Super Easy Way of Building Image Search with Keras

Introduction of Deep Learning for Information Retrieval Researchers

Hiroki Tanioka
Tokushima University
Center for Administration of Information Technology
tanioka.hiroki@tokushima-u.ac.jp

ABSTRACT

This paper provides detailed suggestions to create an Image Search Engine with Deep Learning. There are still few attempts with Deep Learning on a search engine. Here is a good idea of an extremely easy way of building an image search with Elasticsearch and Keras on Jupyter Notebook. So, it is demonstrated how an image search engine can be created where Keras is used to extract features from images, and Elasticsearch is used for indexing and retrieval. To demonstrate how easily this can be achieved, Jupyter notebook is used with two open source libraries, Keras and Elasticsearch. Then, the image search engine can search relevant images by images or keywords. In our approach, keras is regarded as analysis which is a process of converting an image into tokens or terms which are added to the inverted index for searching.

KEYWORDS

Image Search, Elasticsearch, Keras, Jupyter Notebook

ACM Reference format:
Hiroki Tanioka. 2016. Super Easy Way of Building Image Search with Keras. In *Proceedings of SIGIR '17 LIARR Workshop, Shinjuku, Tokyo, Japan, August 11, 2017,* 3 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In this paper, I demonstrate how an Image Search Engine can be created where Keras [5] is used to extract features from images, and Elasticsearch [1] is used for indexing and retrieval. To demonstrate how easily this can be achieved, we use Jupyter Notebook [3] and import two open source libraries, Keras and Elasticsearch. Jupyter notebook provides a web-based application in Python. Keras is an open source library containing neural networks APIs. Elasticsearch is an open source search engine based on Lucene. If you are an information retrieval researcher or engineer, you could begin quickly to make an image search engine with deep learning. Hence, let us introduce an extremely easy way of making a tiny deep learning search engine with Keras and Elasticsearch on Jupyter notebook.

2 COMPONENTS

Keras and Elasticsearch are combined on Jupyter notebook. The Jupyter notebook is an open-source web application. We can demonstrate and share easily our image search engine code on Jupyter notebook.

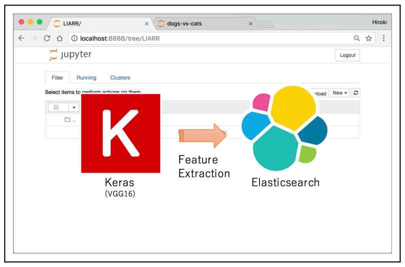


Figure 1: Kearas and Elasticsearch on Jupyter.

Keras is a neural networks API written in Python and capable of running on TensorFlow, CNTK, or Theano. Also, Keras contains some pre-trained models and sample datasets. Elasticsearch is employed for an image search engine. Keras is used as pre-processing for extracting features from images so that Elasticsearch can index images.

3 FEATURES

Image features are extracted by VGG16 [6] model on Keras. VGG team achieved an excellent result with VGG16 model in ILSVRC-2014 [2] competition. Image features are $1,000$ kinds of ImageNet classes(synsets) which are employed by VGG16 model. Image features which really mean ImageNet class tags are used as word features. Therefore, it is easy to modify using deep image representations or other deep features.

Table 1 shows the features of VGG16 model. The VGG16 model for Keras is pre-trained, which model of the 16-leyers network is used by the VGG team in the ILSVRC-2014. Keras with the VGG16 model calculates and returns tuple of class(synset) and probability from an image. There are pre-defined 1,000 classes. These classes are used as features for a feature vector on Elasticsearch.

Table 1: Examples of VGG16 features.

No.	synset ID	synset Name
0	n01440764	tench
1	n01443537	goldfish
2	n01484850	great_white_shark
3	n01491361	tiger_shark
4	n01494475	hammerhead
5	n01496331	electric_ray

4 ALGORITHMS

Bag-of-Visual-Words is employed for an index of a full-text search engine. Similarity is based on inner product of two vectors. A vector is query image vector is constituted of image features. Another vector is indexed image vector is also constituted of image features. Index schema is shown in Figure 2. A document contains *filename*

```
doc = {
  'filename': filename,
   'synset': {
    (synset Name): {
      'wind':(synset ID),
      'words':(synset Name),
      'score':(float)
    }
  }
}
```

Figure 2: Index schema using synset features.

and *synset* which consists of *wind*, *words*, and *score* from predicted information by Keras. Then, search engine calculates relevance scores based on inner product as similarity between images. So, relevance score is as follows,

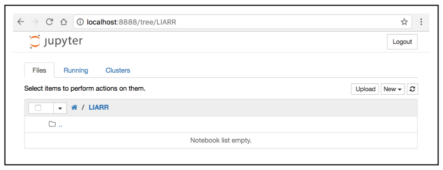
$$sim(\eta, \xi_i) = \sum_{i,j=1}^{n} \eta_j \cdot \xi_{i,j} \tag{1}$$

where *j* is the number of synset feature and *n* is 1,000 as the amount of synset feature. η_j is the *j*th synset feature score of a query image. $\xi_{i,j}$ is the *j*th synset feature score of an indexed image.

