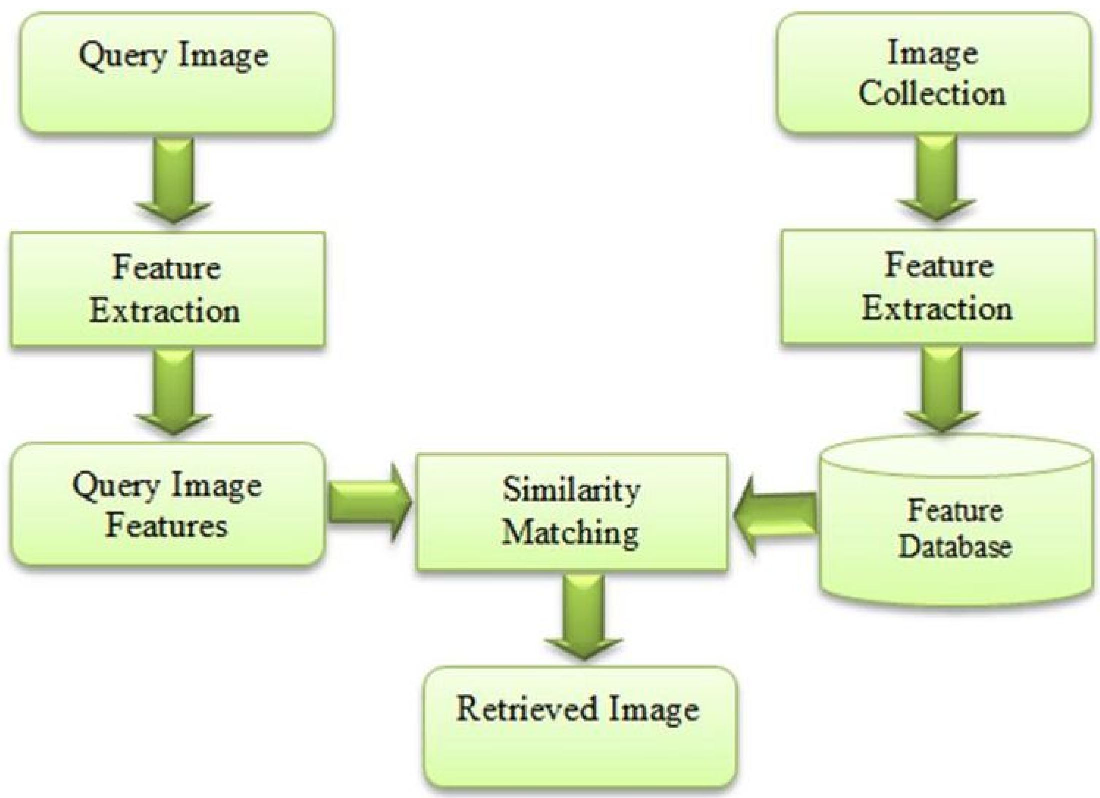
SUMMER INTERNSHIP REPORT

Content Based Image Retrieval

Architecture of CBIR systems



https://images.app.goo.gl/h9q1rmQhRWJpS5Uj9

Pragya Paramita Pal

18.06.2019
Industrial Internship

Table of Contents

Acknowledgement

Abstract

Introduction

Content-based image retrieval, also known as query by image content and content-based visual information retrieval, is the application of computer vision techniques to the image retrieval problem, that is, the problem of searching for digital images in large databases.

Our final goal is to have a search engine that can take in images and output either similar images or tags, and take in text and output similar words, or images.

The three methods of image querying and retrieval are:
- ❏ Searching for **similar images to an input image** (Image → Image)
- ❏ Searching for **similar words to an input word** (Text → Text)
- ❏ Generating **tags for images**, and **searching images using tex**t (Image ↔ Text)

It is easy to get confused between CBR and CBIR. The differences are elucidated below:

Content Based Image Retrieval (CBIR)	Case Based Reasoning (CBR)
❖ Retrieval of images based on a given query	❖ The process of solving new problems based on the solutions of similar past problems
❖ Semantic based image retrieval	❖ Knowledge-based. Follows the process of retrieve, revise, retain.
❖ Structured by images only	❖ Structured by cases
❖ Static database	❖ Dynamic database
❖ Context-dependant	❖ Context-modelling

Feature extraction for CBIR was originally started using singular features like colour, texture or shape. A method for colour based extraction is RGB histogram analysis. A method for texture based extraction is Gabor filter. Methods for shape based extraction are edge histogram analysis or HOG (histogram of gradient). Since 2012 however, Deep Learning has slowly started overtaking classical methods and are now more popular now using the VGG net or Residual net model. The reasons for this shift are discussed later.

System specifications

GPU: GK208B [GeForce GT 730]
CUDA Driver Version = 9.1
Ubuntu 18.04.2 LTS
RAM: 8GB
OS type: 64-bit
Processor: Intel® Core™ i5-6402P CPU @ 2.80GHz × 4

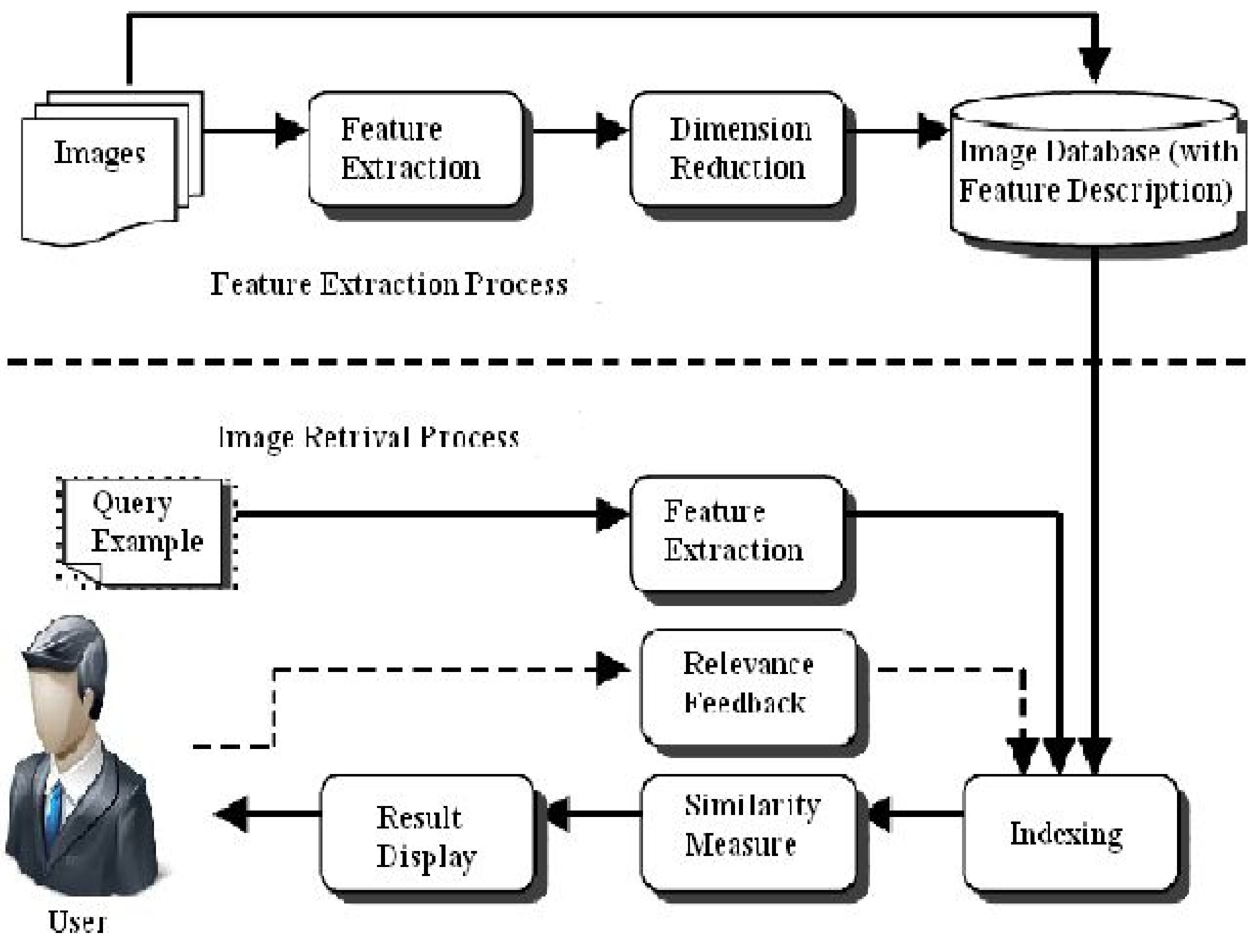
Software Configuration

cuDNN and tensorRT installed
Python requirements: numexpr==2.6.9 numpy==1.16.3 scipy==1.2.1 matplotlib==3.0.3 scikit-learn==0.15.0 pandas==0.4.0 sympy==1.4 scikit-image==0.15.0 pylint==2.3.1 PyYAML==5.1 Pillow==6.0.0 jupyter==1.0.0 Flask==1.0.2 flask_cors==3.0.7 gunicorn==19.9.0 Cython==0.29.7 h5py==2.9.0 easydict==1.9 python-dotenv==0.10.2

IPython[all]==7.5.0 imgaug==0.2.9
Python requirements for AI: tensorflow-gpu==1.9.0 Keras, version>= 2.0.8
Extra requirements: annoy==1.15.2 pyquery==1.4.0 cloudpickle==1.1.1 xmltodict==0.12.0 gpxpy==1.3.5 loky==2.4.2 pyproj==2.13 pytest==45.0 python-dateutil==2.8.0 repoze.lru==0.7 Jinja2==2.10.1 Pymongo==3.8.0 ExifRead==2.1.2 numba==0.43.1
Created a python 3 virtualenvwrapper

CBIR Concept and Workflow

Content Based Image Retrieval Workflow



https://images.app.goo.gl/msqBQT9AuYWaMKkX7

We observe that there are two main parts-- feature extraction and image matching.

