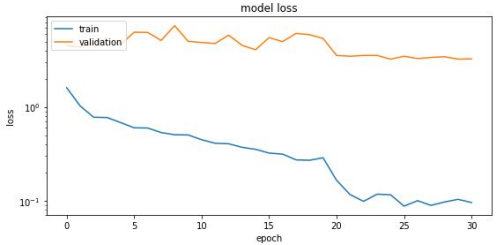
We'll use `hinge_loss3_op()` to calculate it. The input will be a tensor of the following shape:

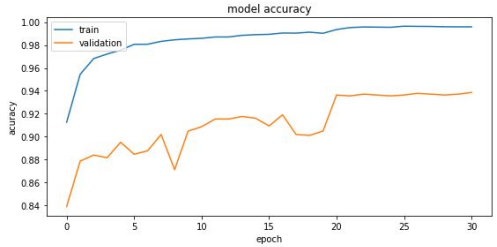
```
(None, 3, N)
```

where N is the embedding feature vector length (512 in our current implementation).
3 represents: `q`, `p` and `n` vectors respectively.

```
from keras import backend as K
# The actual loss op
def hinge_loss3_op(y_true, y_pred):
    q = y_pred[:, 0]
    p = y_pred[:, 1]
    n = y_pred[:, 2]

    g = 0.05

    qp_distance = K.sqrt(K.sum(K.square(q - p), axis=-1))
    qn_distance = K.sqrt(K.sum(K.square(q - n), axis=-1))

    return K.maximum(0., g + qp_distance - qn_distance)
```

Overall model



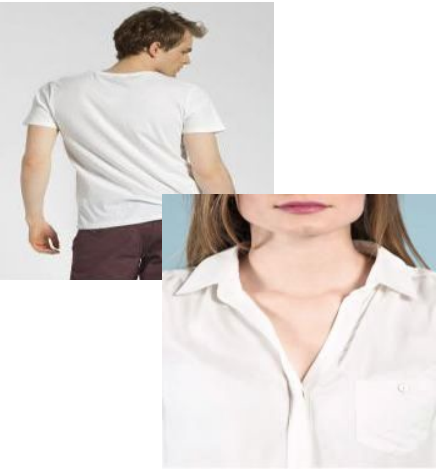



```
Epoch 1/1
1178/1178 [==============================] - 735s - loss: 1.6064 - hinge_accuracy: 0.9126 - val_loss:
4.5300 - val_hinge_accuracy: 0.8391
Epoch 2/2
1178/1178 [==============================] - 729s - loss: 1.0275 - hinge_accuracy: 0.9545 - val_loss:
4.3757 - val_hinge_accuracy: 0.8787
Epoch 3/3
1178/1178 [==============================] - 730s - loss: 0.7793 - hinge_accuracy: 0.9682 - val_loss:
4.8373 - val_hinge_accuracy: 0.8840
Epoch 4/4
1178/1178 [==============================] - 729s - loss: 0.7732 - hinge_accuracy: 0.9721 - val_loss:
5.1069 - val_hinge_accuracy: 0.8817
Epoch 5/5
1178/1178 [==============================] - 734s - loss: 0.6832 - hinge_accuracy: 0.9756 - val_loss:
4.6312 - val_hinge_accuracy: 0.8952
Epoch 6/6
1178/1178 [==============================] - 734s - loss: 0.6015 - hinge_accuracy: 0.9806 - val_loss:
6.2822 - val_hinge_accuracy: 0.8847
Epoch 7/7
1178/1178 [==============================] - 730s - loss: 0.5978 - hinge_accuracy: 0.9807 - val_loss:
6.2499 - val_hinge_accuracy: 0.8877
Epoch 8/8
1178/1178 [==============================] - 738s - loss: 0.5350 - hinge_accuracy: 0.9832 - val_loss:
5.1330 - val_hinge_accuracy: 0.9019
Epoch 9/9
1178/1178 [==============================] - 730s - loss: 0.5063 - hinge_accuracy: 0.9845 - val_loss:
7.3766 - val_hinge_accuracy: 0.8713
Epoch 10/10
1178/1178 [==============================] - 727s - loss: 0.5041 - hinge_accuracy: 0.9853 - val_loss:
5.8277 - val_hinge_accuracy: 0.9049
```