5 TOOLS

Jupyter notebook can be started easily as follows,

```
$ jupyter notebook
```



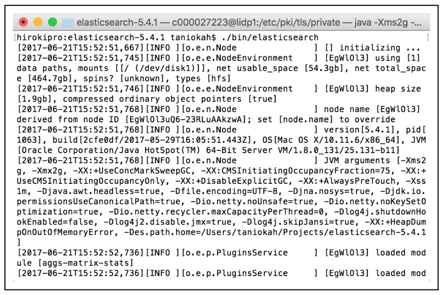
Select items to perform actions on them.

Notebook list empty.

Figure 3: Jupyter notebook.

If you can see the page like Figure 3, Jupyter notebook is ready to work. Elasticsearch also can be started easily on another terminal.

```
$ ./bin/elasticsearch
```



```
elasticsearch-5.4.1 — c000027223@lidp1:/etc/pki/tls/private — java -Xms2g -...
hirokipro:elasticsearch-5.4.1 taniokah$ ./bin/elasticsearch
[2017-06-21T15:52:51,667][INFO ][o.e.n.Node               ] [] initializing ...
[2017-06-21T15:52:51,745][INFO ][o.e.e.NodeEnvironment    ] [EgWlOl3] using [1]
data paths, mounts [[/ (/dev/disk1)]], net usable_space [54.3gb], net total_spac
e [464.7gb], spins? [unknown], types [hfs]
[2017-06-21T15:52:51,746][INFO ][o.e.e.NodeEnvironment    ] [EgWlOl3] heap size
[1.9gb], compressed ordinary object pointers [true]
[2017-06-21T15:52:51,768][INFO ][o.e.n.Node               ] node name [EgWlOl3]
derived from node ID [EgWlOl3uQ6-23RLuAAkzwA]; set [node.name] to override
[2017-06-21T15:52:51,768][INFO ][o.e.n.Node               ] version[5.4.1], pid[
1063], build[2cfe0df/2017-05-29T16:05:51.443Z], OS[Mac OS X/10.11.6/x86_64], JVM
[Oracle Corporation/Java HotSpot(TM) 64-Bit Server VM/1.8.0_131/25.131-b11]
[2017-06-21T15:52:51,768][INFO ][o.e.n.Node               ] JVM arguments [-Xms2
g, -Xmx2g, -XX:+UseConcMarkSweepGC, -XX:CMSInitiatingOccupancyFraction=75, -XX:+
UseCMSInitiatingOccupancyOnly, -XX:+DisableExplicitGC, -XX:+AlwaysPreTouch, -Xss
1m, -Djava.awt.headless=true, -Dfile.encoding=UTF-8, -Djna.nosys=true, -Djdk.io.
permissionsUseCanonicalPath=true, -Dio.netty.noUnsafe=true, -Dio.netty.noKeySetO
ptimization=true, -Dio.netty.recycler.maxCapacityPerThread=0, -Dlog4j.shutdownHo
okEnabled=false, -Dlog4j2.disable.jmx=true, -Dlog4j.skipJansi=true, -XX:+HeapDum
pOnOutOfMemoryError, -Des.path.home=/Users/taniokah/Projects/elasticsearch-5.4.1
]
[2017-06-21T15:52:52,736][INFO ][o.e.p.PluginsService     ] [EgWlOl3] loaded mod
ule [aggs-matrix-stats]
[2017-06-21T15:52:52,736][INFO ][o.e.p.PluginsService     ] [EgWlOl3] loaded mod
```

Figure 4: Elasticsearch server.

6 CODE

Keras should be imported as feature extraction library on the Jupyter notebook like Figure 7. Elasticsearch should be imported as search engine on the Jupyter notebook like Figure 6. The default value

```python
from keras.applications.vgg16 import VGG16, preprocess_input, decode_predictions
from keras.preprocessing import image
import numpy as np
import sys

model = VGG16(weights='imagenet')

def predict(filename, featuresize):
    img = image.load_img(filename, target_size=(224, 224))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    preds = model.predict(preprocess_input(x))
    results = decode_predictions(preds, top=featuresize)[0]
    return results
```

Figure 5: Keras library is imported and declaration of predict method.