Feature extraction process:

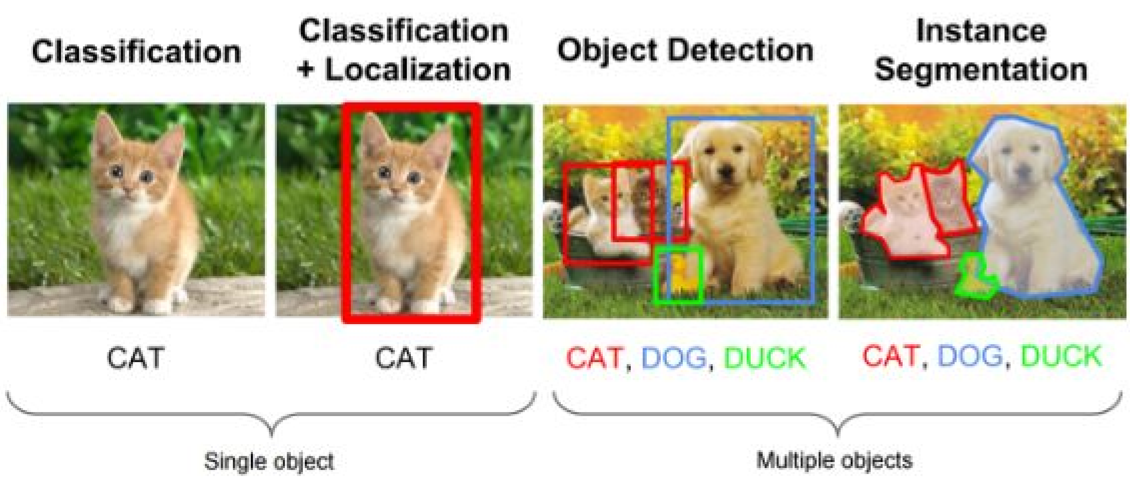
Computer Vision Tasks

What is 'Computer Vision'?

Computer Vision is an interdisciplinary field of science that aims to make computers process, analyze images and videos and extract details in the same way a human mind does.

Earlier Computer Vision was meant only to mimic human visual systems until we realized how AI can augment its applications and vice versa. We may also not realize this every day but we are being assisted by the applications of Computer Vision in automotive, retail, banking and financial services, healthcare, etc.

The various computer vision tasks, shown comparatively



https://towardsdatascience.com/object-localization-in-overfeat-5bb2f7328b62

Object classification: It is the process of broadly recognizing what is being shown in the image. It does not involve recognizing each object individually but only recognizing the major one and classifying it as such.

Object localization: It is the task of predicting an object as well as its bounding box. Usually it is limited to images with single object.

Object detection: The next task is an extension of the above two activities so as to make it useful real-world issues. In multiple object images it becomes necessary to recognize each object individually as well as draw a bounding box around each object for localization.

Semantic segmentation: It is the task of associating each pixel of an image with a class label. The most important thing to note is that semantic segmentation does not highlight individual instances of a class differently. For example, if there were 3 cows in an image, the model would highlight the area they occupy, but it will not be able to distinguish one cow from another.

Instance segmentation: This task does one better on semantic segmentation. It colours every instance of an object differently for distinguishing each separately. As the name suggests, the goal is to segment or separate each “instance” of a class in an image.

Deep Neural Network

I. Basic Concept of Deep Learning

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. Deep learning is getting lots of attention lately and for good reason. It’s achieving results that were not possible before.

In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labeled data and neural network architectures that contain many layers.

How does deep learning attain such impressive results?

In a word, accuracy. Deep learning achieves recognition accuracy at higher levels than ever before. This helps consumer electronics meet user expectations, and it is crucial for safety-critical applications like driverless cars. Recent advances in deep learning have improved to the point where deep learning outperforms humans in some tasks like classifying objects in images.

While deep learning was first theorized in the 1980s, there are two main reasons it has only recently become useful:

1. Deep learning requires large amounts of **labeled data**. For example, driverless car development requires millions of images and thousands of hours of video.
2. Deep learning requires substantial **computing power**. High-performance GPUs have a parallel architecture that is efficient for deep learning. When combined with clusters or cloud computing, this enables development teams to reduce training time for a deep learning network from weeks to hours or less.

II. Literature on some important CNN architectures

Deep Neural Networks (DNN) have greater capabilities for image pattern recognition and are widely used in Computer Vision algorithms. And, Convolutional Neural Network (**CNN**, or **ConvNet**) is a class of DNN which is most commonly applied to analyzing visual imagery. It is used not only in Computer Vision but also for text classification in Natural Language Processing (NLP). CNN translates to Convolutional Neural Networks which is a very popular algorithm for image classification and typically comprises of convolution layers, activation function layers, pooling (primarily max_pooling) layers to reduce dimensionality without losing a lot of features.

AlexNet, designed by the SuperVision group, including Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever from the University of Toronto, was the winner of the 2012 ImageNet LSVRC-2012 competition. ImageNet is a yearly competition focused on image classification, with an error rate of 15.3 per cent. AlexNet uses ReLu activation function instead of tanh to add non-linearity. ReLu function is given by: $f(x) = max(0,x)$

The advantage of the ReLu over sigmoid is that it trains much faster than the latter because the derivative of sigmoid becomes very small in the saturating region and therefore the updates to the weights almost vanish. Using ReLu accelerated the speed of training (by 6 times) and increased the accuracy. In the network, ReLu layer is put after each and every convolutional and fully-connected layers(FC).

Another problem that this architecture solved was reducing the over-fitting by using a Dropout layer that is applied only before the first and the second fully connected layer.

AlexNet is the first neural net that was capable of localization and object detection.

The winner of the ILSVRC 2014 competition was **GoogleNet**(a.k.a. **Inception V1**) from Google. It achieved a top-5 error rate of 6.67%! This was very close to human level performance which the organisers of the challenge were now forced to evaluate. As it turns out, this was actually rather hard to do and required some human training in order to beat GoogLeNets accuracy. After a few days of training, the human expert (Andrej Karpathy) was able to achieve a top-5 error rate of 5.1%(single model) and 3.6%(ensemble). The network used a CNN inspired by LeNet but implemented a novel element which is dubbed an inception module. It used batch normalization, image distortions and RMSprop. This module is based on several very small convolutions in order to drastically reduce the number of parameters. Their architecture consisted of a 22 layer deep CNN but reduced the number of parameters from 60 million (AlexNet) to 4 million.

VGGNet is invented by VGG (Visual Geometry Group) from University of Oxford, Though VGGNet is the 1st runner-up, not the winner of the ILSVRC (ImageNet Large Scale Visual Recognition Competition) 2014 in the classification task, which has significantly improvement over ZFNet (The winner in 2013) and AlexNet (The winner in 2012). And GoogLeNet is the winner of ILSVLC 2014. Nevertheless, VGGNet beats the GoogLeNet and won the localization task in ILSVRC 2014.

It makes the improvement over AlexNet by replacing large kernel-sized filters(11 and 5 in the first and second convolutional layer, respectively) with multiple 3X3 kernel-sized filters one after another. With a given receptive field(the effective area size of input image on which output depends), multiple stacked smaller size kernel is better than the one with a larger size kernel because multiple non-linear layers increases the depth of the network which enables it to learn more complex features, and that too at a lower cost.

For example, three 3X3 filters on top of each other with stride 1 have a receptive size of 7, but the number of parameters involved is 3*(9C^2) in comparison to 49C^2 parameters of kernels with a size of 7. Here, it is assumed that the number of input and output channel of layers is C.Also, 3X3 kernels help in retaining finer level properties of the image. The network architecture is given in the table.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/

The VGG convolutional layers are followed by 3 fully connected layers. The width of the network starts at a small value of 64 and increases by a factor of 2 after every sub-sampling/pooling layer. It achieves the top-5 accuracy of 92.3 % on ImageNet.

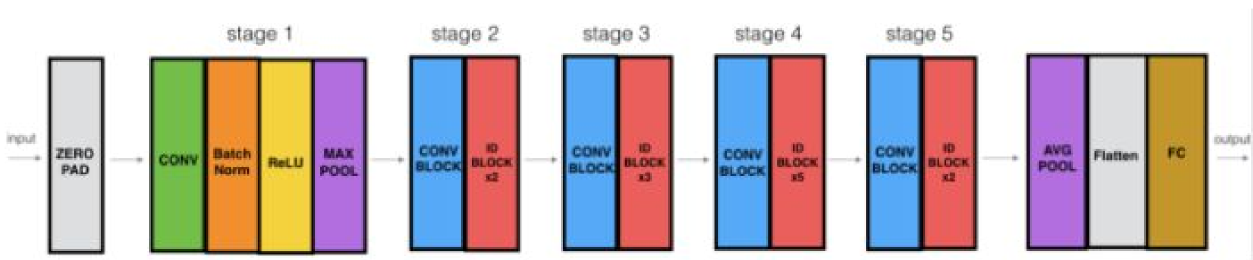
As per what we have seen so far, increasing the depth should increase the accuracy of the network, as long as overfitting is taken care of. But the problem with increased depth is that the signal required to change the weights, which arises from the end of the network by comparing ground-truth and prediction becomes very small at the earlier layers, because of increased depth. It essentially means that earlier layers are almost negligible learned. This is called **vanishing gradient.** In other words increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the notorious vanishing gradient problem — as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient extremely small. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly.

ResNet first introduced the concept of skip connection. In it the original input is added to the output of the convolution block. Why skip connections work here:

1. They mitigate the problem of vanishing gradient by allowing this alternate shortcut for gradient to flow through
2. They allow the model to learn an identity function which ensures that the higher layer will perform at least as good as the lower layer, and not worse

Was ResNet Successful?

- Won 1st place in the ILSVRC 2015 classification competition with top-5 error rate of 3.57% (An ensemble model)
- Won the 1st place in ILSVRC and COCO 2015 competition in ImageNet Detection, ImageNet localization, Coco detection and Coco segmentation.
- Replacing VGG-16 layers in Faster R-CNN with ResNet-101. They observed a relative improvements of 28%
- Efficiently trained networks with 100 layers and 1000 layers also.



ResNet-50 Model

https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

III. R-CNN, Fast R-CNN, Faster R-CNN and Mask R-CNN

The main problem with standard convolutional network followed by a fully connected layer is that the size of the output layer is variable — not constant, which means the number of occurrences of the objects appears in the image is not fixed. A very simple approach to solving this problem would be to take different regions of interest from the image and use a CNN to classify the presence of the object within that region.