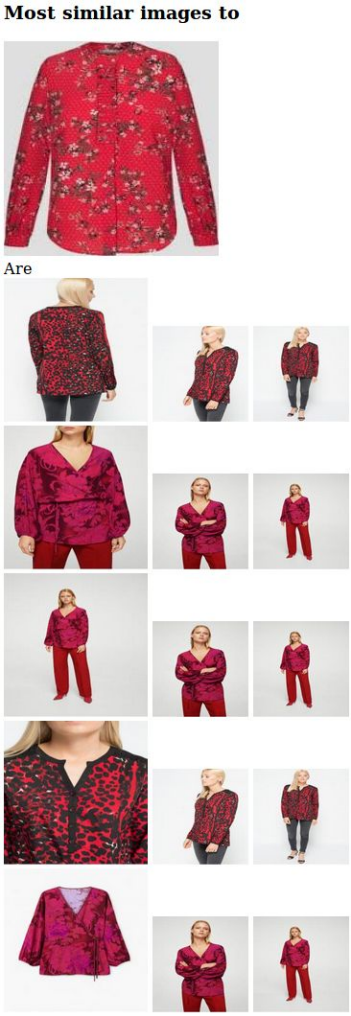
Layer (type)	Output Shape	Param #	Connected to
imput_q (InputLayer)	(None, 224, 224, 3)	0	
imput_p (InputLayer)	(None, 224, 224, 3)	0	
imput_n (InputLayer)	(None, 224, 224, 3)	0	
SimNet (Model)	(None, 512)	42559744	imput_q[0][0] imput_p[0][0] imput_n[0][0]
concatenate_3 (Concatenate)	(None, 1536)	0	SimNet[1][0] SimNet[2][0] SimNet[3][0]
reshape_1 (Reshape)	(None, 3, 512)	0	concatenate_3[0][0]

Total params: 42,559,744
Trainable params: 27,835,456
Non-trainable params: 14,724,288

Problems and obstacles



1. Data issues
    - Ground truth errors
    - Misleading samples (back-facing subjects)
    - Too-fine detail closeups
    - Lack of detail photos (too generic)
2. Model issues
    - Lack of gradient
    - Overfitting

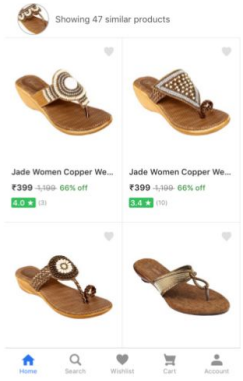


Retrieval part

- GPU for feature extraction (image dataset import)
- CPU Scikit-learn k-NN based retrieval (using Ball Tree for high dimensionality data rather than KDTree)
- Flask based PoC demo server

Use cases

(a) Product Page View



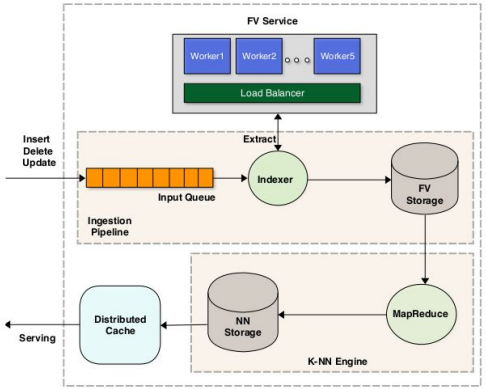
(b) Visual Recommendations

- Image search (retrieval)
- Recommendation engine/platform
- Double-listing detection
- Other domain applications

Work to be done

1. More data sources
2. Better data preprocessing:
    - filtering of misleading and wrong training samples (possibly with other CDNN models trained for detecting most common problems)
    - object detection (bounding boxes),
    - background removal (object masking)
3. Manual verification of final ~50K triplets

Work to be done



1. Data processing pipeline:
   - Design and implement target production pipeline (horizontally scalable, stable and cost effective)
   - Design for dataset insertion/update/deletion
   - Cropping and object detection on datasets and query
2. User and customer facing:
   - Robust and flexible API
   - UI for image retrieval (web/mobile)
   - Recommendation platform