is 1,000 for *total_fields.limit* as the limit number of fields in Elasticsearch. However, the index schema of our image search engine needs more than 1,000 fields. So *total_fields.limit* should be set as an adequate number.

```python
import os
from path import Path
from elasticsearch import Elasticsearch

es = Elasticsearch(host='localhost', port=9200)

def loadimages(directory):
    imagefiles = []
    for file in os.listdir(directory):
        filepath = os.path.join(directory, file)
        imagefiles.append(filepath)
    return imagefiles

def createindex(indexname):
    if es.indices.exists(index=indexname):
        es.indices.delete(index=indexname)
        es.indices.create(index=indexname, body={
            "index.mapping.total_fields.limit": 10000
        })

def indexfiles(directory, featuresize=10):
    imagefiles = loadimages(directory)
    for i in range(len(imagefiles)):
        filename = imagefiles[i]
        indexfile(filename, i, featuresize)
    es.indices.refresh(index="image-search")

def indexfile(filename, i, featuresize):
    doc = {'filename': filename, 'synset':{}}
    results = predict(filename, featuresize)
    for result in results:
        synset = doc['synset']
        synset[result[1]] = {
            'wnid': result[0], 'words': result[1], 'score': float(str(result[2]))
        }
    count = es.count(index='image-search', doc_type='image')['count']
    res = es.index(index="image-search", doc_type='image', id=(count+i), body=doc)

    createindex("image-search")
```

Figure 6: Elasticsearch library is imported and declaration of indexing methods.

These example code contains image search functions to only image file. However, if text query is needed to search among indexed images, the image search engine is extensible by indexing with synset Name.

These codes are put on our Github[1] up to share our idea. There are an instruction and some example codes of the proposed image search engine with VGG16. These instruction and example codes are written in Python on Jupyter notebook.

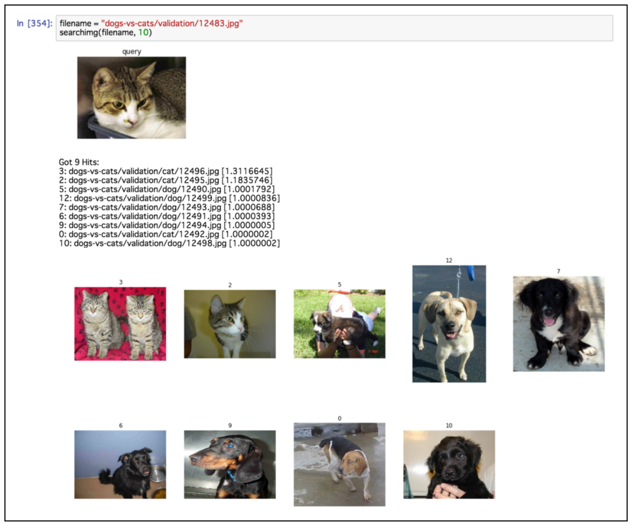
[1] https://github.com/taniokah/liarr2017/

```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

def showimg(filename, title, i):
    im = Image.open(filename)
    im_list = np.asarray(im)
    plt.subplot(2, 5, i)
    plt.title(title)
    plt.axis("off")
    plt.imshow(im_list)

def searchimg(filename, num):
    plt.figure(figsize=(20, 10))
    for i in range(1):
        showimg(filename, "query", i+1)
    plt.show()
    results = predict(filename, num)
    search(results, num)

def search(synsets, num):
    inline = "1.000";
    for synset in synsets:
        words = synset[1]
        score = synset[2]
        inline += " + doc['synset." + words + ".score'].value * " + str(score)
    res = es.search(index="image-search", doc_type='image', body={
        "query": {
            "function_score": {
                "query": {"match_all": {}},
                "script_score" : {"script" : {"inline": inline}}
        }}})

    print("Got %d Hits:" % res['hits']['total'])
    for h in res['hits']['hits'][0:num]:
        print "%s: %s [%s]" % (h["_id"], h["_source"]["filename"], h["_score"])
    plt.figure(figsize=(20, 10))

    for i in range(len(res['hits']['hits'][0:num])):
        hit = res['hits']['hits'][0:num][i]
        showimg(hit["_source"]["filename"], hit["_id"], i+1)
    plt.show()
```

Figure 7: Search method is implemented.