Mainly RCNNs are important for object detection and semantic segmentation.

R-CNN or Regions with CNN features

When an image is input, nearly 2000 candidate region proposals are sent to a CNN that extracts features which are relayed to an SVM to classify the presence of an object in that region. Also, it gives an offset to increase the precision of the bounding box so that the region includes the whole object instead of a part.

However, so many computations make it very slow.

Fast R-CNN

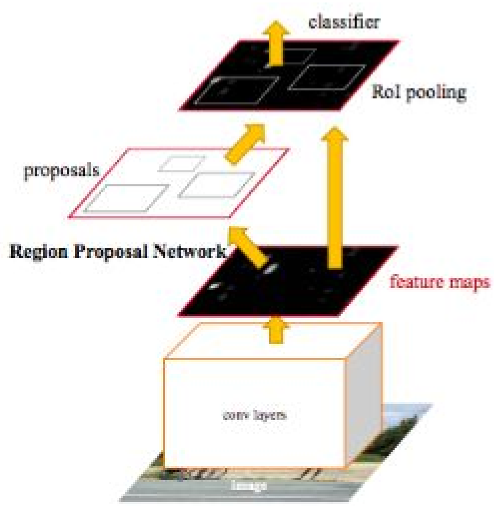
The same author came up with a faster algorithm, named Fast R-CNN. In it, we do not feed the proposed regions to the CNN. Instead we send the whole input image to generate a convolutional feature map. From the convolutional feature map, we identify the region of proposals and warp them into squares and by using a RoI pooling layer we reshape them into a fixed size so that it can be fed into a fully connected layer. From the RoI feature vector, we use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box.

It is faster in this case because the convolution operation is done only once per image and a feature map is generated from it.

Faster R-CNN

In both the above algorithms selective search is used to achieve region proposal. However, in this method, instead of using selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals. To understand the difference between the algorithms we've to explore two concepts

Faster R-CNN architecture



Credit: Original Research paper

How does selective search work? Selective search starts with over-segmenting an input image. The algorithm that follows is:

1. Add all bounding boxes corresponding to segmented parts to the list of regional proposals
2. Group adjacent segments based on similarity
3. Go to step 1

What is Region Proposal Network? To generate these so called "proposals" for the region where the object lies, a small network is to slide over a convolutional feature map that is the output by the last convolutional layer. RPN has a classifier and a regressor. They have introduced the concept of anchors. Anchor is the central point of the sliding window. Classifier determines the probability of a proposal having the target object. Regression regresses the coordinates of the proposals. Ultimately, RPN is like a lightweight neural network that scans the image in a sliding-window fashion and finds areas that contain objects. It also needs to be trained, so we definitely have its own loss function.

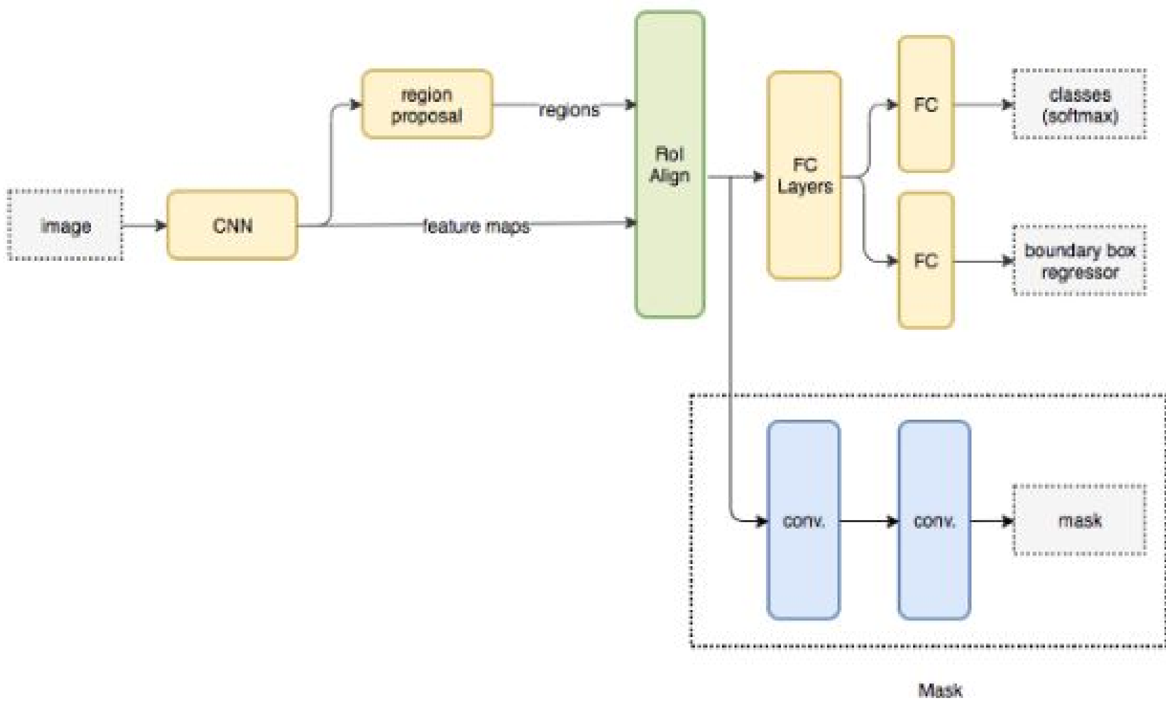
Mask R-CNN

Mask RCNN is a deep neural network aimed to solve instance segmentation problem in machine learning or computer vision.

The Faster R-CNN builds all the ground works for feature extractions and ROI proposals.

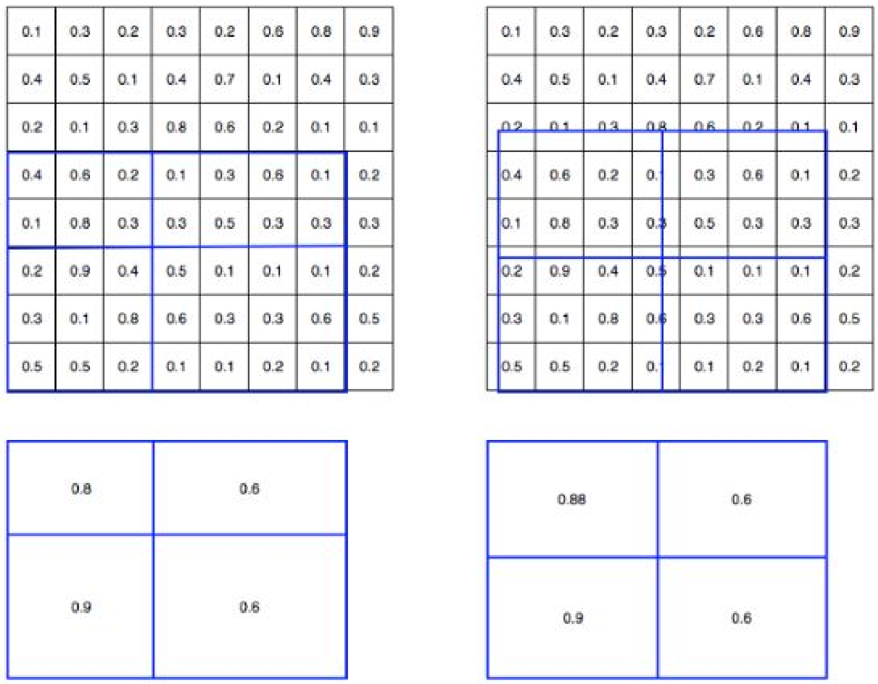
At first sight, performing image segmentation may require more detailed analysis to colorize the image segments. By surprise, not only we can use this model, the extra work required is pretty simple. After the ROI pooling, we add 2 more convolution layers to build the mask.

Mask R-CNN model



https://medium.com/@jonathan_hui/image-segmentation-with-mask-r-cnn-ebe6d793272

Another major contribution of Mask R-CNN is the refinement of the ROI pooling. In ROI, the warping is digitalized: the cell boundaries of the target feature map are forced to realign with the boundary of the input feature maps. Therefore, each target cell may not be in the same size. Mask R-CNN uses **ROI Align** which does not digitalize the boundary of the cells and make every target cell to have the same size (bottom right). It also applies interpolation to calculate the feature map values within the cell better. ROI align works as:



https://medium.com/@jonathan_hui/image-segmentation-with-mask-r-cnn-ebe6d793272

Why neural networks are required for best results in Object Detection

Feature extraction involves extracting information from raw pixel values, like shape, colour, texture,etc. These help in discriminating one category of images from the others. This is done in an unsupervised manner wherein only the algorithms are hard coded. Some of the traditional and most widely used methods used are GIST, HOG, SIFT and some others. After the features are extracted a classification module is used to find associated labels with the images. Some examples of these are SVM, Logistic Regression and Random Forest.

However, the algorithm doesn't adjust itself to different classes and images so if the feature that the algorithm extracts does not contain enough information to discriminate between the different images, then the accuracy of the classification model suffers.
One method to deal with this extract multiple features. This too, though, involves a lot of tedious tweaking of parameters for the domains.

The philosophy behind deep learning is no hard-coded feature extractor is built in. Feature extraction and classification are combined together into one module. Discriminating representations are extracted and classified based on supervised data.

CBIR Implementation

CBIR with DNN as Feature Extractor

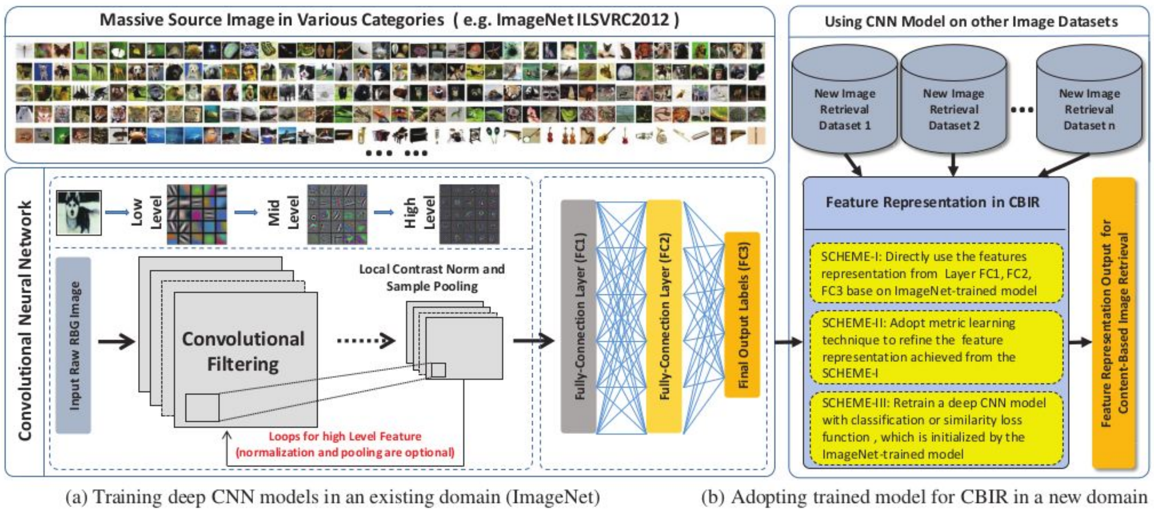


Figure 1: A Framework of Deep Learning with Application to Content-based Image Retrieval.

https://ink.library.smu.edu.sg/cgi/viewcontent.cgi?referer=&httpsredir=1&article=3320&context=sis_research

Semantic search

Introduction

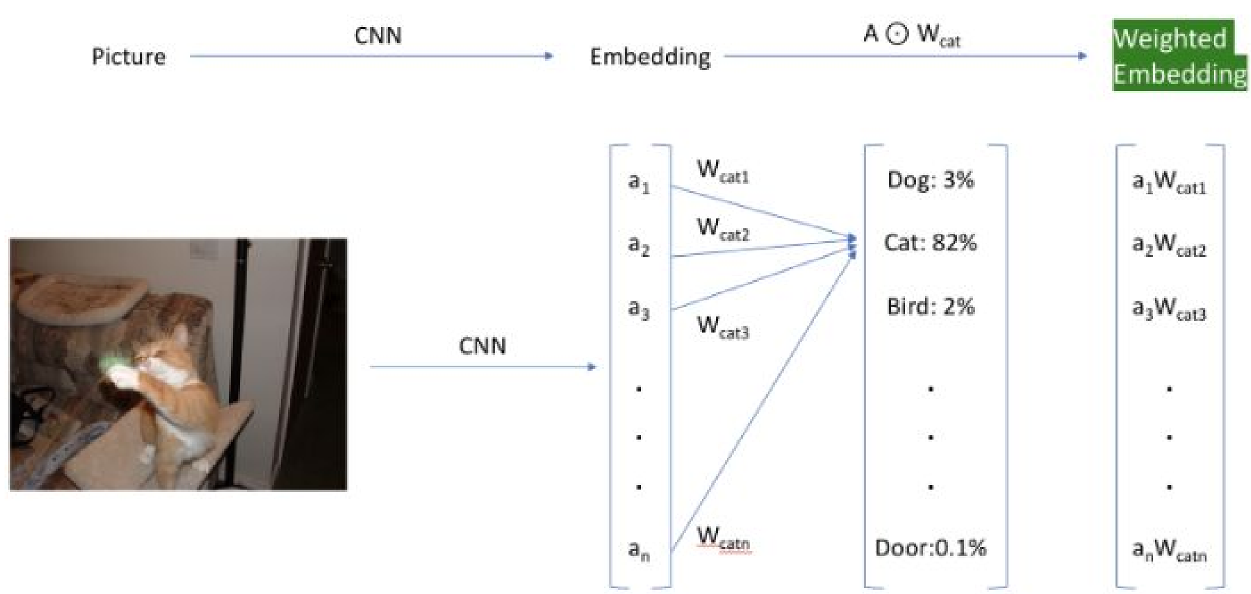
The Github repository reference: https://github.com/hundredblocks/semantic-search

The repository is essentially about creating an image search service. Since deep learning has the capacity to automatically **extract meaningful representations** when trained on a large enough dataset, VGG16 is used for feature extraction to obtain activations for the images. With these activations we will be trying to match the image to other such images. GloVe model is used for when text queries are used or a word to image relation is required to be established. All three types namely image to image and text to image query-results methods are used.

Our aim is to find expressive vector representations, or embeddings for the images. We find the embeddings for all the images in our dataset and store them in .npy file and store them to the disk for re-use, without needing to re-index. We can then calculate the similarity by finding how close the vectors are to one another. Annoy library is capable of building a fast index for the vectors. Cosine similarity is used to find the similarity--high similarity between images means high cosine similarity.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}}$$

For queries which are images and similar images have to be retrieved, we begin with a workflow that was using VGG16 to extract the features and produce a list of feature vectors. Other deep neural nets like ResNet can also be used for improved results, as mentioned before.

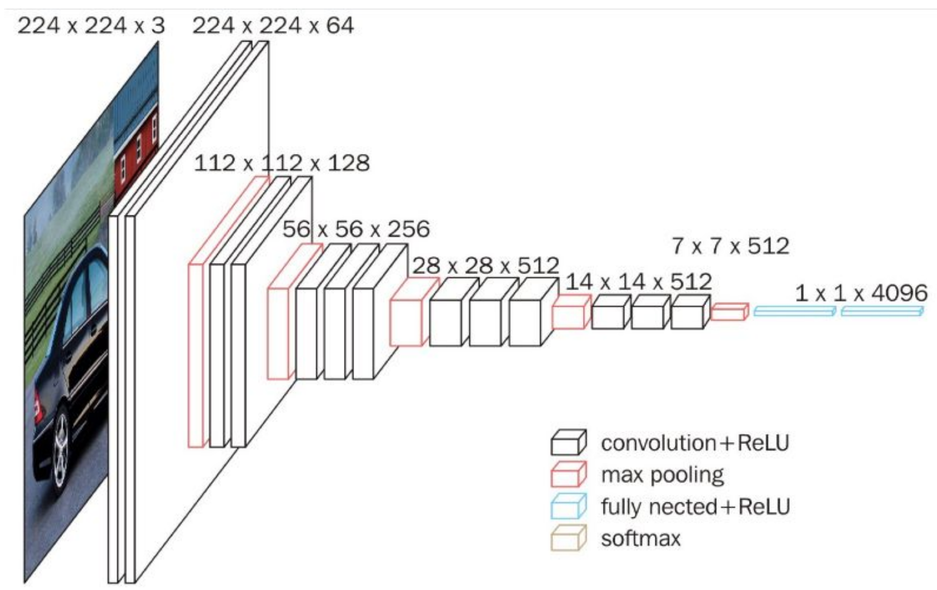


The hack to get weighted embeddings. The classification layer is shown for reference only.

https://blog.insightdatascience.com/the-unreasonable-effectiveness-of-deep-learning-representations-4ce83fc663cf

Here, we reweigh the activations. The weights of the last layer are weighed with weights of the index of the class we are looking for. For this reason we do not include the softmax layer of the CNN used-- VGG16 that was pre-trained on Imagenet. Softmax calculates a probability for every possible class, which is not required in our case. The last fully connected layer (FC1000) is also not taken. Hence after the FC4096 layer we receive feature vectors of length of 4096.

VGG16 model used [image from demo (Streamlit) in the repo]



Implementation

Running the pipeline

To get the demo running, firstly we have to clone the repo to a newly created folder after forking it. The command `git clone` along with the web URL.

Then we can download the dataset provided or use our own. If we wish to use their dataset we have to download it:

```
mkdir dataset

python downloader.py
```

Make sure all the categories having two or more words have an underscore in between the words.

Now we index the images in the dataset:

```
python search.py \

  --index_folder dataset \

  --features_path feat_4096 \

  --file_mapping index_4096 \

  --index_boolean True \

  --features_from_new_model_boolean False
```

The first argument specifies the folder to be indexed (dataset) and feat_4096.npy is the list of vectors stored for all the images in the dataset. Each image is mapped and the path is stored in index_4096.json as stated by the next argument. index_boolean True means that we are trying to index the dataset not search for similar images. The last argument specifies if features are to be created from a new model. In this case, it will remain False.

Now that indexing is done for image <-> image requests, we can search for similar images.

```
python search.py \

  --input_image dataset/cat/2008_001335.jpg \
```

```
  --features_path feat_4096 \
  --file_mapping index_4096 \
  --index_boolean False \
  --features_from_new_model_boolean False
```

The first argument-- input_image specifies the image we are querying and that we want images similar to that one. Here, index-boolean is going to be False because re-indexing is not required.

Now for word <-> word requests or word <-> image requests we will be needing another model called GloVe, an unsupervised learning algorithm. GloVe (Global Vectors for word representations) was trained on Wikipedia. Training is performed on aggregate global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

The training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. Owing to the fact that the logarithm of a ratio equals the difference of logarithms, this objective associates (the logarithm of) ratios of co-occurrence probabilities with vector differences in the word vector space. Because these ratios can encode some form of meaning, this information is encoded as vector differences as well.

To download the vector model:

```
curl -LO http://nlp.stanford.edu/data/glove.6B.zip
unzip glove.6B.zip
mkdir models
mkdir models/glove.6B
mv glove.6B.300d.txt models/glove.6B/
```

For our project we will be using the 300 dimension vectors in the GloVe model. For text <-> image queries we will need to tweak our VGG16 model. Two new layers will be added so as to be able to semantically map a word to an image. An

Now to index our images using the GloVe 300d model:

```
python search.py \
  --index_folder dataset \
  --features_path feat_300 \
  --file_mapping index_300 \
  --model_path my_model.hdf5 \
  --index_boolean True \
  --features_from_new_model_boolean True \
  --glove_path models/glove.6B
```

The last argument specifies which new model is being used to index the activations.