7 DEMOS

Kaggle dogs-vs-cats dataset [4] should be downloaded on the machine. The indexing function is composed of Keras and Elasticsearch, which indexes dog and cat images as a search target. Keras plays a role of feature extraction. Then, Elasticsearch has an index for image file retrieval. Figure 8 shows an example for searching a cat image among indexed cat and dog images. Here, some sim-

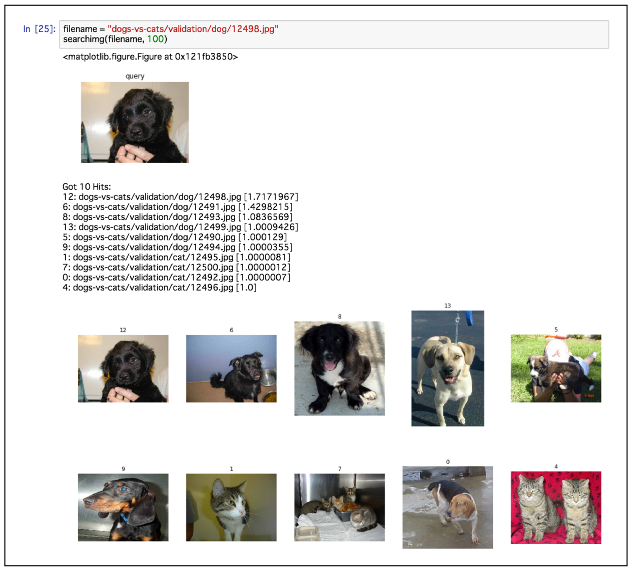


```
In [354]: filename = "dogs-vs-cats/validation/12483.jpg"
          searchimg(filename, 10)
```

```
Got 9 Hits:
3: dogs-vs-cats/validation/cat/12496.jpg [1.3116645]
2: dogs-vs-cats/validation/cat/12495.jpg [1.1835746]
5: dogs-vs-cats/validation/dog/12490.jpg [1.0001792]
12: dogs-vs-cats/validation/dog/12499.jpg [1.0000836]
7: dogs-vs-cats/validation/dog/12493.jpg [1.0000688]
6: dogs-vs-cats/validation/dog/12491.jpg [1.0000393]
9: dogs-vs-cats/validation/dog/12494.jpg [1.0000005]
0: dogs-vs-cats/validation/cat/12492.jpg [1.0000002]
10: dogs-vs-cats/validation/dog/12498.jpg [1.0000002]
```

Figure 8: Search similar cats with a cat.

ilar cats are retrieved by the image search from a cat picture as query. Similarity score is listed with each file path. If you search a



```
In [25]: filename = "dogs-vs-cats/validation/dog/12498.jpg"
         searchimg(filename, 100)
```

```
<matplotlib.figure.Figure at 0x121fb3850>
```

```
Got 10 Hits:
12: dogs-vs-cats/validation/dog/12498.jpg [1.7171967]
6: dogs-vs-cats/validation/dog/12491.jpg [1.4298215]
8: dogs-vs-cats/validation/dog/12493.jpg [1.0836569]
13: dogs-vs-cats/validation/dog/12499.jpg [1.0009426]
5: dogs-vs-cats/validation/dog/12490.jpg [1.000129]
9: dogs-vs-cats/validation/dog/12494.jpg [1.0000355]
1: dogs-vs-cats/validation/cat/12495.jpg [1.0000081]
7: dogs-vs-cats/validation/cat/12500.jpg [1.0000012]
0: dogs-vs-cats/validation/cat/12492.jpg [1.0000007]
4: dogs-vs-cats/validation/cat/12496.jpg [1.0]
```

Figure 9: Example of searching a dog.

dog image among indexed cat and dog images, the search engine shows the dog image on the top of retrieved images as shown in Figure 9. Eventually, this image search engine seems to work well as intended.

8 CONCLUSION

Information Retrieval researchers and engineers are easy to understand an image search system using VGG16 image features made by Deep Learning. If you would like to challenge deep image representations and higher precision, the system architecture allows to improve for them, because it is a standard architecture of lucene based search engine. The proposed image search engine is very useful and has potential for expansion. VGG16 model can be changed to other models. Additionally, if you scale the index, you might use TF-IDF score or BM25 on sparse indexes. Obviously, this proposed system can make an index with deep image representations or other deep features instead of VGG16 image features.

ACKNOWLEDGEMENT

I appreciate my colleagues in Tokushima University who advised and gave consultations. And, I thank my IR friends who provided insight and expertise that greatly assisted this research.

REFERENCES

[1] elastic. 2017. Elasticsearch: RESTful, Distributed Search & Analytics. (2017). https://www.elastic.co/ [Online; accessed 19-June-2017].
[2] ImageNet. 2014. Large Scale Visual Recognition Challenge (ILSVRC). (2014). http://www.image-net.org/challenges/LSVRC/
[3] Jupyter. 2017. The Jupyter Notebook. (2017). http://jupyter.org/ [Online; accessed 19-June-2017].
[4] Kaggle. 2014. Dogs vs. Cats. (2014). https://www.kaggle.com/c/dogs-vs-cats
[5] keras. 2014. Keras: The Python Deep Learning library. (2014). https://keras.io/ [Online; accessed 19-June-2017].
[6] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). http://arxiv.org/abs/1409.1556