Now to search for an image using text:

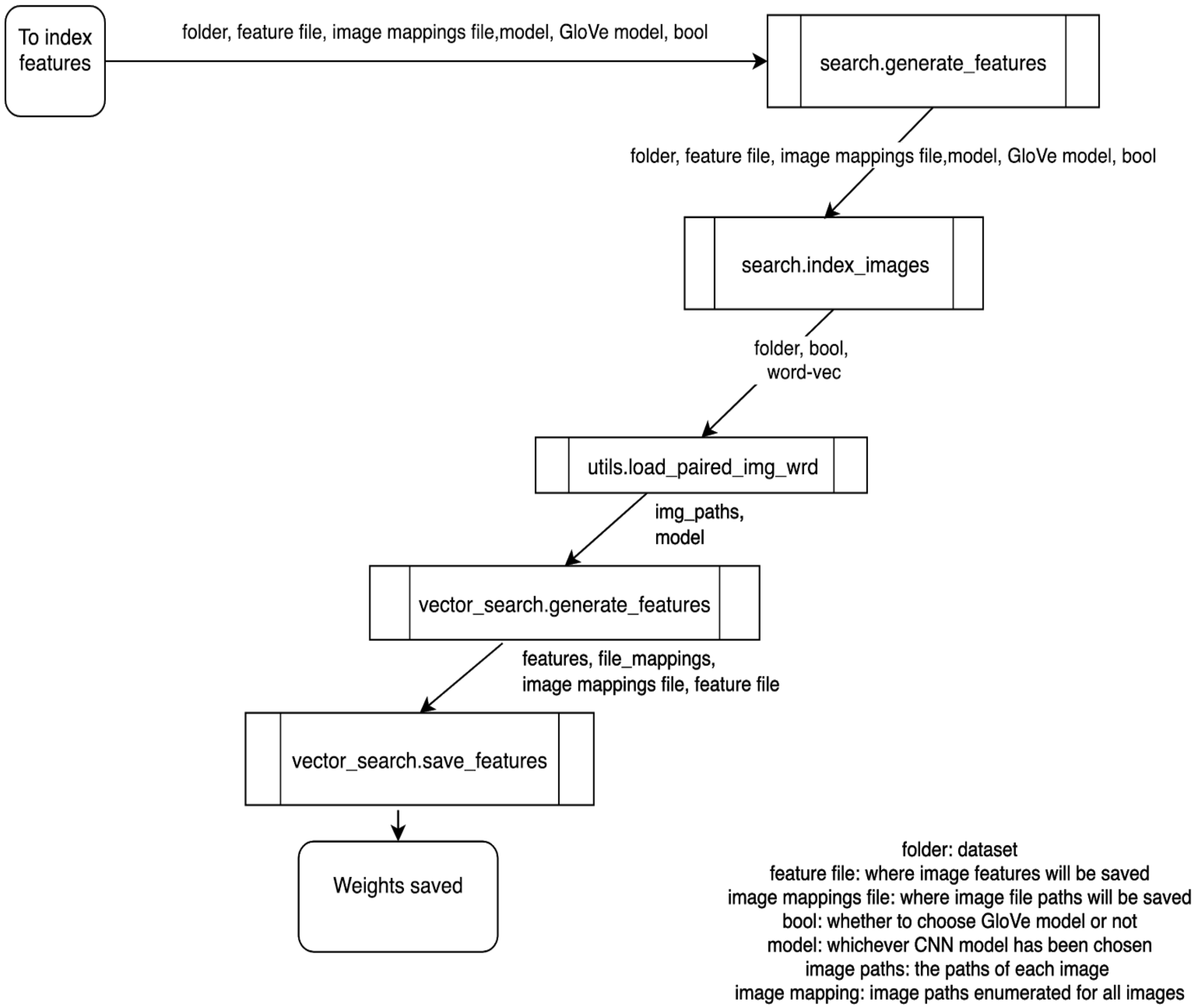
```
python search.py \
  --input_word cat \
  --features_path feat_300 \
  --file_mapping index_300 \
  --model_path my_model.hdf5 \
  --index_boolean False \
  --features_from_new_model_boolean True \
  --glove_path models/glove.6B
```

To be able to run the whole demo now:

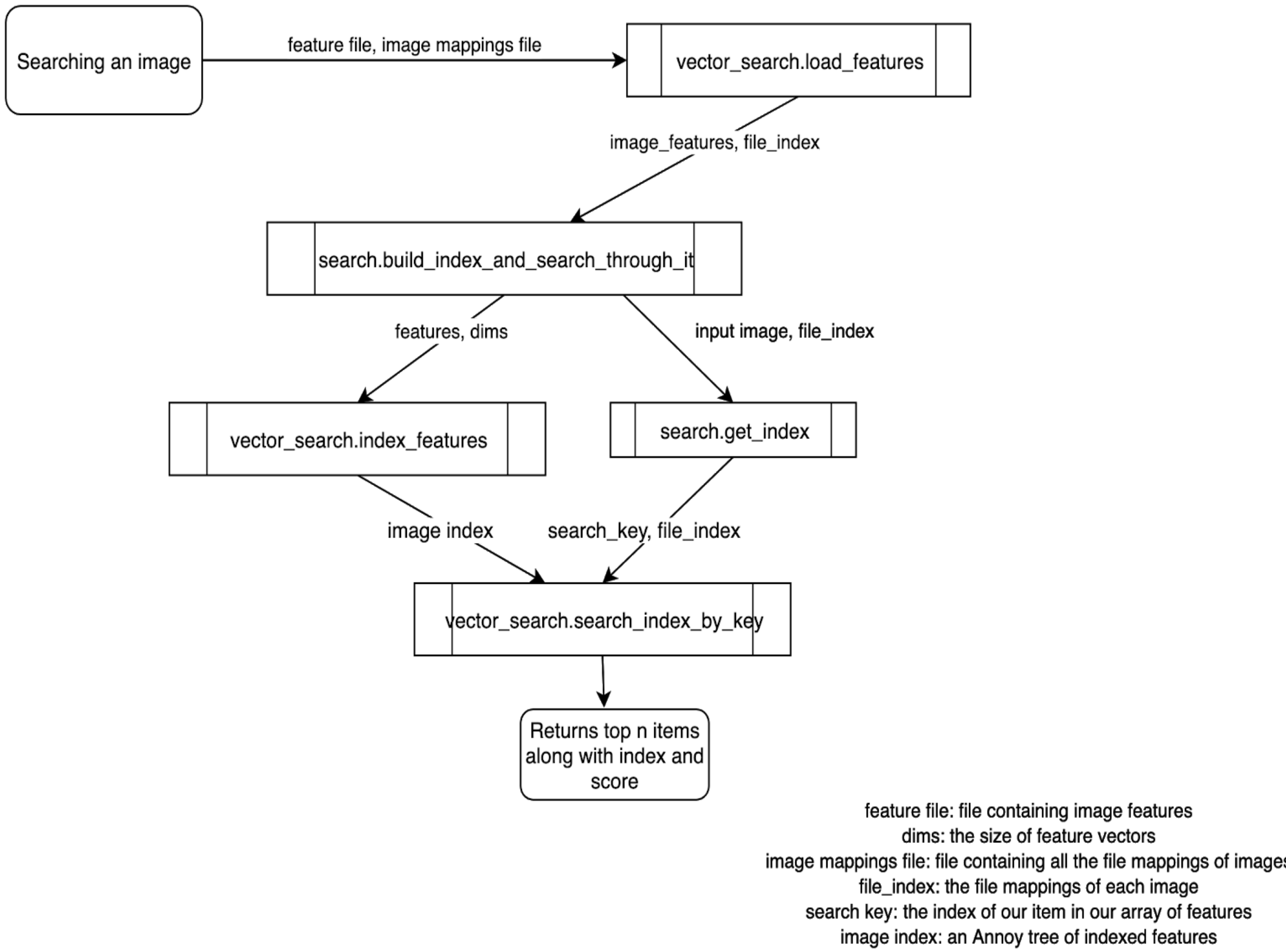
```
python demo.py \
  --features_path feat_4096 \
  --file_mapping_path index_4096 \
  --model_path my_model.hdf5 \
```

```
--custom_features_path feat_300 \
--custom_features_file_mapping_path index_300 \
--search_key 872 \
--train_model False \
--generate_image_features False \
--generate_custom_features False
```

For indexing the images, the detailed workflow involving the functions along with parameters are shown below:



Now once indexed, images can be searched and the top n (where n is changeable) and displayed. The detailed workflow for that is again shown below:



Customisation

Certain changes were made initially due to small cache size and added to search.py. The first one was to clear session each time the program is run:

```
from keras import backend as K K.clear_session()
```

Another was to increase GPU utilization:

```
import tensorflow as tf gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.5) sess = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options)) [to specify amount of GPU utilization]
```

The next step was to replace the dataset with another one:
http://www.vision.caltech.edu/Image_Datasets/Caltech101/101_ObjectCategories.tar.gz

The pipeline was run again to see if we could extract features similarly from this dataset and search using image or text queries. Results are displayed below.

The objective of my project is to be able to get suitable image features for image classification and being able to retrieve similar images. The dimensions of the features obtained have to be compatible so that they remain indexable using Annoy library.

Hence, the next step was to replace VGG16 with another CNN model. Now, Mask RCNN essentially uses ResNet50 or ResNet101 for image feature extractor. It builds the feature map with either of the models, hence, VGG16 is now replaced with a pre-trained model of ResNet50 loaded from Keras applications library. In vector_search.py the load_headless_pretrained_model() function has been edited to include:

```
resn_model = ResNet50(weights='imagenet',include_top=True)
model=Model(inputs=resn_model.inputs, outputs=resn_model.layers[-2].output)
```

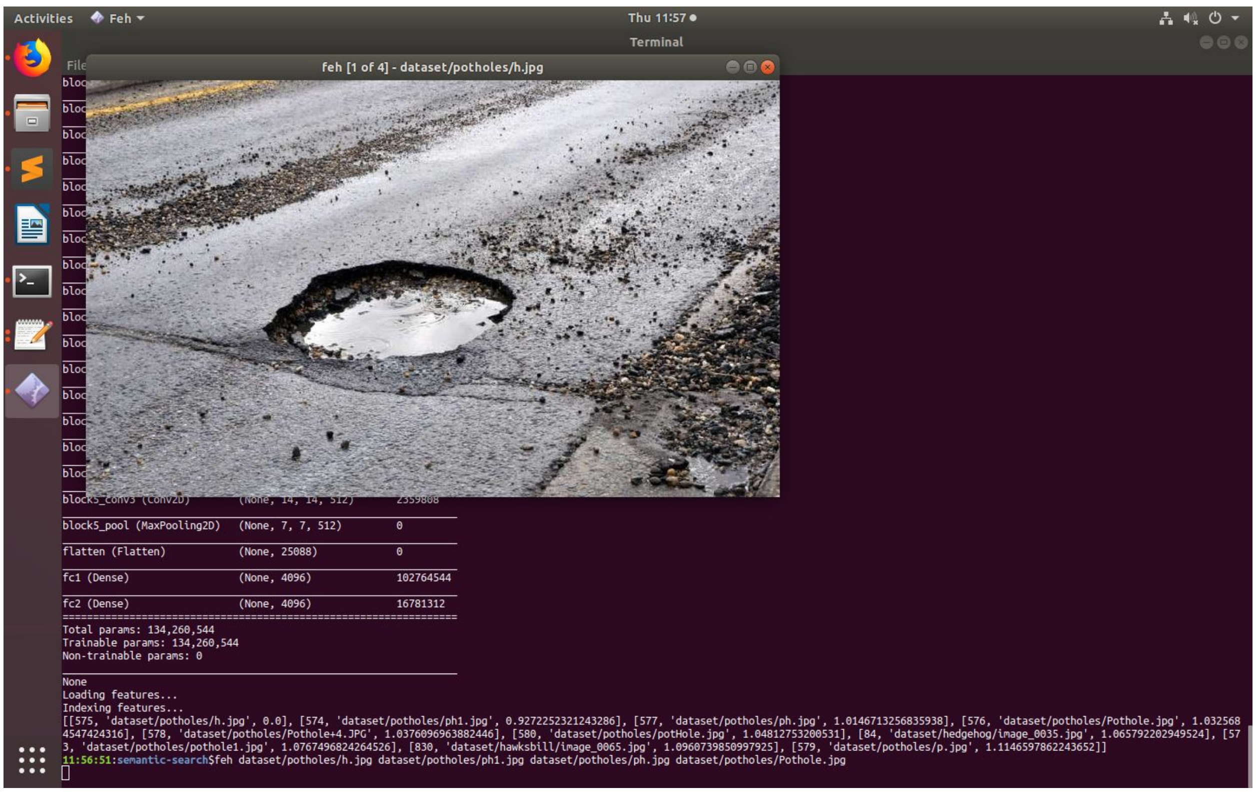
The model is pre-training on ImageNet. Flatten and Dense layers are removed. The output then, are features of length 2048 which are indexed after changing vector lengths.

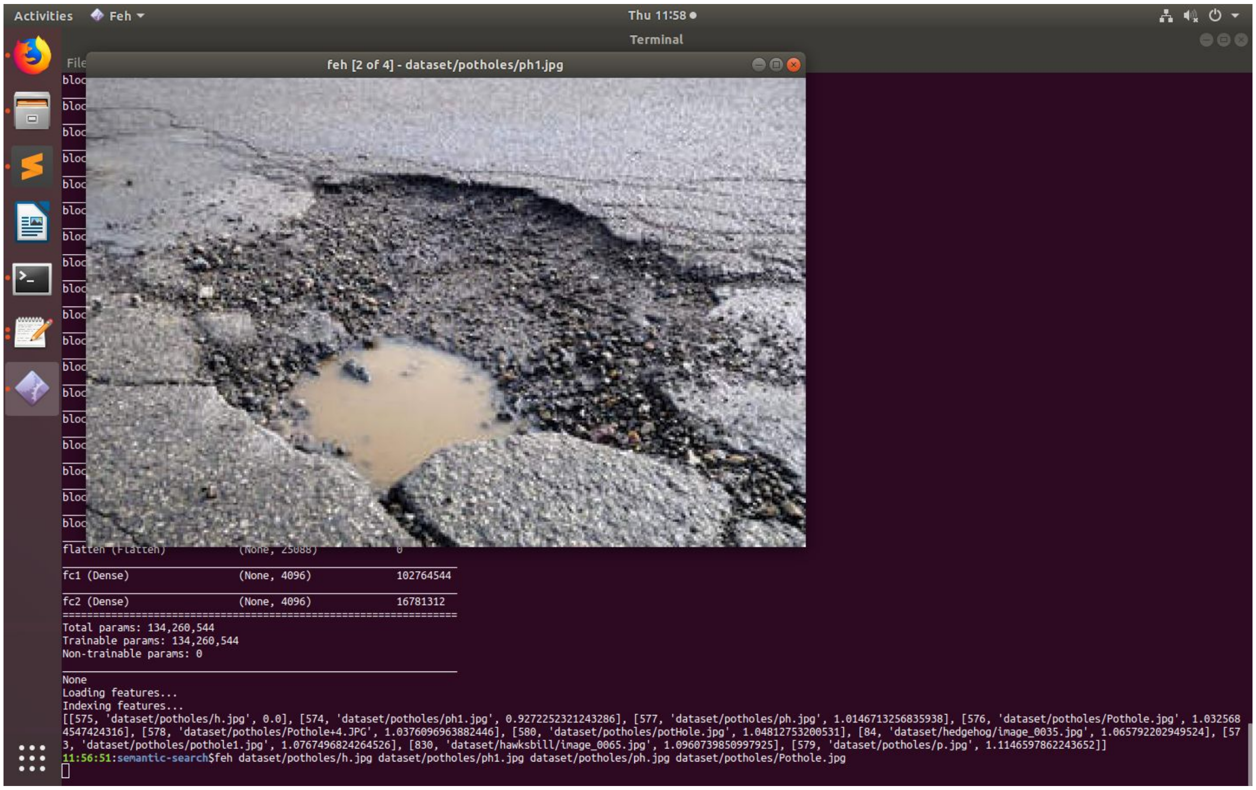
Finally, I attempted to code the ResNet50 model layer by layer, but this time it would be trained on MSCOCO dataset weights instead of the ImageNet weights. The codes will be uploaded to my Github repository.

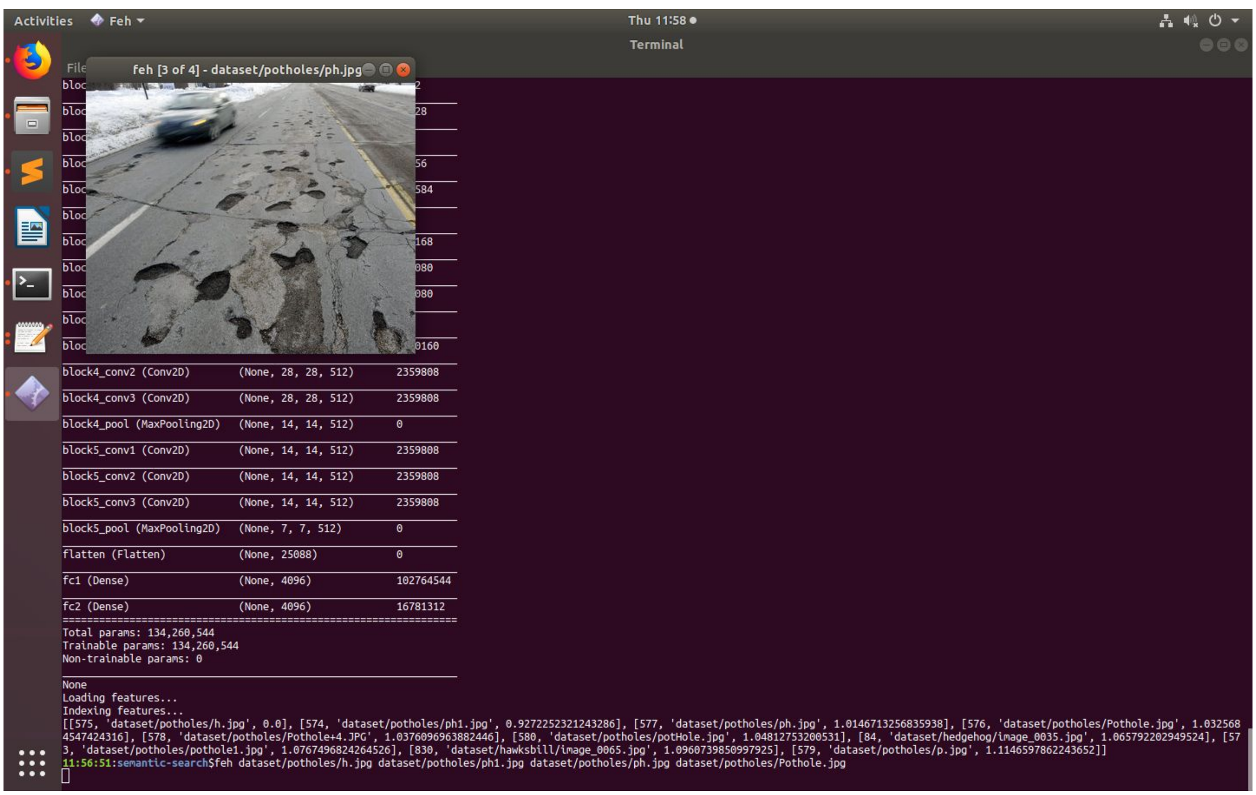
Certain references: The complete Mask R-CNN model has been implemented in-- https://github.com/matterport/Mask_RCNN The explanation of the entire methodology has been given in:-- https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b4

Results

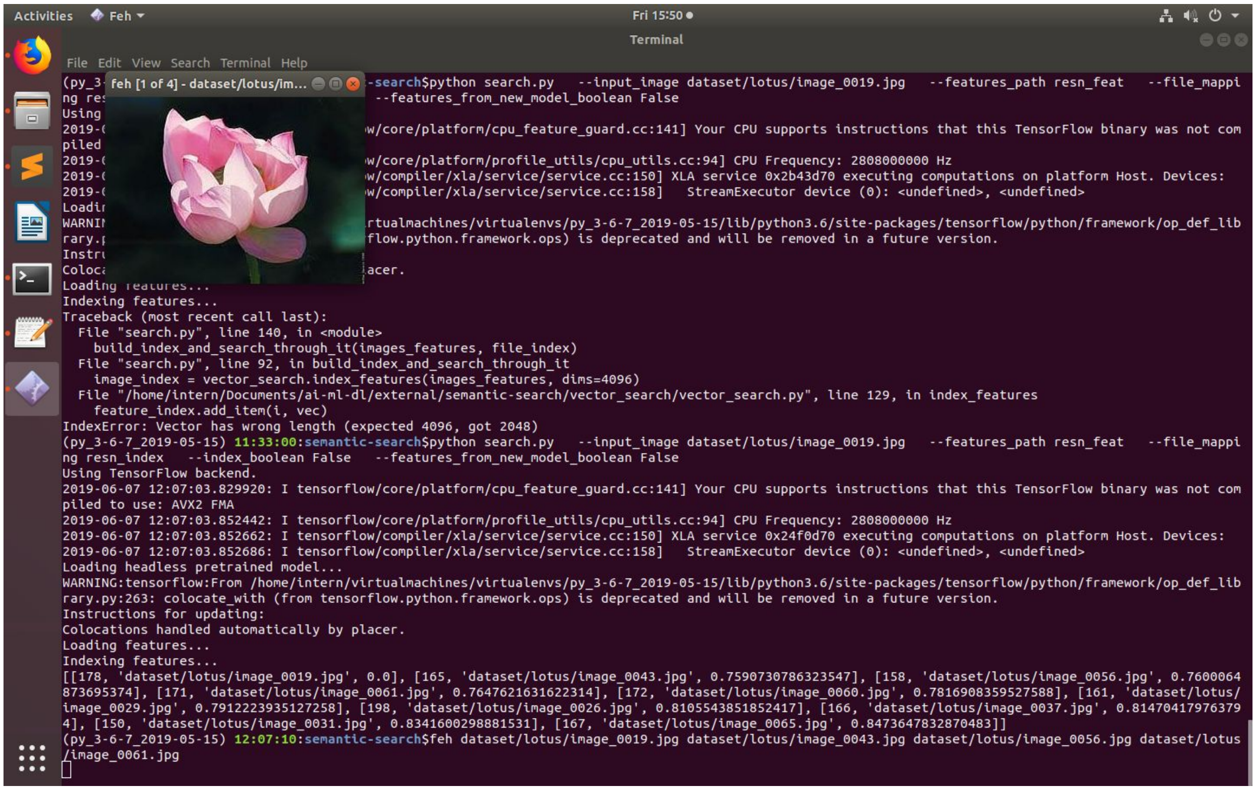
When the dataset was changed to include a folder containing pothole images, we tried to query an image from that folder so that similar images may come up. The top 3 results were as follows:

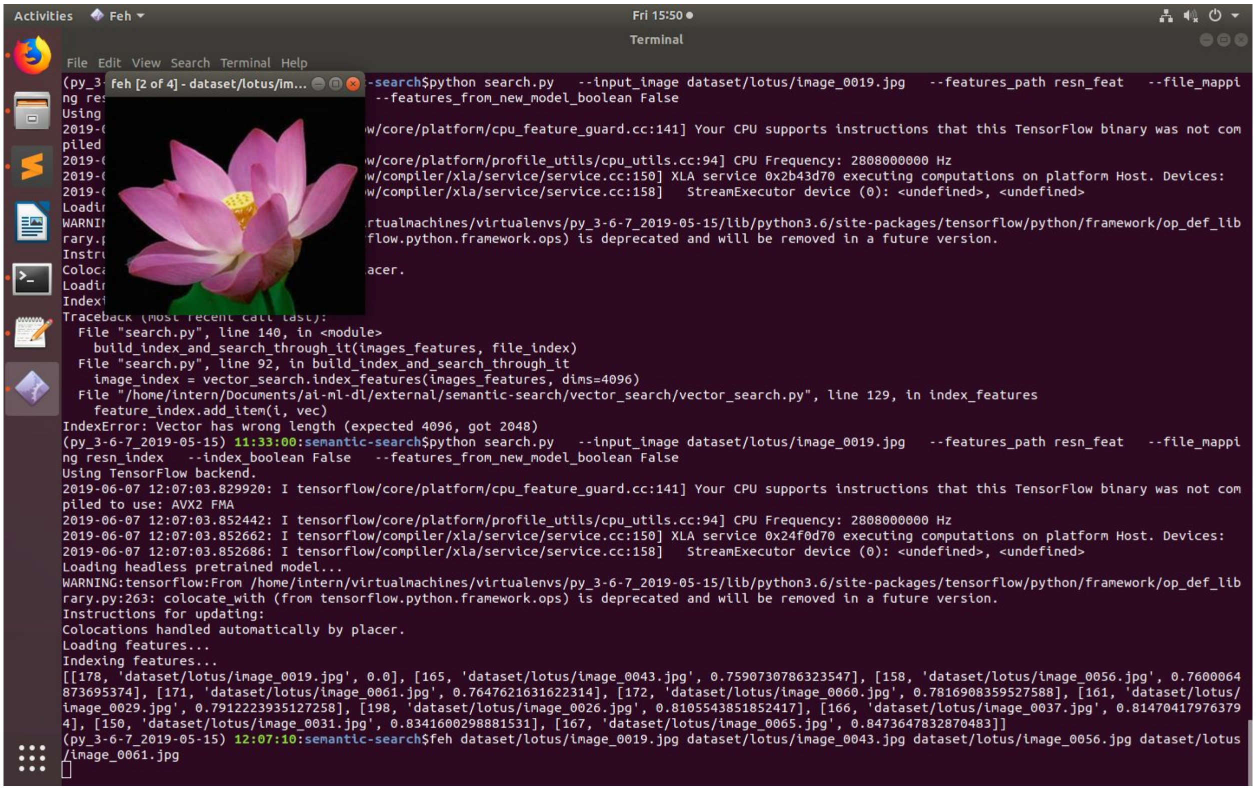



Activities
Feh
Thu 11:58
Terminal
feh [2 of 4] - dataset/potholes/ph1.jpg
fc1 (Dense) (None, 4096) 102764544
fc2 (Dense) (None, 4096) 16781312
Total params: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0
None
Loading features...
Indexing features...
[[575, 'dataset/potholes/h.jpg', 0.0], [574, 'dataset/potholes/ph1.jpg', 0.9272252321243286], [577, 'dataset/potholes/ph.jpg', 1.0146713256835938], [576, 'dataset/potholes/Pothole.jpg', 1.0325684547424316], [578, 'dataset/potholes/Pothole+4.JPG', 1.0376096963882446], [580, 'dataset/potholes/potHole.jpg', 1.04812753200531], [84, 'dataset/hedgehog/image_0035.jpg', 1.065792202949524], [573, 'dataset/potholes/pothole1.jpg', 1.0767496824264526], [830, 'dataset/hawksbill/image_0065.jpg', 1.0960739850997925], [579, 'dataset/potholes/p.jpg', 1.1146597862243652]]
11:56:51:semantic-search$feh dataset/potholes/h.jpg dataset/potholes/ph1.jpg dataset/potholes/ph.jpg dataset/potholes/Pothole.jpg

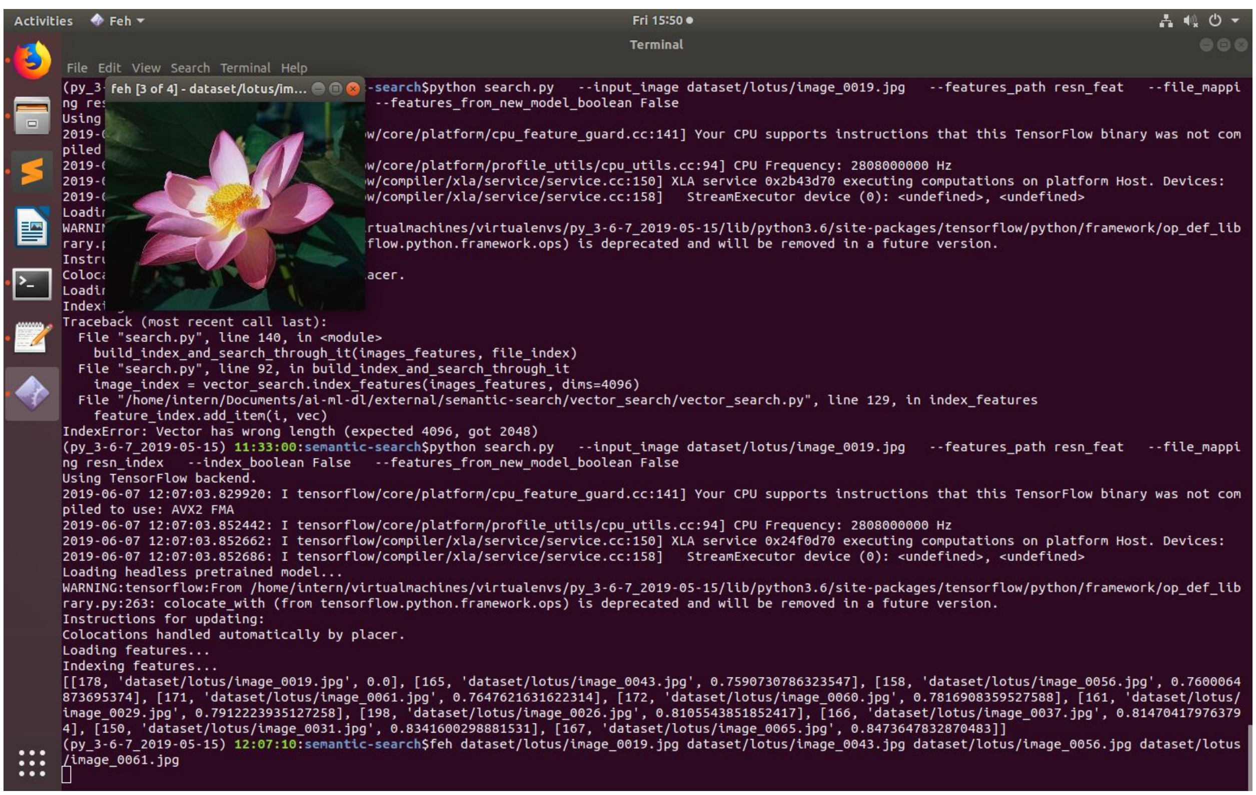

Activities
Feh
Thu 11:58
Terminal
feh [3 of 4] - dataset/potholes/ph.jpg
block4_conv2 (Conv2D) (None, 28, 28, 512) 2359808
block4_conv3 (Conv2D) (None, 28, 28, 512) 2359808
block4_pool (MaxPooling2D) (None, 14, 14, 512) 0
block5_conv1 (Conv2D) (None, 14, 14, 512) 2359808
block5_conv2 (Conv2D) (None, 14, 14, 512) 2359808
block5_conv3 (Conv2D) (None, 14, 14, 512) 2359808
block5_pool (MaxPooling2D) (None, 7, 7, 512) 0
flatten (Flatten) (None, 25088) 0
fc1 (Dense) (None, 4096) 102764544
fc2 (Dense) (None, 4096) 16781312
Total params: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0
None
Loading features...
Indexing features...
[[575, 'dataset/potholes/h.jpg', 0.0], [574, 'dataset/potholes/ph1.jpg', 0.9272252321243286], [577, 'dataset/potholes/ph.jpg', 1.0146713256835938], [576, 'dataset/potholes/Pothole.jpg', 1.0325684547424316], [578, 'dataset/potholes/Pothole+4.JPG', 1.0376096963882446], [580, 'dataset/potholes/potHole.jpg', 1.04812753200531], [84, 'dataset/hedgehog/image_0035.jpg', 1.065792202949524], [573, 'dataset/potholes/pothole1.jpg', 1.0767496824264526], [830, 'dataset/hawksbill/image_0065.jpg', 1.0960739850997925], [579, 'dataset/potholes/p.jpg', 1.1146597862243652]]
11:56:51:semantic-search$feh dataset/potholes/h.jpg dataset/potholes/ph1.jpg dataset/potholes/ph.jpg dataset/potholes/Pothole.jpg

Next, the pretrained model was used to index (indexed to resn_feat.npy and resn_index.json) and search images, the top 4 results are displayed as follows:

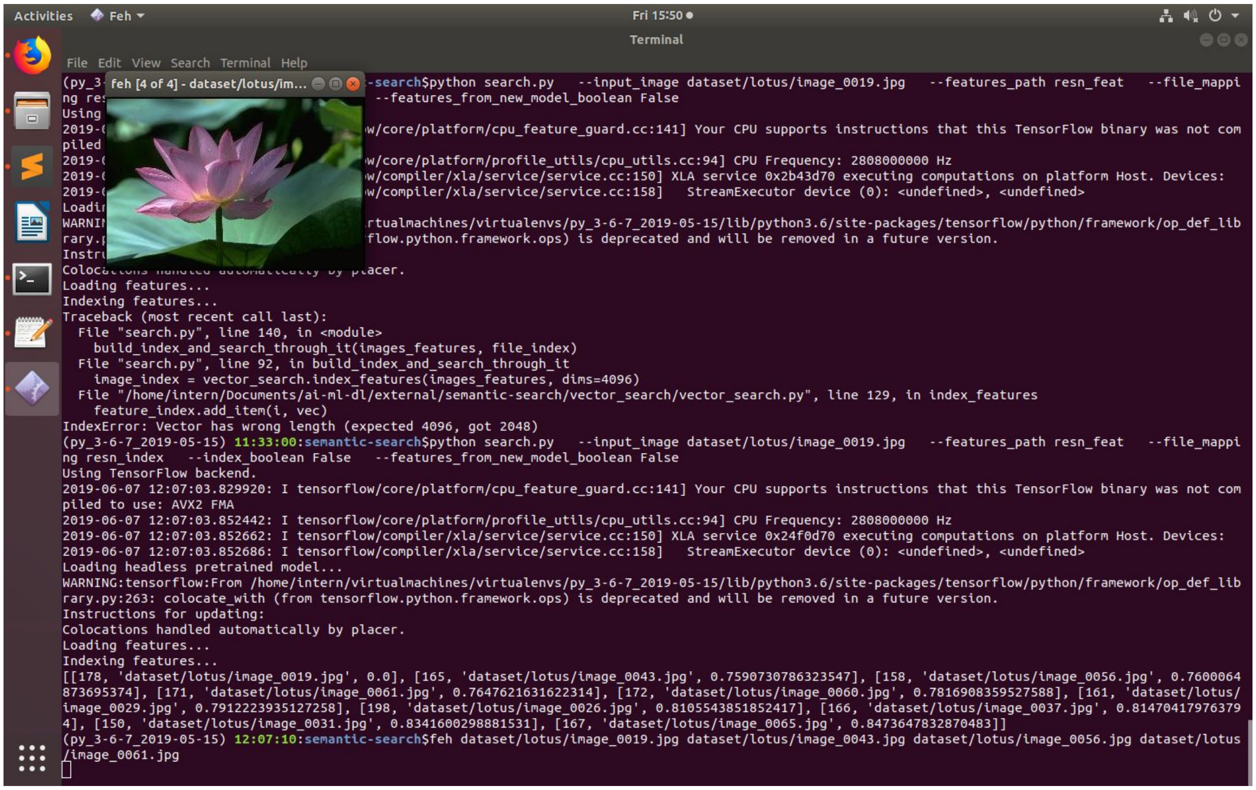




Activities
Feh
Fri 15:50
Terminal
File Edit View Search Terminal Help
feh [2 of 4] - dataset/lotus/im...
File "search.py", line 140, in <module>
build_index_and_search_through_it(images_features, file_index)
File "search.py", line 92, in build_index_and_search_through_it
image_index = vector_search.index_features(images_features, dims=4096)
File "/home/intern/Documents/ai-ml-dl/external/semantic-search/vector_search/vector_search.py", line 129, in index_features
feature_index.add_item(i, vec)
IndexError: Vector has wrong length (expected 4096, got 2048)
(py_3-6-7_2019-05-15) 11:33:00:semantic-search$python search.py --input_image dataset/lotus/image_0019.jpg --features_path resn_feat --file_mapping resn_index --index_boolean False --features_from_new_model_boolean False
Using TensorFlow backend.
2019-06-07 12:07:03.829920: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
2019-06-07 12:07:03.852442: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency: 2808000000 Hz
2019-06-07 12:07:03.852662: I tensorflow/compiler/xla/service/service.cc:150] XLA service 0x24f0d70 executing computations on platform Host. Devices:
2019-06-07 12:07:03.852686: I tensorflow/compiler/xla/service/service.cc:158] StreamExecutor device (0): <undefined>, <undefined>
Loading headless pretrained model...
WARNING:tensorflow:From /home/intern/virtualmachines/virtualenvs/py_3-6-7_2019-05-15/lib/python3.6/site-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
Loading features...
Indexing features...
[[178, 'dataset/lotus/image_0019.jpg', 0.0], [165, 'dataset/lotus/image_0043.jpg', 0.7590730786323547], [158, 'dataset/lotus/image_0056.jpg', 0.7600064873695374], [171, 'dataset/lotus/image_0061.jpg', 0.7647621631622314], [172, 'dataset/lotus/image_0060.jpg', 0.7816908359527588], [161, 'dataset/lotus/image_0029.jpg', 0.7912223935127258], [198, 'dataset/lotus/image_0026.jpg', 0.8105543851852417], [166, 'dataset/lotus/image_0037.jpg', 0.814704179763794], [150, 'dataset/lotus/image_0031.jpg', 0.8341600298881531], [167, 'dataset/lotus/image_0065.jpg', 0.8473647832870483]]
(py_3-6-7_2019-05-15) 12:07:10:semantic-search$feh dataset/lotus/image_0019.jpg dataset/lotus/image_0043.jpg dataset/lotus/image_0056.jpg dataset/lotus/image_0061.jpg



Activities
Feh
Fri 15:50
Terminal
File Edit View Search Terminal Help
feh [3 of 4] - dataset/lotus/im...
Traceback (most recent call last):
File "search.py", line 140, in <module>
build_index_and_search_through_it(images_features, file_index)
File "search.py", line 92, in build_index_and_search_through_it
image_index = vector_search.index_features(images_features, dims=4096)
File "/home/intern/Documents/ai-ml-dl/external/semantic-search/vector_search/vector_search.py", line 129, in index_features
feature_index.add_item(i, vec)
IndexError: Vector has wrong length (expected 4096, got 2048)
(py_3-6-7_2019-05-15) 11:33:00:semantic-search$python search.py --input_image dataset/lotus/image_0019.jpg --features_path resn_feat --file_mapping resn_index --index_boolean False --features_from_new_model_boolean False
Using TensorFlow backend.
2019-06-07 12:07:03.829920: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
2019-06-07 12:07:03.852442: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency: 2808000000 Hz
2019-06-07 12:07:03.852662: I tensorflow/compiler/xla/service/service.cc:150] XLA service 0x24f0d70 executing computations on platform Host. Devices:
2019-06-07 12:07:03.852686: I tensorflow/compiler/xla/service/service.cc:158] StreamExecutor device (0): <undefined>, <undefined>
Loading headless pretrained model...
WARNING:tensorflow:From /home/intern/virtualmachines/virtualenvs/py_3-6-7_2019-05-15/lib/python3.6/site-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
Loading features...
Indexing features...
[[178, 'dataset/lotus/image_0019.jpg', 0.0], [165, 'dataset/lotus/image_0043.jpg', 0.7590730786323547], [158, 'dataset/lotus/image_0056.jpg', 0.7600064873695374], [171, 'dataset/lotus/image_0061.jpg', 0.7647621631622314], [172, 'dataset/lotus/image_0060.jpg', 0.7816908359527588], [161, 'dataset/lotus/image_0029.jpg', 0.7912223935127258], [198, 'dataset/lotus/image_0026.jpg', 0.8105543851852417], [166, 'dataset/lotus/image_0037.jpg', 0.814704179763794], [150, 'dataset/lotus/image_0031.jpg', 0.8341600298881531], [167, 'dataset/lotus/image_0065.jpg', 0.8473647832870483]]
(py_3-6-7_2019-05-15) 12:07:10:semantic-search$feh dataset/lotus/image_0019.jpg dataset/lotus/image_0043.jpg dataset/lotus/image_0056.jpg dataset/lotus/image_0061.jpg



Discussion and Future Work

Future work would include trying to save, index and search images having more than one object. That would require object detection rather than simple classification and saving features of all objects so that they can be queried accordingly.

Conclusion

Bibliography

(add research paper ref.s)